made with doc-o-matic

RAD Studio VCL Reference

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

Table of Contents

ExptIntf Namespace 1

Classes **3**

TIExpert Class 3

TIExpert Members 3

TIExpert Methods 3

Functions **7**

ExptIntf.RegisterLibraryExpert Function 7

Structs, Records, Enums **8**

ExptIntf.TExpertStyle Enumeration 8

Types **9**

ExptIntf.TExpertInitProc Type 9

ExptIntf.TExpertRegisterProc Type 9

ExptIntf.TExpertState Type 10

ExptIntf.TExpertTerminateProc Type 10

Variables **11**

ExptIntf.LibraryExpertProc Variable 11

ExptIntf.ToolServices Variable 11

Constants **12**

ExptIntf.ExpertEntryPoint Constant 12

ExptIntf.isExperts Constant 12

ExptIntf.ValidExpertVersion Constant 12

ExtActns Namespace 15

Classes **17**

TCustomFileRun Class 17

TCustomFileRun Members 18

TCustomFileRun.Create Constructor 19

TCustomFileRun Methods 19

TCustomFileRun Properties 22

TFileRun Class 27

TFileRun Members 28

TFileRun Properties 29

TFileRun Events 36

TListControlAction Class 37
TListControlAction Members 38
TListControlAction.Create Constructor 38
TListControlAction Methods 39
TListControlAction Properties 40
TListControlClearSelection Class 41
TListControlClearSelection Members 41
TListControlClearSelection Methods 42
TListControlClearSelection Properties 43
TListControlClearSelection Events 48
TListControlCopySelection Class 50
TListControlCopySelection Members 50
TListControlCopySelection Methods 51
TListControlCopySelection Properties 53
TListControlCopySelection Events 58
TListControlDeleteSelection Class 60
TListControlDeleteSelection Members 60
TListControlDeleteSelection Methods 61
TListControlDeleteSelection Properties 62
TListControlDeleteSelection Events 67
TListControlMoveSelection Class 68
TListControlMoveSelection Members 69
TListControlMoveSelection Methods 70
TListControlSelectAll Class 71
TListControlSelectAll Members 71
TListControlSelectAll Methods 72
TListControlSelectAll Properties 73
TListControlSelectAll Events 78
TNextTab Class 80
TNextTab Members 80
TNextTab Methods 81
TNextTab Properties 83
TNextTab Events 88
TOpenPicture Class 92
TOpenPicture Members 93
TOpenPicture Methods 93
TOpenPicture Properties 94
TOpenPicture Events 98
TPreviousTab Class 101

TPreviousTab Members 101
TPreviousTab Methods 102
TPreviousTab Properties 103
TPreviousTab Events 109
TRichEditAction Class 112
TRichEditAction Members 113
TRichEditAction.Create Constructor 113
TRichEditAction Methods 113
TRichEditAlignCenter Class 115
TRichEditAlignCenter Members 115
TRichEditAlignCenter Methods 116
TRichEditAlignLeft Class 117
TRichEditAlignLeft Members 117
TRichEditAlignLeft Methods 118
TRichEditAlignRight Class 119
TRichEditAlignRight Members 120
TRichEditAlignRight Methods 120
TRichEditBold Class 121
TRichEditBold Members 122
TRichEditBold Methods 122
TRichEditBullets Class 123
TRichEditBullets Members 124
TRichEditBullets Methods 124
TRichEditItalic Class 125
TRichEditItalic Members 126
TRichEditItalic Methods 126
TRichEditStrikeOut Class 127
TRichEditStrikeOut Members 128
TRichEditStrikeOut Methods 128
TRichEditUnderline Class 130
TRichEditUnderline Members 130
TRichEditUnderline Methods 131
TSavePicture Class 132
TSavePicture Members 133
TSavePicture Methods 133
TSavePicture Properties 134
TSavePicture Events 138
TSendMail Class 141
TSendMail Members 141

TSendMail.Destroy Destructor 142
TSendMail Methods 142
TSendMail Properties 143
TSendMail Events 148
TTabAction Class 150
TTabAction Members 150
TTabAction.Create Constructor 151
TTabAction Methods 151
TTabAction Properties 153
TTabAction Events 155

Structs, Records, Enums **157**
ExtActns.TShowCmd Enumeration 157

Types **158**
ExtActns.TValidateTabEvent Type 158

ExtCtrls Namespace 159

Classes **162**
EGridPanelException Class 163
TBevel Class 163
TBevel Members 164
TBevel.Create Constructor 164
TBevel Methods 164
TBevel Properties 165
TBoundLabel Class 174
TBoundLabel Members 175
TBoundLabel.Create Constructor 176
TBoundLabel Methods 176
TBoundLabel Properties 177
TBoundLabel Events 200
TButtonedEdit Class 220
TButtonedEdit Members 224
TButtonedEdit Properties 226
TButtonedEdit Events 270
TCategoryPanel Class 309
TCategoryPanel Members 309
TCategoryPanel Properties 312
TCategoryPanel Events 344
TCategoryPanelGroup Class 379

TCategoryPanelGroup Members 379
TCategoryPanelGroup Properties 382
TCategoryPanelGroup Events 418
TCategoryPanelSurface Class 456
TCategoryPanelSurface Members 456
TCategoryPanelSurface.Create Constructor 456
TCategoryPanelSurface Methods 457
TCellCollection Class 458
TCellCollection Members 458
TCellCollection Methods 458
TCellCollection Properties 461
TCellItem Class 462
TCellItem Members 462
TCellItem.Create Constructor 462
TCellItem Methods 463
TCellItem Properties 464
TColorBox Class 466
TColorBox Members 466
TColorBox Properties 468
TColorBox Events 496
TColorListBox Class 525
TColorListBox Members 526
TColorListBox Properties 527
TColorListBox Events 554
TColumnCollection Class 587
TColumnCollection Members 588
TColumnCollection.Create Constructor 588
TColumnCollection Methods 588
TColumnItem Class 590
TControlBar Class 590
TControlBar Members 591
TControlBar Properties 593
TControlBar Events 622
TControlCollection Class 663
TControlCollection Members 663
TControlCollection.Create Constructor 664
TControlCollection Methods 664
TControlCollection Properties 667
TControlItem Class 668

TControlItem Members 669
TControlItem.Create Constructor 669
TControlItem.Destroy Destructor 669
TControlItem Methods 670
TControlItem Properties 671
TCustomButtonedEdit Class 673
TCustomButtonedEdit Members 674
TCustomButtonedEdit.Create Constructor 674
TCustomButtonedEdit.Destroy Destructor 675
TCustomButtonedEdit Methods 675
TCustomButtonedEdit Properties 682
TCustomButtonedEdit Events 691
TCustomCategoryPanel Class 698
TCustomCategoryPanel Members 698
TCustomCategoryPanel.Create Constructor 700
TCustomCategoryPanel.Destroy Destructor 700
TCustomCategoryPanel Enumerations 701
TCustomCategoryPanel Fields 702
TCustomCategoryPanel Methods 702
TCustomCategoryPanel Properties 714
TCustomCategoryPanel Events 718
TCustomCategoryPanelGroup Class 718
TCustomCategoryPanelGroup Members 719
TCustomCategoryPanelGroup.Create Constructor 720
TCustomCategoryPanelGroup.Destroy Destructor 721
TCustomCategoryPanelGroup Enumerations 721
TCustomCategoryPanelGroup Methods 722
TCustomCategoryPanelGroup Properties 729
TCustomColorBox Class 735
TCustomColorBox Members 736
TCustomColorBox.Create Constructor 737
TCustomColorBox Methods 737
TCustomColorBox Properties 741
TCustomColorBox Events 744
TCustomColorListBox Class 745
TCustomColorListBox Members 745
TCustomColorListBox.Create Constructor 746
TCustomColorListBox Methods 746
TCustomColorListBox Properties 749

TCustomColorListBox Events 753
TCustomControlBar Class 754
TCustomControlBar Members 754
TCustomControlBar.Create Constructor 755
TCustomControlBar.Destroy Destructor 756
TCustomControlBar Methods 756
TCustomControlBar Properties 765
TCustomControlBar Events 771
TCustomFlowPanel Class 774
TCustomFlowPanel Members 774
TCustomFlowPanel.Create Constructor 775
TCustomFlowPanel.Destroy Destructor 775
TCustomFlowPanel Methods 776
TCustomFlowPanel Properties 777
TCustomGridPanel Class 781
TCustomGridPanel Members 782
TCustomGridPanel.Create Constructor 783
TCustomGridPanel.Destroy Destructor 783
TCustomGridPanel Methods 784
TCustomGridPanel Properties 788
TCustomLabeledEdit Class 790
TCustomLabeledEdit Members 791
TCustomLabeledEdit.Create Constructor 791
TCustomLabeledEdit Methods 792
TCustomLabeledEdit Properties 795
TCustomLinkLabel Class 797
TCustomLinkLabel Members 797
TCustomLinkLabel.Create Constructor 798
TCustomLinkLabel Methods 798
TCustomLinkLabel Properties 800
TCustomLinkLabel Nested Types 801
TCustomLinkLabel Events 801
TCustomPanel Class 802
TCustomPanel Members 803
TCustomPanel.Create Constructor 803
TCustomPanel Methods 804
TCustomPanel Properties 806
TCustomRadioGroup Class 815
TCustomRadioGroup Members 815

TCustomRadioGroup.Create Constructor 816
TCustomRadioGroup.Destroy Destructor 816
TCustomRadioGroup Methods 817
TCustomRadioGroup Properties 819
TCustomTrayIcon Class 822
TCustomTrayIcon Members 823
TCustomTrayIcon.Create Constructor 824
TCustomTrayIcon.Destroy Destructor 824
TCustomTrayIcon Methods 824
TCustomTrayIcon Properties 830
TCustomTrayIcon Events 833
TEditButton Class 835
TEditButton Members 839
TEditButton.Create Constructor 839
TEditButton.Destroy Destructor 840
TEditButton Enumerations 840
TEditButton Methods 841
TEditButton Properties 842
TFlowPanel Class 857
TFlowPanel Members 860
TFlowPanel Properties 862
TFlowPanel Events 903
TGridPanel Class 941
TGridPanel Members 941
TGridPanel Properties 944
TGridPanel Events 983
THeader Class 1021
THeader Members 1021
THeader.Create Constructor 1022
THeader.Destroy Destructor 1023
THeader Methods 1023
THeader Properties 1026
THeader Events 1046
TImage Class 1048
TImage Members 1049
TImage.Create Constructor 1050
TImage.Destroy Destructor 1050
TImage Methods 1051
TImage Properties 1053

TImage Events 1079
TLabeledEdit Class 1100
TLabeledEdit Members 1100
TLabeledEdit Properties 1102
TLabeledEdit Events 1134
TLinkLabel Class 1167
TLinkLabel Members 1167
TLinkLabel Properties 1169
TLinkLabel Events 1191
TNotebook Class 1211
TNotebook Members 1212
TNotebook.Create Constructor 1213
TNotebook.Destroy Destructor 1214
TNotebook Methods 1214
TNotebook Properties 1216
TNotebook Events 1241
TPage Class 1267
TPage Members 1268
TPage.Create Constructor 1268
TPage Methods 1269
TPage Properties 1270
TPage Events 1278
TPaintBox Class 1279
TPaintBox Members 1280
TPaintBox.Create Constructor 1281
TPaintBox Methods 1281
TPaintBox Properties 1282
TPaintBox Events 1303
TPanel Class 1326
TPanel Members 1327
TPanel Properties 1329
TPanel Events 1368
TRadioGroup Class 1406
TRadioGroup Members 1406
TRadioGroup Properties 1408
TRadioGroup Events 1438
TRowCollection Class 1455
TRowCollection Members 1455
TRowCollection.Create Constructor 1456

TRowCollection Methods 1456
TRowItem Class 1457
TShape Class 1458
TShape Members 1458
TShape.Create Constructor 1459
TShape.Destroy Destructor 1460
TShape Methods 1460
TShape Properties 1461
TShape Events 1474
TSplitter Class 1492
TSplitter Members 1492
TSplitter.Create Constructor 1493
TSplitter.Destroy Destructor 1495
TSplitter Methods 1495
TSplitter Properties 1499
TSplitter Events 1517
TTimer Class 1520
TTimer Members 1521
TTimer.Create Constructor 1521
TTimer.Destroy Destructor 1522
TTimer Methods 1522
TTimer Properties 1522
TTimer Events 1527
TTrayIcon Class 1528
TTrayIcon Members 1528
TTrayIcon Properties 1530
TTrayIcon Events 1533

Functions **1536**
ExtCtrls.Frame3D Function 1536
ExtCtrls.NotebookHandlesNeeded Function 1537

Structs, Records, Enums **1538**
ExtCtrls.TBalloonFlags Enumeration 1538
ExtCtrls.TBandDrawingStyle Enumeration 1539
ExtCtrls.TBandPaintOption Enumeration 1539
ExtCtrls.TBevelShape Enumeration 1540
ExtCtrls.TBevelStyle Enumeration 1541
ExtCtrls.TColorBoxStyles Enumeration 1541
ExtCtrls.TCornerEdge Enumeration 1542

ExtCtrls.TExpandStyle Enumeration 1542
ExtCtrls.TFlowStyle Enumeration 1543
ExtCtrls.TLabelPosition Enumeration 1543
ExtCtrls.TResizeStyle Enumeration 1544
ExtCtrls.TShapeType Enumeration 1544
ExtCtrls.TSizeStyle Enumeration 1545
ExtCtrls.TSysLinkType Enumeration 1546

Types **1547**
ExtCtrls.NaturalNumber Type 1547
ExtCtrls.TBandDragEvent Type 1548
ExtCtrls.TBandInfoEvent Type 1548
ExtCtrls.TBandMoveEvent Type 1548
ExtCtrls.TBandPaintEvent Type 1549
ExtCtrls.TBandPaintOptions Type 1549
ExtCtrls.TBeginBandMoveEvent Type 1549
ExtCtrls.TCanResizeEvent Type 1550
ExtCtrls.TCategoryPanelClass Type 1550
ExtCtrls.TCategoryPanelSurfaceClass Type 1550
ExtCtrls.TCellSpan Type 1551
ExtCtrls.TColorBoxStyle Type 1551
ExtCtrls.TEditButtonClass Type 1552
ExtCtrls.TEndBandMoveEvent Type 1552
ExtCtrls.TGetColorsEvent Type 1552
ExtCtrls.TLBGetColorsEvent Type 1553
ExtCtrls.TPanelBevel Type 1553
ExtCtrls.TRowSize Type 1554
ExtCtrls.TSectionEvent Type 1554
ExtCtrls.TSplitterCanResizeEvent Type 1554
ExtCtrls.TSysLinkEvent Type 1555

Constants **1556**
ExtCtrls.DefaultCategoryPanelHeight Constant 1556
ExtCtrls.NoColorSelected Constant 1556
ExtCtrls.WM_SYSTEM_TRAY_MESSAGE Constant 1556

ExtDlgs Namespace 1559

Classes **1560**
TOpenPictureDialog Class 1560
TOpenPictureDialog Members 1561

TOpenPictureDialog.Create Constructor 1562
TOpenPictureDialog Methods 1562
TOpenPictureDialog Properties 1565
TOpenTextFileDialog Class 1570
TOpenTextFileDialog Members 1570
TOpenTextFileDialog.Create Constructor 1571
TOpenTextFileDialog.Destroy Destructor 1571
TOpenTextFileDialog Methods 1571
TOpenTextFileDialog Properties 1576
TSavePictureDialog Class 1577
TSavePictureDialog Members 1578
TSavePictureDialog Methods 1578
TSaveTextFileDialog Class 1580
TSaveTextFileDialog Members 1580
TSaveTextFileDialog Methods 1580

Functions **1585**
ExtDlgs.StandardEncodingFromName Function 1585

Constants **1586**
ExtDlgs.DefaultEncodingNames Constant 1586

FileCtrl Namespace 1587

Classes **1589**
TDirectoryListBox Class 1589
TDirectoryListBox Members 1590
TDirectoryListBox.Create Constructor 1592
TDirectoryListBox.Destroy Destructor 1593
TDirectoryListBox Fields 1593
TDirectoryListBox Methods 1594
TDirectoryListBox Properties 1601
TDirectoryListBox Events 1635
TDriveComboBox Class 1668
TDriveComboBox Members 1669
TDriveComboBox.Create Constructor 1671
TDriveComboBox.Destroy Destructor 1671
TDriveComboBox Fields 1672
TDriveComboBox Methods 1673
TDriveComboBox Properties 1675
TDriveComboBox Events 1707

TFileListBox Class 1734
TFileListBox Members 1736
TFileListBox.Create Constructor 1738
TFileListBox.Destroy Destructor 1738
TFileListBox Fields 1739
TFileListBox Methods 1742
TFileListBox Properties 1747
TFileListBox Events 1781
TFilterComboBox Class 1813
TFilterComboBox Members 1813
TFilterComboBox.Create Constructor 1815
TFilterComboBox.Destroy Destructor 1815
TFilterComboBox Methods 1816
TFilterComboBox Properties 1818
TFilterComboBox Events 1850

Functions **1879**
FileCtrl.DirectoryExists Function 1879
FileCtrl.ForceDirectories Function 1879
FileCtrl.MinimizeName Function 1880
FileCtrl.ProcessPath Function 1881
FileCtrl.SelectDirectory Function 1881

Structs, Records, Enums **1883**
FileCtrl.TDriveType Enumeration 1883
FileCtrl.TFileAttr Enumeration 1883
FileCtrl.TSelectDirExtOpt Enumeration 1884
FileCtrl.TSelectDirOpt Enumeration 1885
FileCtrl.TTextCase Enumeration 1886

Types **1887**
FileCtrl.TFileType Type 1887
FileCtrl.TSelectDirExtOpts Type 1887
FileCtrl.TSelectDirOpts Type 1888

Constants **1889**
FileCtrl.WNTYPE_DRIVE Constant 1889

FMTBcd Namespace 1891

Classes **1894**
EBcdException Class 1894

EBcdOverflowException Class 1894

Functions **1896**

FMTBcd.BcdAdd Function 1897
FMTBcd.BcdCompare Function 1897
FMTBcd.BcdDivide Function 1898
FMTBcd.BcdFromBytes Function 1899
FMTBcd.BcdMultiply Function 1899
FMTBcd.BcdPrecision Function 1900
FMTBcd.BcdScale Function 1900
FMTBcd.BcdSubtract Function 1901
FMTBcd.BcdToBytes Function 1901
FMTBcd.BCDToCurr Function 1901
FMTBcd.BcdToDouble Function 1902
FMTBcd.BcdToInteger Function 1902
FMTBcd.BcdToStr Function 1903
FMTBcd.BcdToStrF Function 1903
FMTBcd.CurrToBCD Function 1904
FMTBcd.DoubleToBcd Function 1905
FMTBcd.FormatBcd Function 1905
FMTBcd.IntegerToBcd Function 1906
FMTBcd.IsBcdNegative Function 1906
FMTBcd.NormalizeBcd Function 1907
FMTBcd.OldBCDToCurr Function 1907
FMTBcd.OldCurrToBCD Function 1908
FMTBcd.RoundAt Function 1908
FMTBcd.StrToBcd Function 1908
FMTBcd.TryStrToBcd Function 1909
FMTBcd.VarFMTBcd Function 1910
FMTBcd.VarFMTBcdCreate Function 1910
FMTBcd.VarIsFMTBcd Function 1911
FMTBcd.VarToBcd Function 1911

Structs, Records, Enums **1913**

FMTBcd.TBcd Record 1913

Types **1914**

FMTBcd.PBcd Type 1914
FMTBcd.TNibbleArray Type 1914

Constants **1915**

FMTBcd._DefaultDecimals Constant 1915
FMTBcd._NoDecimal Constant 1915
FMTBcd.DefaultFMTBcdScale Constant 1916
FMTBcd.MaxBcdPrecision Constant 1916
FMTBcd.MaxBcdScale Constant 1916
FMTBcd.MaxFMTBcdDigits Constant 1917
FMTBcd.MaxFMTBcdFractionSize Constant 1917
FMTBcd.MaxStringDigits Constant 1917
FMTBcd.NullBcd Constant 1918
FMTBcd.SizeOfFraction Constant 1918
FMTBcd.SizeOfTBCD Constant 1918

Forms Namespace 1921

Classes 1924
TApplication Class 1924
TApplication Members 1925
TApplication.Create Constructor 1928
TApplication.Destroy Destructor 1928
TApplication Methods 1929
TApplication Properties 1967
TApplication Events 1990
TControlScrollBar Class 2009
TControlScrollBar Members 2010
TControlScrollBar Methods 2010
TControlScrollBar Properties 2012
TCustomActiveForm Class 2024
TCustomActiveForm Members 2024
TCustomActiveForm.Create Constructor 2029
TCustomActiveForm Methods 2029
TCustomActiveForm Properties 2030
TCustomActiveForm Events 2057
TCustomDockForm Class 2092
TCustomDockForm Members 2092
TCustomDockForm.Create Constructor 2097
TCustomDockForm Methods 2097
TCustomDockForm Properties 2099
TCustomForm Class 2103
TCustomForm Members 2104

TCustomForm.Create Constructor 2108
TCustomForm.CreateNew Constructor 2108
TCustomForm.Destroy Destructor 2109
TCustomForm Fields 2109
TCustomForm Methods 2110
TCustomForm Properties 2142
TCustomForm Events 2195
TCustomFrame Class 2237
TCustomFrame Members 2238
TCustomFrame.Create Constructor 2238
TCustomFrame Methods 2239
TForm Class 2241
TForm Members 2242
TForm Methods 2246
TForm Properties 2250
TForm Events 2311
TFrame Class 2366
TFrame Members 2367
TFrame Properties 2369
TFrame Events 2396
TGlassFrame Class 2435
TGlassFrame Members 2435
TGlassFrame.Create Constructor 2436
TGlassFrame Methods 2436
TGlassFrame Properties 2438
TGlassFrame Events 2439
TMonitor Class 2439
TMonitor Members 2440
TMonitor Properties 2440
TScreen Class 2445
TScreen Members 2446
TScreen.Create Constructor 2447
TScreen.Destroy Destructor 2448
TScreen Methods 2448
TScreen Properties 2451
TScreen Events 2479
TScrollBox Class 2483
TScrollBox Members 2484
TScrollBox.Create Constructor 2486

TScrollBox Methods 2486
TScrollBox Properties 2487
TScrollBox Events 2516
TScrollingWinControl Class 2554
TScrollingWinControl Members 2555
TScrollingWinControl.Create Constructor 2555
TScrollingWinControl.Destroy Destructor 2556
TScrollingWinControl Methods 2556
TScrollingWinControl Properties 2561
TScrollingWinControl Events 2564

Interfaces 2566
IDesignerHook Interface 2566
IDesignerHook Members 2567
IDesignerHook Methods 2567
IDesignerHook Properties 2571
IOleForm Interface 2572
IOleForm Members 2572
IOleForm Methods 2572

Functions 2574
Forms.DisableTaskWindows Function 2574
Forms.DoneCtl3D Function 2574
Forms.EnableTaskWindows Function 2575
Forms.ForegroundTask Function 2575
Forms.GetParentForm Function 2576
Forms.InitCtl3D Function 2576
Forms.IsAccel Function 2576
Forms.KeyboardStateToShiftState Function 2577
Forms.KeyDataToShiftState Function 2578
Forms.KeysToShiftState Function 2578
Forms.RestoreFocusState Function 2579
Forms.SaveFocusState Function 2579
Forms.SetAutoSubClass Function 2579
Forms.Subclass3DDlg Function 2580
Forms.Subclass3DWnd Function 2580
Forms.ValidParentForm Function 2580

Structs, Records, Enums 2582
Forms.TActiveFormBorderStyle Enumeration 2582
Forms.TBorderIcon Enumeration 2583

Forms.TCloseAction Enumeration 2584
Forms.TCMHintShowPause Record 2584
Forms.TCursorRec Record 2585
Forms.TDefaultMonitor Enumeration 2585
Forms.TFormBorderStyle Enumeration 2586
Forms.TFormStyle Enumeration 2587
Forms.TMonitorDefaultTo Enumeration 2587
Forms.TPopupForm Record 2588
Forms.TPopupMode Enumeration 2588
Forms.TPopupWnd Record 2589
Forms.TPosition Enumeration 2590
Forms.TPrintScale Enumeration 2591
Forms.TScrollBarKind Enumeration 2591
Forms.TScrollBarStyle Enumeration 2592
Forms.TShowAction Enumeration 2592
Forms.TTileMode Enumeration 2593
Forms.TTimerMode Enumeration 2593
Forms.TWindowState Enumeration 2594

Types **2595**

Forms.PCursorRec Type 2595
Forms.PHintInfo Type 2596
Forms.TBorderIcons Type 2596
Forms.TBorderStyle Type 2597
Forms.TCloseEvent Type 2597
Forms.TCloseQueryEvent Type 2598
Forms.TCursorRecType Type 2598
Forms.TCustomFormClass Type 2599
Forms.TCustomFrameClass Type 2599
Forms.TExceptionEvent Type 2599
Forms.TFocusState Type 2600
Forms.TFormClass Type 2600
Forms.TFormState Type 2601
Forms.TGetHandleEvent Type 2601
Forms.THelpEvent Type 2602
Forms.THintInfo Type 2602
Forms.TIdleEvent Type 2604
Forms.TMessageEvent Type 2605
Forms.TPopupFormArray Type 2605
Forms.TPopupWndArray Type 2605

Forms.TScrollBarInc Type 2606
Forms.TSetLayeredWindowAttributes Type 2606
Forms.TSettingChangeEvent Type 2607
Forms.TShortCutEvent Type 2609
Forms.TShowHintEvent Type 2610
Forms.TTaskWindowList Type 2610
Forms.TWindowHook Type 2611

Variables **2612**
Forms.Application Variable 2612
Forms.Ctl3DBtnWndProc Variable 2614
Forms.Ctl3DCtlColorEx Variable 2614
Forms.Ctl3DDlgFramePaint Variable 2614
Forms.HintWindowClass Variable 2615
Forms.Screen Variable 2615
Forms.SetLayeredWindowAttributes Variable 2617

Graphics Namespace 2619

Classes **2627**
EInvalidGraphic Class 2627
EInvalidGraphicOperation Class 2628
TBitmap Class 2628
TBitmap Members 2629
TBitmap.Create Constructor 2631
TBitmap.Destroy Destructor 2632
TBitmap Methods 2632
TBitmap Properties 2650
TBitmapImage Class 2662
TBitmapImage Members 2662
TBitmapImage.Destroy Destructor 2663
TBitmapImage Methods 2663
TBrush Class 2664
TBrush Members 2664
TBrush.Create Constructor 2665
TBrush.Destroy Destructor 2665
TBrush Methods 2666
TBrush Properties 2668
TBrushRecall Class 2671
TBrushRecall Members 2672

TBrushRecall.Create Constructor 2672
TCanvas Class 2673
TCanvas Members 2673
TCanvas.Create Constructor 2675
TCanvas.Destroy Destructor 2678
TCanvas Methods 2678
TCanvas Properties 2716
TCanvas Events 2729
TFont Class 2730
TFont Members 2731
TFont.Create Constructor 2732
TFont.Destroy Destructor 2732
TFont Methods 2732
TFont Properties 2738
TFontRecall Class 2747
TFontRecall Members 2747
TFontRecall.Create Constructor 2747
TGraphic Class 2748
TGraphic Members 2749
TGraphic.Create Constructor 2750
TGraphic Methods 2750
TGraphic Properties 2760
TGraphic Events 2763
TGraphicsObject Class 2764
TGraphicsObject Members 2765
TGraphicsObject Methods 2765
TGraphicsObject Properties 2767
TGraphicsObject Events 2768
TIcon Class 2768
TIcon Members 2769
TIcon.Create Constructor 2770
TIcon.Destroy Destructor 2772
TIcon Methods 2772
TIcon Properties 2777
TIconImage Class 2778
TIconImage Members 2779
TIconImage.Destroy Destructor 2779
TIconImage Methods 2779
TMetafile Class 2780

TMetafile Members 2780
TMetafile.Create Constructor 2782
TMetafile.Destroy Destructor 2783
TMetafile Methods 2784
TMetafile Properties 2791
TMetafileCanvas Class 2794
TMetafileCanvas Members 2795
TMetafileCanvas.Create Constructor 2796
TMetafileCanvas.CreateWithComment Constructor 2798
TMetafileCanvas.Destroy Destructor 2799
TMetafileImage Class 2799
TMetafileImage Members 2800
TMetafileImage.Destroy Destructor 2800
TMetafileImage Methods 2800
TPen Class 2801
TPen Members 2801
TPen.Create Constructor 2802
TPen.Destroy Destructor 2802
TPen Methods 2803
TPen Properties 2806
TPenRecall Class 2811
TPenRecall Members 2811
TPenRecall.Create Constructor 2811
TPicture Class 2812
TPicture Members 2813
TPicture.Create Constructor 2814
TPicture.Destroy Destructor 2814
TPicture Methods 2814
TPicture Properties 2825
TPicture Events 2829
TSharedImage Class 2830
TSharedImage Members 2831
TSharedImage.Destroy Destructor 2831
TSharedImage Methods 2831
TSharedImage Properties 2832

Interfaces 2834

IChangeNotifier Interface 2834
IChangeNotifier Members 2834
IChangeNotifier Methods 2834

Functions **2836**
Graphics.AllocPatternBitmap Function 2836
Graphics.BytesPerScanline Function 2837
Graphics.CharsetToIdent Function 2837
Graphics.ColorToIdent Function 2838
Graphics.ColorToRGB Function 2838
Graphics.ColorToString Function 2840
Graphics.CopyPalette Function 2841
Graphics.CreateGrayMappedBmp Function 2841
Graphics.CreateGrayMappedRes Function 2842
Graphics.CreateMappedBmp Function 2842
Graphics.CreateMappedRes Function 2843
Graphics.FreeMemoryContexts Function 2844
Graphics.GetCharsetValues Function 2844
Graphics.GetColorValues Function 2848
Graphics.GetDefFontCharSet Function 2848
Graphics.GetDIB Function 2849
Graphics.GetDIBSizes Function 2849
Graphics.GraphicExtension Function 2850
Graphics.GraphicFileMask Function 2852
Graphics.GraphicFilter Function 2854
Graphics.IdentToCharset Function 2857
Graphics.IdentToColor Function 2857
Graphics.PaletteChanged Function 2858
Graphics.StringToColor Function 2858
Graphics.TransparentStretchBlt Function 2859

Structs, Records, Enums **2860**
Graphics.TAlphaFormat Enumeration 2860
Graphics.TBitmapHandleType Enumeration 2861
Graphics.TBrushData Record 2861
Graphics.TBrushStyle Enumeration 2862
Graphics.TCanvasOrientation Enumeration 2863
Graphics.TCanvasStates Enumeration 2863
Graphics.TCursorOrIcon Record 2864
Graphics.TFillMode Enumeration 2864
Graphics.TFillStyle Enumeration 2864
Graphics.TFontData Record 2865
Graphics.TFontPitch Enumeration 2866

Graphics.TFontStyle Enumeration 2866
Graphics.TIconRec Record 2867
Graphics.TPenData Record 2868
Graphics.TPenMode Enumeration 2868
Graphics.TPenStyle Enumeration 2870
Graphics.TPixelFormat Enumeration 2871
Graphics.TProgressStage Enumeration 2872
Graphics.TResData Record 2872
Graphics.TResource Record 2873
Graphics.TTextFormats Enumeration 2873
Graphics.TTransparentMode Enumeration 2874

Types **2875**
Graphics.HENHMETAFILE Type 2875
Graphics.HMETAFILE Type 2876
Graphics.PColor Type 2876
Graphics.PCursorOrIcon Type 2876
Graphics.PIconRec Type 2877
Graphics.PResource Type 2877
Graphics.TCanvasState Type 2877
Graphics.TColor Type 2878
Graphics.TCopyMode Type 2880
Graphics.TFontCharset Type 2881
Graphics.TFontDataName Type 2882
Graphics.TFontName Type 2882
Graphics.TFontStyles Type 2883
Graphics.TFontStylesBase Type 2887
Graphics.TGraphicClass Type 2887
Graphics.TProgressEvent Type 2888
Graphics.TTextFormat Type 2889

Variables **2890**
Graphics.DDBsOnly Variable 2890
Graphics.DefFontData Variable 2891
Graphics.SystemPalette16 Variable 2891

Constants **2892**
Graphics.cl3DDkShadow Constant 2896
Graphics.cl3DLight Constant 2896
Graphics.clActiveBorder Constant 2897
Graphics.clActiveCaption Constant 2897

Graphics.clAppWorkSpace Constant 2898
Graphics.clAqua Constant 2898
Graphics.clBackground Constant 2899
Graphics.clBlack Constant 2899
Graphics.clBlue Constant 2899
Graphics.clBtnFace Constant 2900
Graphics.clBtnHighlight Constant 2900
Graphics.clBtnShadow Constant 2901
Graphics.clBtnText Constant 2901
Graphics.clCaptionText Constant 2902
Graphics.clCream Constant 2902
Graphics.clDefault Constant 2903
Graphics.clDkGray Constant 2903
Graphics.clFuchsia Constant 2903
Graphics.clGradientActiveCaption Constant 2904
Graphics.clGradientInactiveCaption Constant 2904
Graphics.clGray Constant 2905
Graphics.clGrayText Constant 2905
Graphics.clGreen Constant 2906
Graphics.clHighlight Constant 2906
Graphics.clHighlightText Constant 2907
Graphics.clHotLight Constant 2907
Graphics.clInactiveBorder Constant 2908
Graphics.clInactiveCaption Constant 2908
Graphics.clInactiveCaptionText Constant 2908
Graphics.clInfoBk Constant 2909
Graphics.clInfoText Constant 2909
Graphics.clLime Constant 2910
Graphics.clLtGray Constant 2910
Graphics.clMaroon Constant 2911
Graphics.clMedGray Constant 2911
Graphics.clMenu Constant 2912
Graphics.clMenuBar Constant 2912
Graphics.clMenuHighlight Constant 2912
Graphics.clMenuText Constant 2913
Graphics.clMoneyGreen Constant 2913
Graphics.clNavy Constant 2914
Graphics.clNone Constant 2914
Graphics.clOlive Constant 2915

Graphics.clPurple Constant 2915
Graphics.clRed Constant 2916
Graphics.clScrollBar Constant 2916
Graphics.clSilver Constant 2916
Graphics.clSkyBlue Constant 2917
Graphics.clSystemColor Constant 2917
Graphics.clTeal Constant 2918
Graphics.clWebAliceBlue Constant 2918
Graphics.clWebAntiqueWhite Constant 2919
Graphics.clWebAqua Constant 2919
Graphics.clWebAquamarine Constant 2919
Graphics.clWebAzure Constant 2920
Graphics.clWebBeige Constant 2920
Graphics.clWebBisque Constant 2920
Graphics.clWebBlack Constant 2921
Graphics.clWebBlanchedAlmond Constant 2921
Graphics.clWebBlue Constant 2921
Graphics.clWebBlueViolet Constant 2922
Graphics.clWebBrown Constant 2922
Graphics.clWebBurlywood Constant 2922
Graphics.clWebCadetBlue Constant 2923
Graphics.clWebChartreuse Constant 2923
Graphics.clWebChocolate Constant 2923
Graphics.clWebCoral Constant 2924
Graphics.clWebCornFlowerBlue Constant 2924
Graphics.clWebCornSilk Constant 2924
Graphics.clWebCrimson Constant 2925
Graphics.clWebCyan Constant 2925
Graphics.clWebDarkBlue Constant 2925
Graphics.clWebDarkCyan Constant 2926
Graphics.clWebDarkGoldenRod Constant 2926
Graphics.clWebDarkGray Constant 2926
Graphics.clWebDarkgreen Constant 2927
Graphics.clWebDarkKhaki Constant 2927
Graphics.clWebDarkMagenta Constant 2927
Graphics.clWebDarkOliveGreen Constant 2928
Graphics.clWebDarkOrange Constant 2928
Graphics.clWebDarkOrchid Constant 2928
Graphics.clWebDarkRed Constant 2929

Graphics.clWebDarkSalmon Constant 2929
Graphics.clWebDarkSeaGreen Constant 2929
Graphics.clWebDarkSlateBlue Constant 2930
Graphics.clWebDarkSlategray Constant 2930
Graphics.clWebDarkTurquoise Constant 2930
Graphics.clWebDarkViolet Constant 2931
Graphics.clWebDeepPink Constant 2931
Graphics.clWebDeepskyBlue Constant 2931
Graphics.clWebDimGray Constant 2932
Graphics.clWebDodgerBlue Constant 2932
Graphics.clWebFirebrick Constant 2932
Graphics.clWebFloralWhite Constant 2933
Graphics.clWebForestGreen Constant 2933
Graphics.clWebFuchsia Constant 2933
Graphics.clWebGainsboro Constant 2934
Graphics.clWebGhostWhite Constant 2934
Graphics.clWebGold Constant 2934
Graphics.clWebGoldenRod Constant 2935
Graphics.clWebGray Constant 2935
Graphics.clWebGreen Constant 2935
Graphics.clWebGreenYellow Constant 2936
Graphics.clWebHoneydew Constant 2936
Graphics.clWebHotPink Constant 2936
Graphics.clWebIndianRed Constant 2937
Graphics.clWebIndigo Constant 2937
Graphics.clWebIvory Constant 2937
Graphics.clWebKhaki Constant 2938
Graphics.clWebLavender Constant 2938
Graphics.clWebLavenderBlush Constant 2938
Graphics.clWebLawnGreen Constant 2939
Graphics.clWebLemonChiffon Constant 2939
Graphics.clWebLightBlue Constant 2939
Graphics.clWebLightCoral Constant 2940
Graphics.clWebLightCyan Constant 2940
Graphics.clWebLightGoldenrodYellow Constant 2940
Graphics.clWebLightGreen Constant 2941
Graphics.clWebLightgrey Constant 2941
Graphics.clWebLightPink Constant 2941
Graphics.clWebLightSalmon Constant 2942

Graphics.clWebLightSeaGreen Constant 2942
Graphics.clWebLightSkyBlue Constant 2942
Graphics.clWebLightSlateGray Constant 2943
Graphics.clWebLightSteelBlue Constant 2943
Graphics.clWebLightYellow Constant 2943
Graphics.clWebLime Constant 2944
Graphics.clWebLimeGreen Constant 2944
Graphics.clWebLinen Constant 2944
Graphics.clWebMagenta Constant 2945
Graphics.clWebMaroon Constant 2945
Graphics.clWebMediumAquamarine Constant 2945
Graphics.clWebMediumBlue Constant 2946
Graphics.clWebMediumOrchid Constant 2946
Graphics.clWebMediumPurple Constant 2946
Graphics.clWebMediumSeaGreen Constant 2947
Graphics.clWebMediumSlateBlue Constant 2947
Graphics.clWebMediumSpringGreen Constant 2947
Graphics.clWebMediumTurquoise Constant 2948
Graphics.clWebMediumVioletRed Constant 2948
Graphics.clWebMidnightBlue Constant 2948
Graphics.clWebMintcream Constant 2949
Graphics.clWebMistyRose Constant 2949
Graphics.clWebMoccasin Constant 2949
Graphics.clWebNavajoWhite Constant 2950
Graphics.clWebNavy Constant 2950
Graphics.clWebOldLace Constant 2950
Graphics.clWebOlive Constant 2951
Graphics.clWebOliveDrab Constant 2951
Graphics.clWebOrange Constant 2951
Graphics.clWebOrangeRed Constant 2952
Graphics.clWebOrchid Constant 2952
Graphics.clWebPaleGoldenrod Constant 2952
Graphics.clWebPaleGreen Constant 2953
Graphics.clWebPaleTurquoise Constant 2953
Graphics.clWebPaleVioletRed Constant 2953
Graphics.clWebPapayaWhip Constant 2954
Graphics.clWebPeachPuff Constant 2954
Graphics.clWebPeru Constant 2954
Graphics.clWebPink Constant 2955

Graphics.clWebPlum Constant 2955
Graphics.clWebPowderBlue Constant 2955
Graphics.clWebPurple Constant 2956
Graphics.clWebRed Constant 2956
Graphics.clWebRosyBrown Constant 2956
Graphics.clWebRoyalBlue Constant 2957
Graphics.clWebSaddleBrown Constant 2957
Graphics.clWebSalmon Constant 2957
Graphics.clWebSandyBrown Constant 2958
Graphics.clWebSeaGreen Constant 2958
Graphics.clWebSeashell Constant 2958
Graphics.clWebSienna Constant 2959
Graphics.clWebSilver Constant 2959
Graphics.clWebSkyBlue Constant 2959
Graphics.clWebSlateBlue Constant 2960
Graphics.clWebSlateGray Constant 2960
Graphics.clWebSnow Constant 2960
Graphics.clWebSpringGreen Constant 2961
Graphics.clWebSteelBlue Constant 2961
Graphics.clWebTan Constant 2961
Graphics.clWebTeal Constant 2962
Graphics.clWebThistle Constant 2962
Graphics.clWebTomato Constant 2962
Graphics.clWebTurquoise Constant 2963
Graphics.clWebViolet Constant 2963
Graphics.clWebWheat Constant 2963
Graphics.clWebWhite Constant 2964
Graphics.clWebWhiteSmoke Constant 2964
Graphics.clWebYellow Constant 2964
Graphics.clWebYellowGreen Constant 2965
Graphics.clWhite Constant 2965
Graphics.clWindow Constant 2965
Graphics.clWindowFrame Constant 2966
Graphics.clWindowText Constant 2966
Graphics.clYellow Constant 2967
Graphics.cmBlackness Constant 2967
Graphics.cmDstInvert Constant 2968
Graphics.cmMergeCopy Constant 2968
Graphics.cmMergePaint Constant 2969

Graphics.cmNotSrcCopy Constant 2969
Graphics.cmNotSrcErase Constant 2970
Graphics.cmPatCopy Constant 2970
Graphics.cmPatInvert Constant 2971
Graphics.cmPatPaint Constant 2971
Graphics.cmSrcAnd Constant 2972
Graphics.cmSrcCopy Constant 2972
Graphics.cmSrcErase Constant 2973
Graphics.cmSrcInvert Constant 2973
Graphics.cmSrcPaint Constant 2974
Graphics.cmWhiteness Constant 2974
Graphics.ExtendedColorsCount Constant 2975
Graphics.rc3_Cursor Constant 2976
Graphics.rc3_Icon Constant 2976
Graphics.rc3_StockIcon Constant 2976
Graphics.StandardColorsCount Constant 2977
Graphics.WebColorsCount Constant 2978

GraphUtil Namespace 2979

Functions 2981

GraphUtil.ColorAdjustLuma Function 2981
GraphUtil.ColorHLSToRGB Function 2982
GraphUtil.ColorRGBToHLS Function 2982
GraphUtil.ColorToWebColorName Function 2982
GraphUtil.ColorToWebColorStr Function 2983
GraphUtil.DrawArrow Function 2983
GraphUtil.DrawCheck Function 2983
GraphUtil.DrawChevron Function 2984
GraphUtil.DrawTransparentBitmap Function 2984
GraphUtil.DrawTransparentBitmap Function 2984
GraphUtil.GetHighLightColor Function 2985
GraphUtil.GetShadowColor Function 2985
GraphUtil.GradientFillCanvas Function 2985
GraphUtil.LoadCompressedResourceBitmap Function 2986
GraphUtil.RGBToWebColorName Function 2986
GraphUtil.RGBToWebColorStr Function 2986
GraphUtil.ScaleImage Function 2987
GraphUtil.SortColorArray Function 2987

GraphUtil.SplitTransparentBitmap Function 2987
GraphUtil.WebColorNameToColor Function 2988
GraphUtil.WebColorStrToColor Function 2988
GraphUtil.WebColorToRGB Function 2988

Structs, Records, Enums **2989**
GraphUtil.TArrowType Enumeration 2989
GraphUtil.TColorArraySortType Enumeration 2989
GraphUtil.TGradientDirection Enumeration 2990
GraphUtil.TScrollDirection Enumeration 2990

Types **2992**
GraphUtil.TColorArray Type 2992

Constants **2993**
GraphUtil.WebNamedColors Constant 2993
GraphUtil.WebNamedColorsCount Constant 2995

Grids Namespace 2997

Classes **2999**
EInvalidGridOperation Class 2999
TCustomDrawGrid Class 3000
TCustomDrawGrid Members 3000
TCustomDrawGrid Methods 3003
TCustomDrawGrid Properties 3007
TCustomDrawGrid Events 3019
TCustomGrid Class 3027
TCustomGrid Members 3028
TCustomGrid.Create Constructor 3030
TCustomGrid.Destroy Destructor 3031
TCustomGrid Fields 3031
TCustomGrid Methods 3032
TCustomGrid Properties 3059
TDrawGrid Class 3083
TDrawGrid Members 3083
TDrawGrid Properties 3087
TDrawGrid Events 3124
TInplaceEdit Class 3161
TInplaceEdit Members 3161
TInplaceEdit.Create Constructor 3162

TInplaceEdit Methods 3163
TInplaceEdit Properties 3171
TInplaceEditList Class 3171
TInplaceEditList Members 3171
TInplaceEditList.Create Constructor 3172
TInplaceEditList Methods 3173
TInplaceEditList Properties 3177
TInplaceEditList Events 3179
TStringGrid Class 3180
TStringGrid Members 3181
TStringGrid.Create Constructor 3185
TStringGrid.Destroy Destructor 3186
TStringGrid Methods 3186
TStringGrid Properties 3188
TStringGridStrings Class 3192
TStringGridStrings Members 3192
TStringGridStrings.Create Constructor 3193
TStringGridStrings Methods 3193

Structs, Records, Enums 3199
Grids.TEditStyle Enumeration 3199
Grids.TGridAxisDrawInfo Record 3200
Grids.TGridCoord Record 3200
Grids.TGridDrawInfo Record 3201
Grids.TGridOption Enumeration 3201
Grids.TGridRect Record 3203
Grids.TGridState Enumeration 3203

Types 3205
Grids.TDrawCellEvent Type 3205
Grids.TGetEditEvent Type 3206
Grids.TGetExtentsFunc Type 3206
Grids.TGridDrawState Type 3206
Grids.TGridMovement Type 3207
Grids.TGridOptions Type 3207
Grids.TGridScrollDirection Type 3209
Grids.TMovedEvent Type 3210
Grids.TOnGetPickListItems Type 3210
Grids.TSelectCellEvent Type 3210
Grids.TSetEditEvent Type 3211

Constants **3212**
Grids.MaxCustomExtents Constant 3212
Grids.MaxShortInt Constant 3212

HelpIntfs Namespace 3213

Classes **3214**
EHelpSystemException Class 3214

Interfaces **3215**
ICustomHelpViewer Interface 3215
ICustomHelpViewer Members 3216
ICustomHelpViewer Methods 3216
IExtendedHelpViewer Interface 3219
IExtendedHelpViewer Members 3220
IExtendedHelpViewer Methods 3220
IExtendedHelpViewer2 Interface 3222
IExtendedHelpViewer2 Members 3222
IExtendedHelpViewer2 Methods 3223
IFilteredHelpViewer Interface 3224
IFilteredHelpViewer Members 3225
IFilteredHelpViewer Methods 3225
IHelpManager Interface 3226
IHelpManager Members 3226
IHelpManager Methods 3227
IHelpSelector Interface 3228
IHelpSelector Members 3228
IHelpSelector Methods 3229
IHelpSelector2 Interface 3230
IHelpSelector2 Members 3230
IHelpSelector2 Methods 3230
IHelpSystem Interface 3231
IHelpSystem Members 3231
IHelpSystem Methods 3231
IHelpSystem2 Interface 3234
IHelpSystem2 Members 3234
IHelpSystem2 Methods 3234
IHelpSystem3 Interface 3235
IHelpSystem3 Members 3235
IHelpSystem3 Methods 3236

IHelpSystemFlags Interface 3238
IHelpSystemFlags Members 3238
IHelpSystemFlags Methods 3239
ISpecialWinHelpViewer Interface 3239
ISpecialWinHelpViewer Members 3240
ISpecialWinHelpViewer Methods 3240

Functions **3242**
HelpIntfs.GetHelpSystem Function 3242
HelpIntfs.RegisterViewer Function 3242

HTMLHelpViewer Namespace 3245

Interfaces **3246**
IHtmlHelpTester Interface 3246
IHtmlHelpTester Members 3246
IHtmlHelpTester Methods 3246

Variables **3249**
HTMLHelpViewer.HtmlHelpTester Variable 3249
HTMLHelpViewer.ViewerName Variable 3249

HTTPApp Namespace 3251

Classes **3254**
EWebBrokerException Class 3254
TAbstractContentParser Class 3255
TAbstractContentParser Members 3256
TAbstractContentParser.Create Constructor 3256
TAbstractContentParser Methods 3256
TAbstractContentParser Properties 3258
TAbstractWebModuleFactory Class 3259
TAbstractWebModuleFactory Members 3259
TAbstractWebModuleFactory Methods 3260
TAbstractWebModuleFactory Properties 3263
TAbstractWebModuleList Class 3266
TAbstractWebModuleList Members 3267
TAbstractWebModuleList Methods 3268
TAbstractWebModuleList Properties 3272
TAbstractWebModuleList Events 3274
TAbstractWebPageInfo Class 3275

TAbstractWebPageInfo Members 3276
TAbstractWebPageInfo Methods 3276
TAbstractWebPageInfo Properties 3280
TAbstractWebPageModuleFactory Class 3285
TAbstractWebPageModuleFactory Members 3285
TAbstractWebPageModuleFactory.Create Constructor 3286
TAbstractWebPageModuleFactory.Destroy Destructor 3286
TAbstractWebPageModuleFactory Methods 3287
TAbstractWebRequestFile Class 3288
TAbstractWebRequestFile Members 3288
TAbstractWebRequestFile Methods 3289
TAbstractWebRequestFile Properties 3291
TAbstractWebRequestFiles Class 3292
TAbstractWebRequestFiles Members 3293
TAbstractWebRequestFiles Methods 3293
TAbstractWebRequestFiles Properties 3294
TAbstractWebSession Class 3295
TAbstractWebSession Members 3296
TAbstractWebSession Methods 3297
TAbstractWebSession Properties 3300
TContentParser Class 3301
TContentParser Members 3302
TContentParser.Destroy Destructor 3302
TContentParser Methods 3303
TCookie Class 3304
TCookie Members 3305
TCookie.Create Constructor 3305
TCookie Methods 3306
TCookie Properties 3306
TCookieCollection Class 3309
TCookieCollection Members 3310
TCookieCollection.Create Constructor 3310
TCookieCollection Methods 3311
TCookieCollection Properties 3312
TCustomContentProducer Class 3313
TCustomContentProducer Members 3314
TCustomContentProducer Methods 3314
TCustomContentProducer Properties 3317
TCustomWebDispatcher Class 3318

TCustomWebDispatcher Members 3319
TCustomWebDispatcher.Create Constructor 3320
TCustomWebDispatcher.Destroy Destructor 3321
TCustomWebDispatcher Methods 3321
TCustomWebDispatcher Properties 3326
TCustomWebDispatcher Events 3328
TDefaultWebAppServices Class 3330
TDefaultWebAppServices Members 3331
TDefaultWebAppServices Methods 3331
TDefaultWebAppServices Properties 3334
TWebActionItem Class 3335
TWebActionItem Members 3336
TWebActionItem.Create Constructor 3336
TWebActionItem.Destroy Destructor 3337
TWebActionItem Methods 3337
TWebActionItem Properties 3339
TWebActionItem Events 3343
TWebActionItems Class 3344
TWebActionItems Members 3344
TWebActionItems.Create Constructor 3345
TWebActionItems Methods 3345
TWebActionItems Properties 3349
TWebDispatcher Class 3349
TWebDispatcher Members 3350
TWebDispatcher Properties 3351
TWebDispatcher Events 3352
TWebModule Class 3354
TWebModule Members 3355
TWebModule.Create Constructor 3356
TWebModule Properties 3356
TWebModule Events 3357
TWebRequest Class 3358
TWebRequest Members 3359
TWebRequest.Create Constructor 3361
TWebRequest.Destroy Destructor 3361
TWebRequest Methods 3362
TWebRequest Properties 3370
TWebResponse Class 3389
TWebResponse Members 3390

TWebResponse.Create Constructor 3391
TWebResponse.Destroy Destructor 3392
TWebResponse Fields 3392
TWebResponse Methods 3393
TWebResponse Properties 3403

Interfaces 3416
IGetAppDispatcher Interface 3416
IGetAppDispatcher Members 3416
IGetAppDispatcher Methods 3417
IGetWebAppServices Interface 3417
IGetWebAppServices Members 3417
IGetWebAppServices Methods 3418
IGetWebRequestHandler Interface 3418
IGetWebRequestHandler Members 3418
IGetWebRequestHandler Methods 3418
IMultiModuleSupport Interface 3419
IMultiModuleSupport Members 3419
IMultiModuleSupport Methods 3419
IProduceContent Interface 3420
IProduceContent Members 3421
IProduceContent Methods 3421
IProduceContentFrom Interface 3422
IProduceContentFrom Members 3422
IProduceContentFrom Methods 3422
ISetAppDispatcher Interface 3423
ISetAppDispatcher Members 3423
ISetAppDispatcher Methods 3423
IWebAppServices Interface 3424
IWebAppServices Members 3425
IWebAppServices Methods 3425
IWebAppServices Properties 3427
IWebDispatch Interface 3427
IWebDispatch Members 3428
IWebDispatch Methods 3428
IWebDispatch Properties 3429
IWebDispatchActions Interface 3430
IWebDispatcherAccess Interface 3431
IWebDispatcherAccess Members 3432
IWebDispatcherAccess Methods 3432

IWebExceptionHandler Interface 3433
IWebExceptionHandler Members 3433
IWebExceptionHandler Methods 3433
IWebRequestHandler Interface 3434
IWebRequestHandler Members 3434
IWebRequestHandler Methods 3434

Functions **3436**
HTTPApp.DayOfWeekStr Function 3436
HTTPApp.DosPathToUnixPath Function 3437
HTTPApp.EncodingFromContentType Function 3437
HTTPApp.EncodingGetBytes Function 3437
HTTPApp.EncodingGetString Function 3438
HTTPApp.ExtractHeaderFields Function 3438
HTTPApp.ExtractHTTPFields Function 3439
HTTPApp.HTMLDecode Function 3439
HTTPApp.HTMLEncode Function 3440
HTTPApp.HTTPDecode Function 3440
HTTPApp.HTTPEncode Function 3441
HTTPApp.MonthStr Function 3441
HTTPApp.ParseDate Function 3442
HTTPApp.RegisterContentParser Function 3442
HTTPApp.StatusString Function 3442
HTTPApp.UnixPathToDosPath Function 3443

Structs, Records, Enums **3444**
HTTPApp.TMethodType Enumeration 3444
HTTPApp.TWebModuleCacheMode Enumeration 3444
HTTPApp.TWebModuleCreateMode Enumeration 3445

Types **3446**
HTTPApp.TContentParserClass Type 3446
HTTPApp.THTTPMethodEvent Type 3446
HTTPApp.TModuleAddedProc Type 3447
HTTPApp.TWebExceptionEvent Type 3447
HTTPApp.TWebModuleContext Type 3447

Constants **3448**
HTTPApp.MAX_DATETIMES Constant 3448
HTTPApp.MAX_INTEGERS Constant 3448
HTTPApp.MAX_STRINGS Constant 3449

HTTPApp.sDateFormat Constant 3449

HTTPProd Namespace 3451

Classes 3453

EScriptError Class 3453
EScriptError Members 3453
EScriptError.Create Constructor 3454
EScriptError.Destroy Destructor 3454
EScriptError Properties 3455
TAbstractScriptEnginesList Class 3455
TAbstractScriptEnginesList Members 3456
TAbstractScriptEnginesList.Create Constructor 3456
TAbstractScriptEnginesList Methods 3456
TAbstractScriptError Class 3457
TAbstractScriptError Members 3457
TAbstractScriptError Methods 3457
TAbstractScriptError Properties 3459
TAbstractScriptErrors Class 3460
TAbstractScriptErrors Members 3461
TAbstractScriptErrors Methods 3461
TAbstractScriptErrors Properties 3462
TAbstractScriptProducer Class 3463
TAbstractScriptProducer Members 3463
TAbstractScriptProducer.Create Constructor 3463
TBasePageProducer Class 3464
TBasePageProducer Members 3465
TBasePageProducer.Create Constructor 3466
TBasePageProducer Methods 3466
TBasePageProducer Properties 3471
TBasePageProducer Events 3472
TCustomPageProducer Class 3473
TCustomPageProducer Members 3473
TCustomPageProducer.Create Constructor 3474
TCustomPageProducer.Destroy Destructor 3475
TCustomPageProducer Methods 3475
TCustomPageProducer Properties 3476
THTMLTableAttributes Class 3478
THTMLTableAttributes Members 3478

THTMLTableAttributes.Create Constructor 3479
THTMLTableAttributes Methods 3480
THTMLTableAttributes Properties 3481
THTMLTableCellAttributes Class 3484
THTMLTableElementAttributes Class 3484
THTMLTableElementAttributes Members 3485
THTMLTableElementAttributes Methods 3486
THTMLTableElementAttributes Properties 3487
THTMLTableHeaderAttributes Class 3488
THTMLTableHeaderAttributes Members 3489
THTMLTableHeaderAttributes Methods 3489
THTMLTableHeaderAttributes Properties 3491
THTMLTableRowAttributes Class 3491
THTMLTagAttributes Class 3492
THTMLTagAttributes Members 3492
THTMLTagAttributes.Create Constructor 3493
THTMLTagAttributes Methods 3493
THTMLTagAttributes Properties 3494
THTMLTagAttributes Events 3495
TPageProducer Class 3496
TPageProducer Members 3497
TPageProducer Properties 3498
TPageProducer Events 3500

Interfaces 3502
IDesignerFileManager Interface 3502
IDesignerFileManager Members 3502
IDesignerFileManager Methods 3502
IGetLocateFileService Interface 3503
IGetLocateFileService Members 3503
IGetLocateFileService Methods 3504
IGetProducerTemplate Interface 3504
IGetProducerTemplate Members 3504
IGetProducerTemplate Methods 3505
ILocateFileService Interface 3505
ILocateFileService Members 3506
ILocateFileService Methods 3506
IScriptContext Interface 3506
IScriptContext Members 3507
IScriptContext Methods 3507

IScriptContext Properties 3507
IScriptProducer Interface 3508
IScriptProducer Members 3508
IScriptProducer Methods 3508
IScriptProducer Properties 3512
IWebVariableName Interface 3513
IWebVariableName Members 3513
IWebVariableName Methods 3514
IWebVariableName Properties 3514
IWebVariablesContainer Interface 3514
IWebVariablesContainer Members 3515
IWebVariablesContainer Methods 3515
IWebVariablesContainer Properties 3516

Functions **3518**
HTTPProd.ContentFromScriptFile Function 3518
HTTPProd.ContentFromScriptStream Function 3518
HTTPProd.ContentFromStream Function 3519
HTTPProd.ContentFromString Function 3519
HTTPProd.FindComponentWebModuleContext Function 3519
HTTPProd.GetEncodingOfStream Function 3520
HTTPProd.GetTagID Function 3520

Structs, Records, Enums **3521**
HTTPProd.THTMLAlign Enumeration 3521
HTTPProd.THTMLVAlign Enumeration 3521
HTTPProd.TTag Enumeration 3522

Types **3523**
HTTPProd.THandledTagProc Type 3523
HTTPProd.THandleTagProc Type 3523
HTTPProd.THTMLBgColor Type 3524
HTTPProd.THTMLTagEvent Type 3524
HTTPProd.TScriptProducerClass Type 3524

Variables **3525**
HTTPProd.DesignerFileManager Variable 3525
HTTPProd.ScriptEnginesList Variable 3525

Constants **3526**
HTTPProd.HTMLAlign Constant 3526
HTTPProd.HTMLVAlign Constant 3526

HTTPUtil Namespace 3527

Interfaces **3528**

IStreamLoader Interface 3528

IStreamLoader Members 3528

IStreamLoader Methods 3529

IStreamLoader Properties 3532

IStringTokenizer Interface 3534

IStringTokenizer Members 3534

IStringTokenizer Methods 3535

IStringTokenizer Properties 3536

Functions **3537**

HTTPUtil.FirstDelimiter Function 3537

HTTPUtil.FirstDelimiter Function 3537

HTTPUtil.GetDefaultStreamLoader Function 3538

HTTPUtil.HTMLEscape Function 3538

HTTPUtil.StartsWith Function 3538

HTTPUtil.StringTokenizer Function 3539

HTTPUtil.StringToStringArray Function 3539

IB Namespace 3541

Classes **3543**

EIBClientError Class 3543

EIBError Class 3543

EIBError Members 3544

Create Constructor 3544

EIBError Properties 3545

EIBInterBaseError Class 3545

EIBInterBaseRoleError Class 3546

Functions **3547**

IB.CheckStatusVector Function 3547

IB.GetIBDataBaseErrorMessages Function 3547

IB.IBAlloc Function 3548

IB.IBDataBaseError Function 3548

IB.IBError Function 3548

IB.SetIBDataBaseErrorMessages Function 3549

IB.StatusVector Function 3549

IB.StatusVectorArray Function 3549
IB.StatusVectorAsText Function 3550

Structs, Records, Enums **3551**
IB.TIBClientError Enumeration 3551
IB.TIBDataBaseErrorMessage Enumeration 3555
IB.TTraceFlag Enumeration 3555

Types **3557**
IB.PStatusVector Type 3557
IB.TIBDataBaseErrorMessages Type 3557
IB.TStatusVector Type 3558
IB.TTraceFlags Type 3558

Variables **3560**
IB.IBCS Variable 3560

Constants **3561**
IB.IBBigLocalBufferLength Constant 3561
IB.IBErrorMessages Constant 3561
IB.IBHugeLocalBufferLength Constant 3562
IB.IBLocalBufferLength Constant 3563
IB.IBPalette1 Constant 3563
IB.IBPalette2 Constant 3563

IBBatchMove Namespace 3565

Classes **3566**
TIBBatchMove Class 3566
TIBBatchMove Members 3566
TIBBatchMove.Create Constructor 3567
TIBBatchMove.Destroy Destructor 3568
TIBBatchMove Methods 3568
TIBBatchMove Properties 3569

Structs, Records, Enums **3578**
IBBatchMove.TBatchMode Enumeration 3578

IBBlob Namespace 3579

Classes **3580**
TIBBlobStream Class 3580
TIBBlobStream Members 3580

TIBBlobStream.Create Constructor 3581
TIBBlobStream.Destroy Destructor 3582
TIBBlobStream Methods 3582
TIBBlobStream Properties 3590

Functions 3594
IBBlob.GetBlobInfo Function 3594
IBBlob.ReadBlob Function 3594
IBBlob.WriteBlob Function 3595

Constants 3596
IBBlob.DefaultBlobSegmentSize Constant 3596

IBCustomDataSet Namespace 3597

Classes 3599
TIBBCDField Class 3599
TIBBCDField Members 3599
TIBBCDField.Create Constructor 3600
TIBBCDField Methods 3600
TIBBCDField Properties 3601
TIBCustomDataSet Class 3602
TIBCustomDataSet Members 3602
TIBCustomDataSet.Create Constructor 3606
TIBCustomDataSet.Destroy Destructor 3606
TIBCustomDataSet Methods 3607
TIBCustomDataSet Properties 3632
TIBCustomDataSet Events 3648
TIBDataLink Class 3651
TIBDataLink Members 3652
TIBDataLink.Create Constructor 3652
TIBDataLink.Destroy Destructor 3652
TIBDataLink Methods 3653
TIBDataSet Class 3654
TIBDataSet Members 3654
TIBDataSet Methods 3658
TIBDataSet Properties 3661
TIBDataSet Events 3673
TIBDataSetUpdateObject Class 3674
TIBDataSetUpdateObject Members 3675
TIBDataSetUpdateObject.Create Constructor 3675

TIBDataSetUpdateObject.Destroy Destructor 3676
TIBDataSetUpdateObject Methods 3676
TIBDataSetUpdateObject Properties 3677
TIBDSBlobStream Class 3678
TIBDSBlobStream Members 3678
TIBDSBlobStream.Create Constructor 3679
TIBDSBlobStream.Destroy Destructor 3679
TIBDSBlobStream Fields 3679
TIBDSBlobStream Methods 3680
TIBGeneratorField Class 3682
TIBGeneratorField Members 3682
TIBGeneratorField.Create Constructor 3683
TIBGeneratorField Methods 3683
TIBGeneratorField Properties 3684
TIBStringField Class 3686
TIBStringField Members 3686
TIBStringField.Create Constructor 3687
TIBStringField Methods 3687

Structs, Records, Enums **3689**
IBCustomDataSet.TCachedUpdateStatus Enumeration 3689
IBCustomDataSet.TFieldData Record 3690
IBCustomDataSet.TIBDBKey Record 3690
IBCustomDataSet.TIBGeneratorApplyEvent Enumeration 3691
IBCustomDataSet.TIBUpdateAction Enumeration 3691
IBCustomDataSet.TLiveMode Enumeration 3692
IBCustomDataSet.TRecordData Record 3693

Types **3694**
IBCustomDataSet.PBlobDataArray Type 3694
IBCustomDataSet.PDateTime Type 3694
IBCustomDataSet.PFieldData Type 3695
IBCustomDataSet.PIBDBKey Type 3695
IBCustomDataSet.PRecordData Type 3695
IBCustomDataSet.TBlobDataArray Type 3696
IBCustomDataSet.TIBUpdateErrorEvent Type 3696
IBCustomDataSet.TIBUpdateRecordEvent Type 3696
IBCustomDataSet.TIBUpdateRecordTypes Type 3697
IBCustomDataSet.TLiveModes Type 3697

Variables **3699**

IBCustomDataSet.CreateProviderProc Variable 3699

Constants **3700**

IBCustomDataSet.BufferCacheSize Constant 3700

IBCustomDataSet.DefaultFieldClasses Constant 3700

IBCustomDataSet.UniCache Constant 3701

IBDatabase Namespace 3703

Classes **3705**

TIBBase Class 3705

TIBBase Members 3705

TIBBase.Create Constructor 3706

TIBBase.Destroy Destructor 3707

TIBBase Fields 3707

TIBBase Methods 3710

TIBBase Properties 3713

TIBBase Events 3714

TIBDataBase Class 3716

TIBDataBase Members 3716

TIBDataBase.Create Constructor 3718

TIBDataBase.Destroy Destructor 3718

TIBDataBase Methods 3719

TIBDataBase Properties 3728

TIBDataBase Events 3738

TIBSchema Class 3741

TIBSchema Members 3741

TIBSchema Methods 3742

TIBTimer Class 3742

TIBTimer Members 3743

TIBTimer.Create Constructor 3743

TIBTimer.Destroy Destructor 3744

TIBTimer Methods 3745

TIBTimer Properties 3745

TIBTimer Events 3746

TIBTransaction Class 3746

TIBTransaction Members 3747

TIBTransaction.Create Constructor 3748

TIBTransaction.Destroy Destructor 3748

TIBTransaction Methods 3748

TIBTransaction Properties 3755
TIBTransaction Events 3759

Interfaces 3761
IIBEventNotifier Interface 3761
IIBEventNotifier Members 3761
IIBEventNotifier Methods 3762

Functions 3764
IBDatabase.GenerateDPB Function 3764
IBDatabase.GenerateTPB Function 3764

Structs, Records, Enums 3765
IBDatabase.TAutoStopAction Enumeration 3765
IBDatabase.TTransactionAction Enumeration 3766

Types 3767
IBDatabase.TIBDatabaseLoginEvent Type 3767
IBDatabase.TIBFileName Type 3767

Constants 3768
IBDatabase.DPBConstantNames Constant 3768
IBDatabase.DPBPrefix Constant 3769
IBDatabase.TPBConstantNames Constant 3769
IBDatabase.TPBPrefix Constant 3769

IBDatabaseInfo Namespace 3771

Classes 3772
TIBDatabaseInfo Class 3772
TIBDatabaseInfo Members 3772
TIBDatabaseInfo.Create Constructor 3774
TIBDatabaseInfo.Destroy Destructor 3774
TIBDatabaseInfo Fields 3775
TIBDatabaseInfo Methods 3777
TIBDatabaseInfo Properties 3786

IBEvents Namespace 3797

Classes 3798
TIBEvents Class 3798
TIBEvents Members 3798
TIBEvents.Create Constructor 3799

TIBEvents.Destroy Destructor 3800
TIBEvents Methods 3800
TIBEvents Properties 3803
TIBEvents Events 3805

Types 3806
IBEvents.TErrorEvent Type 3806
IBEvents.TEventAlert Type 3806

IBExtract Namespace 3809

Classes 3811
TIBExtract Class 3811
TIBExtract Members 3811
TIBExtract.Create Constructor 3812
TIBExtract.Destroy Destructor 3813
TIBExtract Methods 3813
TIBExtract Properties 3821

Structs, Records, Enums 3823
IBExtract.TExtractObjectTypes Enumeration 3823
IBExtract.TExtractType Enumeration 3824
IBExtract.TPrivTypes Record 3825
IBExtract.TSQLType Record 3825

Types 3827
IBExtract.TExtractTypes Type 3827
IBExtract.TSQLTypes Type 3828

Constants 3829
IBExtract.ColumnTypes Constant 3830
IBExtract.FILE_cache Constant 3830
IBExtract.FILE_conditional Constant 3830
IBExtract.FILE_inactive Constant 3831
IBExtract.FILE_manual Constant 3831
IBExtract.FILE_shadow Constant 3831
IBExtract.IntegralSubtypes Constant 3832
IBExtract.LOG_default Constant 3832
IBExtract.LOG_overflow Constant 3832
IBExtract.LOG_raw Constant 3833
IBExtract.LOG_serial Constant 3833
IBExtract.MAX_INTSUBTYPES Constant 3833

IBExtract.MAXSUBTYPES Constant 3834
IBExtract.obj_computed Constant 3834
IBExtract.obj_count Constant 3834
IBExtract.obj_exception Constant 3835
IBExtract.obj_expression_index Constant 3835
IBExtract.obj_field Constant 3835
IBExtract.obj_index Constant 3836
IBExtract.obj_procedure Constant 3836
IBExtract.obj_relation Constant 3836
IBExtract.obj_sql_role Constant 3837
IBExtract.obj_trigger Constant 3837
IBExtract.obj_user Constant 3837
IBExtract.obj_user_group Constant 3838
IBExtract.obj_validation Constant 3838
IBExtract.obj_view Constant 3838
IBExtract.ODS_VERSION10 Constant 3839
IBExtract.ODS_VERSION6 Constant 3839
IBExtract.ODS_VERSION7 Constant 3839
IBExtract.ODS_VERSION8 Constant 3840
IBExtract.ODS_VERSION9 Constant 3840
IBExtract.priv_DELETE Constant 3840
IBExtract.priv_EXECUTE Constant 3841
IBExtract.priv_INSERT Constant 3841
IBExtract.priv_REFERENCES Constant 3841
IBExtract.priv_SELECT Constant 3842
IBExtract.priv_UNKNOWN Constant 3842
IBExtract.priv_UPDATE Constant 3842
IBExtract.PrivTypes Constant 3843
IBExtract.SubTypes Constant 3843
IBExtract.TriggerTypes Constant 3843

IBInstall Namespace 3845

Classes **3847**
EIBInstall Class 3847
EIBInstall Members 3847
Create Constructor 3848
EIBInstall Properties 3848
EIBInstallerError Class 3849

EIBInstallError Class 3850
TIBInstall Class 3850
TIBInstall Members 3850
TIBInstall.Create Constructor 3851
TIBInstall.Destroy Destructor 3852
TIBInstall Methods 3852
TIBInstall Properties 3855
TIBSetup Class 3857
TIBSetup Members 3857
TIBSetup.Create Constructor 3858
TIBSetup Methods 3858
TIBSetup Properties 3859
TIBSetup Events 3861
TIBUnInstall Class 3862
TIBUnInstall Members 3863
TIBUnInstall Methods 3863
TIBUnInstall Properties 3864
TInstallOptions Class 3865
TInstallOptions Members 3865
TInstallOptions Properties 3865

Structs, Records, Enums 3868
IBInstall.TCmdOption Enumeration 3868
IBInstall.TConnectivityOption Enumeration 3869
IBInstall.TErrorResult Enumeration 3869
IBInstall.TExamplesOption Enumeration 3870
IBInstall.TIBInstallerError Enumeration 3870
IBInstall.TMainOption Enumeration 3871
IBInstall.TStatusResult Enumeration 3872
IBInstall.TWarningResult Enumeration 3872

Types 3873
IBInstall.TCmdOptions Type 3873
IBInstall.TConnectivityOptions Type 3873
IBInstall.TExamplesOptions Type 3874
IBInstall.TIBSetupOnError Type 3874
IBInstall.TIBSetupOnStatus Type 3874
IBInstall.TIBSetupOnWarning Type 3875
IBInstall.TIscError Type 3875
IBInstall.TMainOptions Type 3876

IBIntf Namespace 3877

Functions **3880**

IBIntf.CheckIBInstallLoaded Function 3880
IBIntf.CheckIBLoaded Function 3881
IBIntf.FreeIBInstallLibrary Function 3881
IBIntf.FreeIBLibrary Function 3881
IBIntf.GetIBClientVersion Function 3882
IBIntf.isc_decode_sql_date_stub Function 3882
IBIntf.isc_decode_sql_time_stub Function 3882
IBIntf.isc_decode_timestamp_stub Function 3883
IBIntf.isc_encode_sql_date_stub Function 3883
IBIntf.isc_encode_sql_time_stub Function 3883
IBIntf.isc_encode_timestamp_stub Function 3884
IBIntf.isc_install_clear_options_stub Function 3884
IBIntf.isc_install_execute_stub Function 3884
IBIntf.isc_install_get_info_stub Function 3885
IBIntf.isc_install_get_message_stub Function 3885
IBIntf.isc_install_load_external_text_stub Function 3885
IBIntf.isc_install_precheck_stub Function 3886
IBIntf.isc_install_set_option_stub Function 3886
IBIntf.isc_install_unset_option_stub Function 3886
IBIntf.isc_rollback_retaining_stub Function 3887
IBIntf.isc_service_attach_stub Function 3887
IBIntf.isc_service_detach_stub Function 3887
IBIntf.isc_service_query_stub Function 3888
IBIntf.isc_service_start_stub Function 3888
IBIntf.isc_uninstall_execute_stub Function 3888
IBIntf.isc_uninstall_precheck_stub Function 3889
IBIntf.LoadIBInstallLibrary Function 3889
IBIntf.LoadIBLibrary Function 3889
IBIntf.TryIBLoad Function 3890

Variables **3891**

IBIntf.BLOB_get Variable 3892
IBIntf.BLOB_put Variable 3892
IBIntf.isc_add_user Variable 3893
IBIntf.isc_attach_database Variable 3893
IBIntf.isc_blob_info Variable 3893

IBIntf.isc_cancel_events Variable 3894
IBIntf.isc_close_blob Variable 3894
IBIntf.isc_commit_retaining Variable 3894
IBIntf.isc_commit_transaction Variable 3895
IBIntf.isc_create_blob2 Variable 3895
IBIntf.isc_database_info Variable 3895
IBIntf.isc_decode_date Variable 3896
IBIntf.isc_decode_sql_date Variable 3896
IBIntf.isc_decode_sql_time Variable 3896
IBIntf.isc_decode_timestamp Variable 3897
IBIntf.isc_delete_user Variable 3897
IBIntf.isc_detach_database Variable 3897
IBIntf.isc_drop_database Variable 3898
IBIntf.isc_dsql_alloc_statement2 Variable 3898
IBIntf.isc_dsql_describe Variable 3898
IBIntf.isc_dsql_describe_bind Variable 3899
IBIntf.isc_dsql_execute Variable 3899
IBIntf.isc_dsql_execute_immediate Variable 3899
IBIntf.isc_dsql_execute2 Variable 3900
IBIntf.isc_dsql_fetch Variable 3900
IBIntf.isc_dsql_free_statement Variable 3900
IBIntf.isc_dsql_prepare Variable 3901
IBIntf.isc_dsql_set_cursor_name Variable 3901
IBIntf.isc_dsql_sql_info Variable 3901
IBIntf.isc_encode_date Variable 3902
IBIntf.isc_encode_sql_date Variable 3902
IBIntf.isc_encode_sql_time Variable 3902
IBIntf.isc_encode_timestamp Variable 3903
IBIntf.isc_event_block Variable 3903
IBIntf.isc_event_counts Variable 3903
IBIntf.isc_free Variable 3904
IBIntf.isc_get_segment Variable 3904
IBIntf.isc_install_clear_options Variable 3904
IBIntf.isc_install_execute Variable 3905
IBIntf.isc_install_get_info Variable 3905
IBIntf.isc_install_get_message Variable 3905
IBIntf.isc_install_load_external_text Variable 3906
IBIntf.isc_install_precheck Variable 3906
IBIntf.isc_install_set_option Variable 3906

IBIntf.isc_install_unset_option Variable 3907
IBIntf.isc_interprete Variable 3907
IBIntf.isc_modify_user Variable 3907
IBIntf.isc_open_blob2 Variable 3908
IBIntf.isc_prepare_transaction Variable 3908
IBIntf.isc_prepare_transaction2 Variable 3908
IBIntf.isc_put_segment Variable 3909
IBIntf.isc_que_events Variable 3909
IBIntf.isc_rollback_retaining Variable 3909
IBIntf.isc_rollback_transaction Variable 3910
IBIntf.isc_service_attach Variable 3910
IBIntf.isc_service_detach Variable 3910
IBIntf.isc_service_query Variable 3911
IBIntf.isc_service_start Variable 3911
IBIntf.isc_sql_interprete Variable 3911
IBIntf.isc_sqlcode Variable 3912
IBIntf.isc_start_multiple Variable 3912
IBIntf.isc_uninstall_execute Variable 3912
IBIntf.isc_uninstall_precheck Variable 3913
IBIntf.isc_vax_integer Variable 3913

IBQuery Namespace 3915

Classes 3916
TIBQuery Class 3916
TIBQuery Members 3916
TIBQuery.Create Constructor 3921
TIBQuery.Destroy Destructor 3921
TIBQuery Methods 3921
TIBQuery Properties 3927
TIBQuery Events 3934

IBServices Namespace 3937

Classes 3939
TConfigFileData Class 3939
TConfigFileData Members 3940
TConfigFileData.Create Constructor 3940
TConfigFileData.Destroy Destructor 3940
TConfigFileData Fields 3940

TConfigParams Class 3941
TConfigParams Members 3941
TConfigParams.Create Constructor 3942
TConfigParams.Destroy Destructor 3942
TConfigParams Fields 3942
TDatabaseInfo Class 3944
TDatabaseInfo Members 3944
TDatabaseInfo.Create Constructor 3945
TDatabaseInfo.Destroy Destructor 3945
TDatabaseInfo Fields 3945
TIBBackupRestoreService Class 3946
TIBBackupRestoreService Members 3946
TIBBackupRestoreService Properties 3947
TIBBackupService Class 3948
TIBBackupService Members 3948
TIBBackupService.Create Constructor 3950
TIBBackupService.Destroy Destructor 3950
TIBBackupService Methods 3950
TIBBackupService Properties 3950
TIBConfigService Class 3952
TIBConfigService Members 3952
TIBConfigService Methods 3953
TIBConfigService Properties 3957
TIBControlAndQueryService Class 3957
TIBControlAndQueryService Members 3958
TIBControlAndQueryService.Create Constructor 3959
TIBControlAndQueryService Methods 3959
TIBControlAndQueryService Properties 3960
TIBControlService Class 3961
TIBControlService Members 3962
TIBControlService.Create Constructor 3963
TIBControlService Methods 3963
TIBControlService Properties 3964
TIBCustomService Class 3965
TIBCustomService Members 3966
TIBCustomService.Create Constructor 3966
TIBCustomService.Destroy Destructor 3967
TIBCustomService Methods 3967
TIBCustomService Properties 3969

TIBCustomService Events 3973
TIBLicensingService Class 3973
TIBLicensingService Members 3974
TIBLicensingService Methods 3975
TIBLicensingService Properties 3976
TIBLogService Class 3977
TIBLogService Members 3978
TIBLogService Methods 3979
TIBRestoreService Class 3979
TIBRestoreService Members 3980
TIBRestoreService.Create Constructor 3981
TIBRestoreService.Destroy Destructor 3981
TIBRestoreService Methods 3981
TIBRestoreService Properties 3982
TIBSecurityService Class 3983
TIBSecurityService Members 3984
TIBSecurityService.Create Constructor 3985
TIBSecurityService.Destroy Destructor 3986
TIBSecurityService Methods 3986
TIBSecurityService Properties 3988
TIBServerProperties Class 3992
TIBServerProperties Members 3992
TIBServerProperties.Create Constructor 3993
TIBServerProperties.Destroy Destructor 3994
TIBServerProperties Methods 3994
TIBServerProperties Properties 3996
TIBStatisticalService Class 3998
TIBStatisticalService Members 3999
TIBStatisticalService Methods 4000
TIBStatisticalService Properties 4000
TIBValidationService Class 4001
TIBValidationService Members 4001
TIBValidationService.Create Constructor 4003
TIBValidationService.Destroy Destructor 4003
TIBValidationService Methods 4003
TIBValidationService Properties 4004
TLicenseInfo Class 4006
TLicenseInfo Members 4006
TLicenseInfo.Create Constructor 4007

TLicenseInfo.Destroy Destructor 4007
TLicenseInfo Fields 4007
TLicenseMaskInfo Class 4008
TLicenseMaskInfo Members 4009
TLicenseMaskInfo Fields 4009
TLimboTransactionInfo Class 4010
TLimboTransactionInfo Members 4010
TLimboTransactionInfo Fields 4011
TUserInfo Class 4013
TUserInfo Members 4013
TUserInfo Fields 4013
TVersionInfo Class 4015
TVersionInfo Members 4016
TVersionInfo Fields 4016

Structs, Records, Enums 4018
IBServices.TBackupOption Enumeration 4018
IBServices.TLicensingAction Enumeration 4019
IBServices.TOutputBufferOption Enumeration 4019
IBServices.TPropertyOption Enumeration 4020
IBServices.TProtocol Enumeration 4020
IBServices.TRestoreOption Enumeration 4021
IBServices.TSecurityAction Enumeration 4021
IBServices.TSecurityModifyParam Enumeration 4022
IBServices.TShutdownMode Enumeration 4022
IBServices.TStatOption Enumeration 4023
IBServices.TTransactionAction Enumeration 4023
IBServices.TTransactionAdvise Enumeration 4024
IBServices.TTransactionGlobalAction Enumeration 4024
IBServices.TTransactionState Enumeration 4025
IBServices.TValidateOption Enumeration 4026

Types 4027
IBServices.TBackupOptions Type 4027
IBServices.TLoginEvent Type 4028
IBServices.TPropertyOptions Type 4028
IBServices.TRestoreOptions Type 4029
IBServices.TSecurityModifyParams Type 4030
IBServices.TStatOptions Type 4030
IBServices.TValidateOptions Type 4031

Constants **4032**
IBServices.DefaultBufferSize Constant 4032
IBServices.SPBConstantNames Constant 4032
IBServices.SPBConstantValues Constant 4033
IBServices.SPBPrefix Constant 4033

IBSQL Namespace 4035

Classes **4036**
TIBBatch Class 4036
TIBBatch Members 4036
TIBBatch Fields 4037
TIBBatch Methods 4038
TIBBatch Properties 4038
TIBBatchInput Class 4039
TIBBatchInput Members 4040
TIBBatchInput Methods 4040
TIBBatchOutput Class 4040
TIBBatchOutput Members 4041
TIBBatchOutput Methods 4041
TIBInputDelimitedFile Class 4042
TIBInputDelimitedFile Members 4042
TIBInputDelimitedFile.Destroy Destructor 4043
TIBInputDelimitedFile Fields 4043
TIBInputDelimitedFile Methods 4045
TIBInputDelimitedFile Properties 4046
TIBInputRawFile Class 4047
TIBInputRawFile Members 4047
TIBInputRawFile.Destroy Destructor 4048
TIBInputRawFile Fields 4048
TIBInputRawFile Methods 4048
TIBOutputDelimitedFile Class 4049
TIBOutputDelimitedFile Members 4050
TIBOutputDelimitedFile.Destroy Destructor 4050
TIBOutputDelimitedFile Fields 4050
TIBOutputDelimitedFile Methods 4051
TIBOutputDelimitedFile Properties 4052
TIBOutputRawFile Class 4053
TIBOutputRawFile Members 4054

TIBOutputRawFile.Destroy Destructor 4054
TIBOutputRawFile Fields 4054
TIBOutputRawFile Methods 4055
TIBSQL Class 4055
TIBSQL Members 4056
TIBSQL.Create Constructor 4058
TIBSQL.Destroy Destructor 4058
TIBSQL Fields 4058
TIBSQL Methods 4062
TIBSQL Properties 4071
TIBSQL Events 4077
TIBXSQLDA Class 4078
TIBXSQLDA Members 4078
TIBXSQLDA.Create Constructor 4079
TIBXSQLDA.Destroy Destructor 4079
TIBXSQLDA Fields 4080
TIBXSQLDA Methods 4081
TIBXSQLDA Properties 4084
TIBXSQLVAR Class 4086
TIBXSQLVAR Members 4087
TIBXSQLVAR.Create Constructor 4087
TIBXSQLVAR Methods 4088
TIBXSQLVAR Properties 4090

Structs, Records, Enums **4098**
IBSQL.TIBSQLTypes Enumeration 4098

Types **4100**
IBSQL.TIBXSQLVARArray Type 4100

IBSQLMonitor Namespace 4101

Classes **4103**
TIBCustomSQLMonitor Class 4103
TIBCustomSQLMonitor Members 4103
TIBCustomSQLMonitor.Create Constructor 4104
TIBCustomSQLMonitor.Destroy Destructor 4104
TIBCustomSQLMonitor Methods 4104
TIBCustomSQLMonitor Properties 4105
TIBCustomSQLMonitor Events 4106
TIBSQLMonitor Class 4106

TIBSQLMonitor Members 4107
TIBSQLMonitor Properties 4107
TIBSQLMonitor Events 4108

Interfaces 4109
IIBSQLMonitorHook Interface 4109
IIBSQLMonitorHook Members 4109
IIBSQLMonitorHook Methods 4110
IIBSQLMonitorHook Properties 4115

Functions 4117
IBSQLMonitor.DisableMonitoring Function 4117
IBSQLMonitor.EnableMonitoring Function 4117
IBSQLMonitor.MonitorHook Function 4118
IBSQLMonitor.MonitoringEnabled Function 4118

Types 4119
IBSQLMonitor.TSQLEvent Type 4119

Constants 4120
IBSQLMonitor.WM_IBSQL_SQL_EVENT Constant 4120
IBSQLMonitor.WM_MAX_IBSQL_MONITOR Constant 4120
IBSQLMonitor.WM_MIN_IBSQL_MONITOR Constant 4120

IBStoredProc Namespace 4123

Classes 4124
TIBStoredProc Class 4124
TIBStoredProc Members 4124
TIBStoredProc.Create Constructor 4128
TIBStoredProc.Destroy Destructor 4129
TIBStoredProc Methods 4129
TIBStoredProc Properties 4134
TIBStoredProc Events 4137

IBTable Namespace 4141

Classes 4142
TIBTable Class 4142
TIBTable Members 4142
TIBTable.Create Constructor 4147
TIBTable.Destroy Destructor 4147

TIBTable Methods 4147
TIBTable Properties 4156
TIBTable Events 4165

Structs, Records, Enums 4168
IBTable.TIBTableType Enumeration 4168

Types 4169
IBTable.TIBTableTypes Type 4169
IBTable.TIndexName Type 4169

IBUpdateSQL Namespace 4171

Classes 4172
TIBUpdateSQL Class 4172
TIBUpdateSQL Members 4172
TIBUpdateSQL.Create Constructor 4173
TIBUpdateSQL.Destroy Destructor 4173
TIBUpdateSQL Methods 4174
TIBUpdateSQL Properties 4176

IBUtils Namespace 4181

Functions 4182
IBUtils.AddIBParamSQLForDetail Function 4182
IBUtils.ExtractIdentifier Function 4182
IBUtils.FormatIdentifier Function 4183
IBUtils.FormatIdentifierValue Function 4183
IBUtils.Max Function 4183
IBUtils.Min Function 4184
IBUtils.QuoteIdentifier Function 4184
IBUtils.RandomInteger Function 4184
IBUtils.RandomString Function 4185
IBUtils.StripString Function 4185

Constants 4186
IBUtils.CR Constant 4186
IBUtils.CRLF Constant 4186
IBUtils.LF Constant 4187
IBUtils.NULL_TERMINATOR Constant 4187
IBUtils.TAB Constant 4187

ImgList Namespace 4189

Classes **4190**

TChangeLink Class 4190

TChangeLink Members 4190

TChangeLink.Destroy Destructor 4191

TChangeLink Methods 4191

TChangeLink Properties 4192

TChangeLink Events 4192

TCustomImageList Class 4193

TCustomImageList Members 4193

TCustomImageList.Create Constructor 4195

TCustomImageList.CreateSize Constructor 4195

TCustomImageList.Destroy Destructor 4196

TCustomImageList Methods 4196

TCustomImageList Properties 4217

TCustomImageList Events 4226

Structs, Records, Enums **4227**

ImgList.TColorDepth Enumeration 4227

ImgList.TDrawingStyle Enumeration 4228

ImgList.TImageType Enumeration 4229

ImgList.TLoadResource Enumeration 4229

ImgList.TResType Enumeration 4230

Types **4231**

ImgList.TImageIndex Type 4231

ImgList.TImageListHandle Type 4231

ImgList.TLoadResources Type 4232

ImgList.TOverlay Type 4232

IMouse Namespace 4233

Classes **4235**

TPanningWindow Class 4235

TPanningWindow Members 4235

TPanningWindow.Create Constructor 4236

TPanningWindow.Destroy Destructor 4236

TPanningWindow Methods 4237

TPanningWindow Properties 4245

TPanningWindow Events 4246

Functions **4247**

IMouse.StartPan Function 4247

Structs, Records, Enums **4248**

IMouse.TPanDirection Enumeration 4248

IMouse.TPanOption Enumeration 4248

Types **4250**

IMouse.TPanDirections Type 4250

IMouse.TPanOptions Type 4250

IMouse.TUpdateEvent Type 4250

Constants **4252**

IMouse.crPanAll Constant 4252

IMouse.crPanDown Constant 4252

IMouse.crPanDownLeft Constant 4253

IMouse.crPanDownRight Constant 4253

IMouse.crPanLeft Constant 4253

IMouse.crPanLeftRight Constant 4254

IMouse.crPanRight Constant 4254

IMouse.crPanUp Constant 4254

IMouse.crPanUpDown Constant 4255

IMouse.crPanUpLeft Constant 4255

IMouse.crPanUpRight Constant 4255

IniFiles Namespace 4257

Classes **4258**

EIniFileException Class 4258

TCustomIniFile Class 4258

TCustomIniFile Members 4262

TCustomIniFile.Create Constructor 4262

TCustomIniFile Methods 4263

TCustomIniFile Properties 4290

THashedStringList Class 4291

THashedStringList Members 4292

THashedStringList.Destroy Destructor 4292

THashedStringList Methods 4292

TIniFile Class 4294

TIniFile Members 4297

TIniFile.Destroy Destructor 4298
TIniFile Methods 4298
TMemIniFile Class 4307
TMemIniFile Members 4310
Create Constructor 4311
TMemIniFile.Destroy Destructor 4312
TMemIniFile Methods 4312
TMemIniFile Properties 4323
TStringHash Class 4324
TStringHash Members 4325
TStringHash.Create Constructor 4325
TStringHash.Destroy Destructor 4325
TStringHash Methods 4326

Structs, Records, Enums **4328**
IniFiles.THashItem Record 4328

Types **4329**
IniFiles.PHashItem Type 4329
IniFiles.PPHashItem Type 4329

IntfInfo Namespace 4331

Classes **4333**
EInterfaceRTTIException Class 4333

Functions **4334**
IntfInfo.GetClsMemberTypeInfo Function 4334
IntfInfo.GetDynArrayElTypeInfo Function 4334
IntfInfo.GetDynArrayNextInfo Function 4335
IntfInfo.GetDynArrayNextInfo2 Function 4335
IntfInfo.GetIntfMetaData Function 4335
IntfInfo.GetIntfMetaData Function 4336
IntfInfo.GetMethNum Function 4336
IntfInfo.GetPropListFlat Function 4336
IntfInfo.IsStoredPropConst Function 4337
IntfInfo.OtherTypeName Function 4337
IntfInfo.SameTypeInfo Function 4337
IntfInfo.TypeNamesMatch Function 4338

Structs, Records, Enums **4339**
IntfInfo.TCallConv Enumeration 4339

IntfInfo.TFillMethodArrayOpt Enumeration 4339
IntfInfo.TIntfMetaData Record 4340
IntfInfo.TIntfMethEntry Record 4340
IntfInfo.TIntfParamEntry Record 4341

Types **4342**
IntfInfo.PIntfMetaData Type 4342
IntfInfo.PIntfMethEntry Type 4342
IntfInfo.PIntfParamEntry Type 4343
IntfInfo.TIntfMethEntryArray Type 4343
IntfInfo.TIntfParamEntryArray Type 4343
IntfInfo.TPIntfMethEntryArray Type 4344

Constants **4345**
IntfInfo.CallingConventionName Constant 4345
IntfInfo.TypeInfoNames Constant 4345

InvokeRegistry Namespace 4347

Classes **4350**
ERemotableException Class 4350
ERemotableException Members 4351
Create Constructor 4351
ERemotableException Properties 4352
ETypeRegistryException Class 4353
TDataContext Class 4354
TDataContext Members 4355
TDataContext.Create Constructor 4355
TDataContext.Destroy Destructor 4355
TDataContext Fields 4356
TDataContext Methods 4358
THeaderList Class 4363
THeaderList Members 4363
THeaderList.Create Constructor 4364
THeaderList.Destroy Destructor 4364
THeaderList Methods 4365
THeaderList Properties 4367
TInvContext Class 4369
TInvContext Members 4370
TInvContext Methods 4371
TInvokableClass Class 4373

TInvokableClass Members 4374
TInvokableClass.Create Constructor 4375
TInvokableClass Fields 4375
TInvokableClass Methods 4376
TInvokableClass Properties 4379
TInvokableClassRegistry Class 4379
TInvokableClassRegistry Members 4380
TInvokableClassRegistry.Create Constructor 4381
TInvokableClassRegistry.Destroy Destructor 4382
TInvokableClassRegistry Methods 4382
TRemotable Class 4408
TRemotable Members 4409
TRemotable.Create Constructor 4410
TRemotable.Destroy Destructor 4410
TRemotable Fields 4411
TRemotable Methods 4411
TRemotable Properties 4412
TRemotableTypeRegistry Class 4414
TRemotableTypeRegistry Members 4414
TRemotableTypeRegistry.Create Constructor 4415
TRemotableTypeRegistry.Destroy Destructor 4416
TRemotableTypeRegistry Methods 4416
TRemotableTypeRegistry Properties 4431
TRemotableXS Class 4432
TRemotableXS Members 4433
TRemotableXS Methods 4433
TSOAPAttachment Class 4435
TSOAPAttachment Members 4437
TSOAPAttachment.Create Constructor 4437
TSOAPAttachment.Destroy Destructor 4438
TSOAPAttachment Methods 4438
TSOAPAttachment Properties 4444
TSOAPHeader Class 4448
TSOAPHeader Members 4449
TSOAPHeader Methods 4450
TSOAPHeader Properties 4451
TSOAPHeaders Class 4452
TSOAPHeaders Members 4453
TSOAPHeaders Methods 4454

TSOAPHeadersBase Class 4457
TSOAPHeadersBase Members 4457
TSOAPHeadersBase Fields 4457
TSOAPHeadersBase Methods 4458

Interfaces 4460
IHeadersSetter Interface 4460
IHeadersSetter Members 4460
IHeadersSetter Methods 4460
IObjConverter Interface 4461
IObjConverter Members 4462
IObjConverter Methods 4462
ISOAPHeaders Interface 4464
ISOAPHeaders Members 4465
ISOAPHeaders Methods 4465
ISOAPHeaders Properties 4468

Functions 4470
InvokeRegistry.GetRemotableDataContext Function 4470
InvokeRegistry.InvRegistry Function 4471
InvokeRegistry.RemClassRegistry Function 4471
InvokeRegistry.RemTypeRegistry Function 4472
InvokeRegistry.SetRemotableDataContext Function 4473
InvokeRegistry.SubstituteStrings Function 4474

Structs, Records, Enums 4475
InvokeRegistry.eHeaderMethodType Enumeration 4475
InvokeRegistry.ExtNameMapItem Record 4476
InvokeRegistry.IntfExceptionItem Record 4476
InvokeRegistry.IntfHeaderItem Record 4477
InvokeRegistry.InvRegClassEntry Record 4478
InvokeRegistry.InvRegIntfEntry Record 4478
InvokeRegistry.MethParamNameMapItem Record 4480
InvokeRegistry.ObjectConvertOptions Enumeration 4480
InvokeRegistry.SerializationOptions Enumeration 4481
InvokeRegistry.TDynToClear Record 4482
InvokeRegistry.TIntfInvokeOption Enumeration 4483
InvokeRegistry.TObjMultiOptions Enumeration 4484
InvokeRegistry.TRemHeaderEntry Record 4485
InvokeRegistry.TRemRegEntry Record 4485

Types **4487**
InvokeRegistry.ERemotableExceptionClass Type 4487
InvokeRegistry.InvString Type 4488
InvokeRegistry.InvStringDynArray Type 4488
InvokeRegistry.PTRemotable Type 4488
InvokeRegistry.TCreateInstanceProc Type 4489
InvokeRegistry.TExceptionItemArray Type 4489
InvokeRegistry.THeaderItemArray Type 4489
InvokeRegistry.THeaderMethodTypeArray Type 4490
InvokeRegistry.TIntfInvokeOptions Type 4490
InvokeRegistry.TInvokableClassClass Type 4490
InvokeRegistry.TInvRegIntfEntryArray Type 4491
InvokeRegistry.TObjectConvertOptions Type 4491
InvokeRegistry.TPascalRemotableClassRegistry Type 4491
InvokeRegistry.TPascalRemotableTypeRegistry Type 4492
InvokeRegistry.TRemotableClass Type 4492
InvokeRegistry.TRemotableClassRegistry Type 4492
InvokeRegistry.TRemotableXSClass Type 4493
InvokeRegistry.TRemRegEntryArray Type 4493
InvokeRegistry.TRequiredArray Type 4493
InvokeRegistry.TSerializationOptions Type 4494
InvokeRegistry.TSOAPAttachmentClass Type 4494
InvokeRegistry.TSOAPHeaderClass Type 4494

Variables **4496**
InvokeRegistry.AppNameSpacePrefix Variable 4496

Constants **4497**
InvokeRegistry.AS_ATTRIBUTE Constant 4497
InvokeRegistry.XMLBase64Types Constant 4497
InvokeRegistry.XMLSchemaInstNamepspaces Constant 4498
InvokeRegistry.XMLSchemaNamepspaces Constant 4498

ISAPIApp Namespace 4499

Classes **4500**
TISAPIApplication Class 4500
TISAPIApplication Members 4501
TISAPIApplication.Create Constructor 4501
TISAPIApplication.Destroy Destructor 4501

TISAPIApplication Methods 4502
TISAPIFactory Class 4503
TISAPIFactory Members 4503
TISAPIFactory.Create Constructor 4503
TISAPIFactory Methods 4504
Functions 4505
ISAPIApp.GetExtensionVersion Function 4505
ISAPIApp.HttpExtensionProc Function 4505
ISAPIApp.TerminateExtension Function 4506
Variables 4507
ISAPIApp.DispatchThread Variable 4507
IsapiHTTP Namespace 4509
Classes 4510
TISAPIRequest Class 4510
TISAPIRequest Members 4510
TISAPIRequest.Create Constructor 4512
TISAPIRequest Methods 4512
TISAPIRequest Properties 4517
TISAPIResponse Class 4518
TISAPIResponse Members 4519
TISAPIResponse.Create Constructor 4520
TISAPIResponse Methods 4520
Java Namespace 4529
Java.Sql Namespace 4530
Classes 4530
SQLException Class 4530
ListActns Namespace 4533
Classes 4535
TCustomListAction Class 4535
TCustomListAction Members 4536
TCustomListAction.Create Constructor 4536
TCustomListAction Methods 4537
TCustomListAction Properties 4539

TCustomListAction Events 4542
TCustomStaticListAction Class 4543
TCustomStaticListAction Members 4543
TCustomStaticListAction.Create Constructor 4544
TCustomStaticListAction.Destroy Destructor 4544
TCustomStaticListAction Methods 4545
TCustomStaticListAction Properties 4546
TCustomStaticListAction Events 4547
TCustomVirtualListAction Class 4548
TCustomVirtualListAction Members 4548
TCustomVirtualListAction Methods 4549
TCustomVirtualListAction Properties 4550
TCustomVirtualListAction Events 4550
TListActionLink Class 4550
TListActionLink Members 4551
TListActionLink Methods 4551
TListControlItem Class 4554
TListControlItem Members 4554
TListControlItem.Create Constructor 4555
TListControlItem Fields 4555
TListControlItem Methods 4556
TListControlItem Properties 4559
TListControlItems Class 4560
TListControlItems Members 4561
TListControlItems.Create Constructor 4562
TListControlItems Methods 4562
TListControlItems Properties 4565
TListControlItems Events 4567
TStaticListAction Class 4567
TStaticListAction Members 4568
TStaticListAction Properties 4569
TStaticListAction Events 4574
TStaticListItems Class 4576
TStaticListItems Members 4576
TStaticListItems Methods 4577
TVirtualListAction Class 4578
TVirtualListAction Members 4579
TVirtualListAction Properties 4580
TVirtualListAction Events 4584

Structs, Records, Enums **4586**
ListActns.TListItemsSortType Enumeration 4586

Types **4587**
ListActns.TGetItemCountEvent Type 4587
ListActns.TGetItemEvent Type 4587
ListActns.TGetVirtualItemEvent Type 4588
ListActns.TItemSelectedEvent Type 4588
ListActns.TListCompareEvent Type 4588
ListActns.TListControlItemClass Type 4589
ListActns.TListItemsCompare Type 4589

Mask Namespace 4591

Classes **4592**
EDBEditError Class 4592
TCustomMaskEdit Class 4592
TCustomMaskEdit Members 4593
TCustomMaskEdit.Create Constructor 4594
TCustomMaskEdit Methods 4594
TCustomMaskEdit Properties 4604
TMaskEdit Class 4610
TMaskEdit Members 4610
TMaskEdit Properties 4613
TMaskEdit Events 4648

Types **4681**
Mask.TMaskedState Type 4681

Masks Namespace 4683

Classes **4684**
EMaskException Class 4684
TMask Class 4684
TMask Members 4685
TMask.Create Constructor 4685
TMask.Destroy Destructor 4686
TMask Methods 4686

Functions **4688**
Masks.MatchesMask Function 4688

MaskUtils Namespace 4689

Functions **4691**
MaskUtils.FormatMaskText Function 4691
MaskUtils.IsLiteralChar Function 4693
MaskUtils.MaskDoFormatText Function 4693
MaskUtils.MaskGetCharType Function 4693
MaskUtils.MaskGetCurrentDirectives Function 4694
MaskUtils.MaskGetFldSeparator Function 4694
MaskUtils.MaskGetMaskBlank Function 4695
MaskUtils.MaskGetMaskSave Function 4695
MaskUtils.MaskIntlLiteralToChar Function 4695
MaskUtils.MaskOffsetToOffset Function 4696
MaskUtils.MaskOffsetToWideOffset Function 4696
MaskUtils.OffsetToMaskOffset Function 4697
MaskUtils.PadInputLiterals Function 4697

Structs, Records, Enums **4698**
MaskUtils.TMaskCharType Enumeration 4698

Types **4699**
MaskUtils.TEditMask Type 4699
MaskUtils.TMaskDirectives Type 4700
MaskUtils.TMaskedText Type 4701

Variables **4702**
MaskUtils.DefaultBlank Variable 4702
MaskUtils.MaskFieldSeparator Variable 4702
MaskUtils.MaskNoSave Variable 4703

Constants **4704**
MaskUtils.mDirLiteral Constant 4704
MaskUtils.mDirLowerCase Constant 4704
MaskUtils.mDirReverse Constant 4705
MaskUtils.mDirUpperCase Constant 4705
MaskUtils.mMskAlpha Constant 4706
MaskUtils.mMskAlphaNum Constant 4706
MaskUtils.mMskAlphaNumOpt Constant 4706
MaskUtils.mMskAlphaOpt Constant 4707
MaskUtils.mMskAscii Constant 4707
MaskUtils.mMskAsciiOpt Constant 4708

MaskUtils.mMskDateSeparator Constant 4708
MaskUtils.mMskNumeric Constant 4708
MaskUtils.mMskNumericOpt Constant 4709
MaskUtils.mMskNumSymOpt Constant 4709
MaskUtils.mMskTimeSeparator Constant 4710

Math Namespace 4711

Classes 4715

EInvalidArgument Class 4715

Functions 4717

Math.ArcCos Function 4719
Math.ArcCosh Function 4720
Math.ArcCot Function 4720
Math.ArcCotH Function 4721
Math.ArcCsc Function 4721
Math.ArcCscH Function 4722
Math.ArcSec Function 4722
Math.ArcSecH Function 4722
Math.ArcSin Function 4723
Math.ArcSinh Function 4723
Math.ArcTan2 Function 4724
Math.ArcTanh Function 4724
Math.Ceil Function 4725
Math.ClearExceptions Function 4725
Math.CompareValue Function 4726
Math.Cosecant Function 4726
Math.Cosh Function 4727
Math.Cot Function 4727
Math.Cotan Function 4728
Math.CotH Function 4728
Math.Csc Function 4729
Math.CscH Function 4729
Math.CycleToDeg Function 4730
Math.CycleToGrad Function 4730
Math.CycleToRad Function 4730
Math.DegToCycle Function 4731
Math.DegToGrad Function 4731
Math.DegToRad Function 4732

Math.DivMod Function 4732
Math.DoubleDecliningBalance Function 4732
Math.EnsureRange Function 4733
Math.Floor Function 4733
Math.Frexp Function 4734
Math.FutureValue Function 4734
Math.GetExceptionMask Function 4735
Math.GetPrecisionMode Function 4735
Math.GetRoundMode Function 4736
Math.GradToCycle Function 4737
Math.GradToDeg Function 4737
Math.GradToRad Function 4737
Math.Hypot Function 4738
Math.IfThen Function 4738
Math.InRange Function 4738
Math.InterestPayment Function 4739
Math.InterestRate Function 4740
Math.InternalRateOfReturn Function 4740
Math.IntPower Function 4741
Math.IsInfinite Function 4741
Math.IsNan Function 4742
Math.IsZero Function 4742
Math.Ldexp Function 4743
Math.LnXP1 Function 4743
Math.Log10 Function 4743
Math.Log2 Function 4744
Math.LogN Function 4744
Math.Max Function 4745
Math.MaxIntValue Function 4745
Math.MaxValue Function 4746
Math.Mean Function 4746
Math.MeanAndStdDev Function 4747
Math.Min Function 4747
Math.MinIntValue Function 4748
Math.MinValue Function 4748
Math.MomentSkewKurtosis Function 4749
Math.NetPresentValue Function 4749
Math.Norm Function 4750
Math.NumberOfPeriods Function 4750

Math.Payment Function 4751
Math.PeriodPayment Function 4751
Math.Poly Function 4751
Math.PopnStdDev Function 4752
Math.PopnVariance Function 4752
Math.Power Function 4753
Math.PresentValue Function 4753
Math.RadToCycle Function 4754
Math.RadToDeg Function 4754
Math.RadToGrad Function 4754
Math.RandG Function 4755
Math.RandomFrom Function 4755
Math.RandomRange Function 4756
Math.RoundTo Function 4756
Math.SameValue Function 4757
Math.Sec Function 4758
Math.Secant Function 4758
Math.SecH Function 4759
Math.SetExceptionMask Function 4759
Math.SetPrecisionMode Function 4760
Math.SetRoundMode Function 4760
Math.Sign Function 4761
Math.SimpleRoundTo Function 4761
Math.SinCos Function 4762
Math.Sinh Function 4762
Math.SLNDepreciation Function 4763
Math.StdDev Function 4763
Math.Sum Function 4764
Math.SumInt Function 4764
Math.SumOfSquares Function 4765
Math.SumsAndSquares Function 4765
Math.SYDDepreciation Function 4766
Math.Tan Function 4766
Math.Tanh Function 4767
Math.TotalVariance Function 4767
Math.Variance Function 4768

Structs, Records, Enums **4769**

Math.TFPUException Enumeration 4769
Math.TFPUPrecisionMode Enumeration 4770

Math.TFPURoundingMode Enumeration 4771
Math.TPaymentTime Enumeration 4772

Types **4773**
Math.TFPUExceptionMask Type 4773
Math.TRoundToEXRangeExtended Type 4774
Math.TRoundToRange Type 4774
Math.TValueSign Type 4774

Constants **4776**
Math.Infinity Constant 4776
Math.MaxComp Constant 4777
Math.MaxDouble Constant 4777
Math.MaxExtended Constant 4777
Math.MaxSingle Constant 4778
Math.MinComp Constant 4778
Math.MinDouble Constant 4779
Math.MinExtended Constant 4779
Math.MinSingle Constant 4779
Math.NaN Constant 4780
Math.NegativeValue Constant 4780
Math.NegInfinity Constant 4781
Math.PositiveValue Constant 4781
Math.ZeroValue Constant 4782

MConnect Namespace 4783

Classes **4784**
TCOMConnection Class 4784
TCOMConnection Members 4785
TCOMConnection Methods 4785
TCustomObjectBroker Class 4787
TCustomObjectBroker Members 4788
TCustomObjectBroker Methods 4788
TDCOMConnection Class 4790
TDCOMConnection Members 4791
TDCOMConnection.Create Constructor 4792
TDCOMConnection Methods 4792
TDCOMConnection Properties 4793
TDispatchAppServer Class 4794
TDispatchAppServer Members 4795

TDispatchAppServer.Create Constructor 4795
TDispatchAppServer Methods 4795
TDispatchConnection Class 4796
TDispatchConnection Members 4797
TDispatchConnection.Create Constructor 4797
TDispatchConnection Methods 4798
TDispatchConnection Properties 4801
TDispatchConnection Events 4808
TOLEnterpriseConnection Class 4811
TOLEnterpriseConnection Members 4812
TOLEnterpriseConnection Methods 4813
TOLEnterpriseConnection Properties 4813
TSharedConnection Class 4814
TSharedConnection Members 4815
TSharedConnection.Create Constructor 4816
TSharedConnection.Destroy Destructor 4816
TSharedConnection Methods 4817
TSharedConnection Properties 4819
TSharedConnection Events 4820

Functions **4823**
MConnect.GetMIDASAppServerList Function 4823

Menus Namespace 4825

Classes **4827**
EMenuError Class 4827
TMainMenu Class 4828
TMainMenu Members 4828
TMainMenu Methods 4829
TMainMenu Properties 4833
TMainMenu Events 4836
TMenu Class 4837
TMenu Members 4838
TMenu.Create Constructor 4839
TMenu.Destroy Destructor 4840
TMenu Methods 4840
TMenu Properties 4849
TMenu Events 4857
TMenuActionLink Class 4858

TMenuActionLink Members 4859
TMenuActionLink Fields 4859
TMenuActionLink Methods 4860
TMenuItem Class 4869
TMenuItem Members 4870
TMenuItem.Create Constructor 4872
TMenuItem.Destroy Destructor 4873
TMenuItem Methods 4874
TMenuItem Properties 4898
TMenuItem Events 4922
TMenuItemEnumerator Class 4929
TMenuItemEnumerator Members 4930
TMenuItemEnumerator.Create Constructor 4930
TMenuItemEnumerator Methods 4930
TMenuItemEnumerator Properties 4931
TMenuItemStack Class 4931
TMenuItemStack Members 4931
TMenuItemStack Methods 4932
TPopupList Class 4932
TPopupList Members 4933
TPopupList Fields 4933
TPopupList Methods 4933
TPopupList Properties 4935
TPopupMenu Class 4935
TPopupMenu Members 4936
TPopupMenu.Create Constructor 4938
TPopupMenu.Destroy Destructor 4938
TPopupMenu Methods 4938
TPopupMenu Properties 4940
TPopupMenu Events 4949

Functions **4951**

Menus.AnsiSameCaption Function 4951
Menus.DrawMenuItem Function 4952
Menus.GetHotkey Function 4952
Menus.IsAltGRPressed Function 4952
Menus.NewItem Function 4953
Menus.NewLine Function 4954
Menus.NewMenu Function 4955
Menus.NewPopupMenu Function 4956

Menus.NewSubMenu Function 4957
Menus.SameCaption Function 4958
Menus.ShortCut Function 4958
Menus.ShortCutFromMessage Function 4959
Menus.ShortCutToKey Function 4960
Menus.ShortCutToText Function 4961
Menus.StripHotkey Function 4962
Menus.TextToShortCut Function 4963

Structs, Records, Enums **4965**
Menus.TFindItemKind Enumeration 4965
Menus.TMenuAnimations Enumeration 4966
Menus.TMenuBreak Enumeration 4966
Menus.TMenuItemAutoFlag Enumeration 4967
Menus.TPopupAlignment Enumeration 4968
Menus.TTrackButton Enumeration 4968

Types **4970**
Menus.PMenuItem Type 4970
Menus.TAdvancedMenuDrawItemEvent Type 4970
Menus.TMenuActionLinkClass Type 4971
Menus.TMenuAnimation Type 4971
Menus.TMenuAutoFlag Type 4972
Menus.TMenuChangeEvent Type 4972
Menus.TMenuDrawItemEvent Type 4973
Menus.TMenuHandle Type 4973
Menus.TMenuMeasureItemEvent Type 4973

Variables **4975**
Menus.PopupList Variable 4975
Menus.ShortCutItems Variable 4975
Menus.ValidMenuHotkeys Variable 4975

Constants **4977**
Menus.cDialogSuffix Constant 4977
Menus.cHotkeyPrefix Constant 4977
Menus.cLineCaption Constant 4977

Messages Namespace 4979

Functions **4991**
Messages.SendGetIntMessage Function 4991

Messages.SendGetStructMessage Function 4992
Messages.SendStructMessage Function 4992
Messages.SendTextMessage Function 4993

Structs, Records, Enums **4995**

Messages.TMessage Record 4997
Messages.TMSHMouseWheel Record 4998
Messages.TWMActivate Record 4998
Messages.TWMActivateApp Record 4999
Messages.TWMAskCBFormatName Record 4999
Messages.TWMChangeCBChain Record 5000
Messages.TWMCharToItem Record 5000
Messages.TWMChooseFont_GetLogFont Record 5001
Messages.TWMCommand Record 5001
Messages.TWMCompacting Record 5002
Messages.TWMCompareItem Record 5002
Messages.TWMContextMenu Record 5003
Messages.TWMCopyData Record 5003
Messages.TWMCreate Record 5004
Messages.TWMCtlColor Record 5004
Messages.TWMDDE_Ack Record 5005
Messages.TWMDDE_Advise Record 5005
Messages.TWMDDE_Data Record 5006
Messages.TWMDDE_Execute Record 5006
Messages.TWMDDE_Initiate Record 5007
Messages.TWMDDE_Poke Record 5007
Messages.TWMDDE_Request Record 5008
Messages.TWMDDE_Terminate Record 5008
Messages.TWMDDE_Unadvise Record 5009
Messages.TWMDeleteItem Record 5009
Messages.TWMDevModeChange Record 5010
Messages.TWMDisplayChange Record 5010
Messages.TWMDrawItem Record 5011
Messages.TWMDropFiles Record 5011
Messages.TWMEnable Record 5012
Messages.TWMEndSession Record 5012
Messages.TWMEnterIdle Record 5013
Messages.TWMEnterMenuLoop Record 5013
Messages.TWMEraseBkgnd Record 5014
Messages.TWMGetIcon Record 5014

Messages.TWMGetMinMaxInfo Record 5015
Messages.TWMGetText Record 5015
Messages.TWMHelp Record 5016
Messages.TWMHotKey Record 5016
Messages.TWMHScrollClipboard Record 5017
Messages.TWMInitDialog Record 5017
Messages.TWMInitMenu Record 5018
Messages.TWMInitMenuPopup Record 5018
Messages.TWMKey Record 5019
Messages.TWMKillFocus Record 5019
Messages.TWMMDIActivate Record 5020
Messages.TWMMDICascade Record 5020
Messages.TWMMDICreate Record 5021
Messages.TWMMDIDestroy Record 5021
Messages.TWMMDIMaximize Record 5022
Messages.TWMMDINext Record 5022
Messages.TWMMDIRestore Record 5023
Messages.TWMMDISetMenu Record 5023
Messages.TWMMDITile Record 5024
Messages.TWMMeasureItem Record 5024
Messages.TWMMenuChar Record 5025
Messages.TWMMenuSelect Record 5025
Messages.TWMMouse Record 5026
Messages.TWMMouseActivate Record 5026
Messages.TWMMouseWheel Record 5027
Messages.TWMMove Record 5027
Messages.TWMMoving Record 5028
Messages.TWMNCActivate Record 5028
Messages.TWMNCCalcSize Record 5029
Messages.TWMNCCreate Record 5029
Messages.TWMNCHitMessage Record 5030
Messages.TWMNCHitTest Record 5030
Messages.TWMNCPaint Record 5031
Messages.TWMNextDlgCtl Record 5031
Messages.TWMNoParams Record 5032
Messages.TWMNotify Record 5032
Messages.TWMNotifyFormat Record 5033
Messages.TWMPaint Record 5033
Messages.TWMPaintClipboard Record 5034

Messages.TWMPaletteChanged Record 5034
Messages.TWMPaletteIsChanging Record 5035
Messages.TWMParentNotify Record 5035
Messages.TWMPower Record 5036
Messages.TWMPrint Record 5036
Messages.TWMQueryEndSession Record 5037
Messages.TWMQuit Record 5037
Messages.TWMRenderFormat Record 5038
Messages.TWMScroll Record 5038
Messages.TWMSetCursor Record 5039
Messages.TWMSetFocus Record 5039
Messages.TWMSetFont Record 5040
Messages.TWMSetHotKey Record 5040
Messages.TWMSetIcon Record 5041
Messages.TWMSetRedraw Record 5041
Messages.TWMSetText Record 5042
Messages.TWMSettingChange Record 5042
Messages.TWMShowWindow Record 5043
Messages.TWMSize Record 5043
Messages.TWMSizeClipboard Record 5044
Messages.TWMSpoolerStatus Record 5044
Messages.TWMStyleChange Record 5045
Messages.TWMSysCommand Record 5045
Messages.TWMSysDeadChar Record 5046
Messages.TWMSystemError Record 5046
Messages.TWMTimer Record 5047
Messages.TWMUIState Record 5047
Messages.TWMVScrollClipboard Record 5048
Messages.TWMWindowPosMsg Record 5048
Messages.TWMWinIniChange Record 5049

Types **5050**

Messages.PMessage Type 5051
Messages.TWMCancelMode Type 5052
Messages.TWMChangeUIState Type 5052
Messages.TWMChar Type 5052
Messages.TWMChildActivate Type 5053
Messages.TWMClear Type 5053
Messages.TWMClose Type 5053
Messages.TWMCopy Type 5054

Messages.TWMCtlColorBtn Type 5054
Messages.TWMCtlColorDlg Type 5054
Messages.TWMCtlColorEdit Type 5055
Messages.TWMCtlColorListbox Type 5055
Messages.TWMCtlColorMsgbox Type 5055
Messages.TWMCtlColorScrollbar Type 5056
Messages.TWMCtlColorStatic Type 5056
Messages.TWMCut Type 5056
Messages.TWMDeadChar Type 5057
Messages.TWMDestroy Type 5057
Messages.TWMDestroyClipboard Type 5057
Messages.TWMDrawClipboard Type 5058
Messages.TWMExitMenuLoop Type 5058
Messages.TWMFontChange Type 5058
Messages.TWMGetDlgCode Type 5059
Messages.TWMGetFont Type 5059
Messages.TWMGetHotKey Type 5059
Messages.TWMGetTextLength Type 5060
Messages.TWMHScroll Type 5060
Messages.TWMIconEraseBkgnd Type 5060
Messages.TWMKeyDown Type 5061
Messages.TWMKeyUp Type 5061
Messages.TWMLButtonDblClk Type 5061
Messages.TWMLButtonDown Type 5062
Messages.TWMLButtonUp Type 5062
Messages.TWMMButtonDblClk Type 5062
Messages.TWMMButtonDown Type 5063
Messages.TWMMButtonUp Type 5063
Messages.TWMMDIGetActive Type 5063
Messages.TWMMDIIconArrange Type 5064
Messages.TWMMDIRefreshMenu Type 5064
Messages.TWMMouseMove Type 5064
Messages.TWMNCDestroy Type 5065
Messages.TWMNCLButtonDblClk Type 5065
Messages.TWMNCLButtonDown Type 5065
Messages.TWMNCLButtonUp Type 5066
Messages.TWMNCMButtonDblClk Type 5066
Messages.TWMNCMButtonDown Type 5066
Messages.TWMNCMButtonUp Type 5067

Messages.TWMNCMouseMove Type 5067
Messages.TWMNCRButtonDblClk Type 5067
Messages.TWMNCRButtonDown Type 5068
Messages.TWMNCRButtonUp Type 5068
Messages.TWMPaintIcon Type 5068
Messages.TWMPaste Type 5069
Messages.TWMPrintClient Type 5069
Messages.TWMQueryDragIcon Type 5069
Messages.TWMQueryNewPalette Type 5070
Messages.TWMQueryOpen Type 5070
Messages.TWMQueryUIState Type 5070
Messages.TWMQueueSync Type 5071
Messages.TWMRButtonDblClk Type 5071
Messages.TWMRButtonDown Type 5071
Messages.TWMRButtonUp Type 5072
Messages.TWMRenderAllFormats Type 5072
Messages.TWMStyleChanged Type 5072
Messages.TWMStyleChanging Type 5073
Messages.TWMSysChar Type 5073
Messages.TWMSysColorChange Type 5073
Messages.TWMSysKeyDown Type 5074
Messages.TWMSysKeyUp Type 5074
Messages.TWMTimeChange Type 5074
Messages.TWMUndo Type 5075
Messages.TWMUpdateUIState Type 5075
Messages.TWMVKeyToItem Type 5075
Messages.TWMVScroll Type 5076
Messages.TWMWindowPosChanged Type 5076
Messages.TWMWindowPosChanging Type 5076

Constants **5077**

Messages.BM_CLICK Constant 5084
Messages.BM_GETCHECK Constant 5085
Messages.BM_GETIMAGE Constant 5085
Messages.BM_GETSTATE Constant 5085
Messages.BM_SETCHECK Constant 5086
Messages.BM_SETIMAGE Constant 5086
Messages.BM_SETSTATE Constant 5086
Messages.BM_SETSTYLE Constant 5087
Messages.BN_CLICKED Constant 5087

Messages.BN_DBLCLK Constant 5087
Messages.BN_DISABLE Constant 5088
Messages.BN_DOUBLECLICKED Constant 5088
Messages.BN_HILITE Constant 5088
Messages.BN_KILLFOCUS Constant 5089
Messages.BN_PAINT Constant 5089
Messages.BN_PUSHED Constant 5089
Messages.BN_SETFOCUS Constant 5090
Messages.BN_UNHILITE Constant 5090
Messages.BN_UNPUSHED Constant 5090
Messages.CB_ADDSTRING Constant 5091
Messages.CB_DELETESTRING Constant 5091
Messages.CB_DIR Constant 5091
Messages.CB_FINDSTRING Constant 5092
Messages.CB_FINDSTRINGEXACT Constant 5092
Messages.CB_GETCOUNT Constant 5092
Messages.CB_GETCURSEL Constant 5093
Messages.CB_GETDROPPEDCONTROLRECT Constant 5093
Messages.CB_GETDROPPEDSTATE Constant 5093
Messages.CB_GETDROPPEDWIDTH Constant 5094
Messages.CB_GETEDITSEL Constant 5094
Messages.CB_GETEXTENDEDUI Constant 5094
Messages.CB_GETHORIZONTALEXTENT Constant 5095
Messages.CB_GETITEMDATA Constant 5095
Messages.CB_GETITEMHEIGHT Constant 5095
Messages.CB_GETLBTEXT Constant 5096
Messages.CB_GETLBTEXTLEN Constant 5096
Messages.CB_GETLOCALE Constant 5096
Messages.CB_GETTOPINDEX Constant 5097
Messages.CB_INITSTORAGE Constant 5097
Messages.CB_INSERTSTRING Constant 5097
Messages.CB_LIMITTEXT Constant 5098
Messages.CB_MSGMAX Constant 5098
Messages.CB_RESETCONTENT Constant 5098
Messages.CB_SELECTSTRING Constant 5099
Messages.CB_SETCURSEL Constant 5099
Messages.CB_SETDROPPEDWIDTH Constant 5099
Messages.CB_SETEDITSEL Constant 5100
Messages.CB_SETEXTENDEDUI Constant 5100

Messages.CB_SETHORIZONTALEXTENT Constant 5100
Messages.CB_SETITEMDATA Constant 5101
Messages.CB_SETITEMHEIGHT Constant 5101
Messages.CB_SETLOCALE Constant 5101
Messages.CB_SETTOPINDEX Constant 5102
Messages.CB_SHOWDROPDOWN Constant 5102
Messages.CBN_CLOSEUP Constant 5102
Messages.CBN_DBLCLK Constant 5103
Messages.CBN_DROPDOWN Constant 5103
Messages.CBN_EDITCHANGE Constant 5103
Messages.CBN_EDITUPDATE Constant 5104
Messages.CBN_ERRSPACE Constant 5104
Messages.CBN_KILLFOCUS Constant 5104
Messages.CBN_SELCHANGE Constant 5105
Messages.CBN_SELENDCANCEL Constant 5105
Messages.CBN_SELENDOK Constant 5105
Messages.CBN_SETFOCUS Constant 5106
Messages.DM_GETDEFID Constant 5106
Messages.DM_REPOSITION Constant 5106
Messages.DM_SETDEFID Constant 5107
Messages.EM_CANUNDO Constant 5107
Messages.EM_CHARFROMPOS Constant 5107
Messages.EM_EMPTYUNDOBUFFER Constant 5108
Messages.EM_FMTLINES Constant 5108
Messages.EM_GETFIRSTVISIBLELINE Constant 5108
Messages.EM_GETHANDLE Constant 5109
Messages.EM_GETIMESTATUS Constant 5109
Messages.EM_GETLIMITTEXT Constant 5109
Messages.EM_GETLINE Constant 5110
Messages.EM_GETLINECOUNT Constant 5110
Messages.EM_GETMARGINS Constant 5110
Messages.EM_GETMODIFY Constant 5111
Messages.EM_GETPASSWORDCHAR Constant 5111
Messages.EM_GETRECT Constant 5111
Messages.EM_GETSEL Constant 5112
Messages.EM_GETTHUMB Constant 5112
Messages.EM_GETWORDBREAKPROC Constant 5112
Messages.EM_LIMITTEXT Constant 5113
Messages.EM_LINEFROMCHAR Constant 5113

Messages.EM_LINEINDEX Constant 5113
Messages.EM_LINELENGTH Constant 5114
Messages.EM_LINESCROLL Constant 5114
Messages.EM_POSFROMCHAR Constant 5114
Messages.EM_REPLACESEL Constant 5115
Messages.EM_SCROLL Constant 5115
Messages.EM_SCROLLCARET Constant 5115
Messages.EM_SETHANDLE Constant 5116
Messages.EM_SETIMESTATUS Constant 5116
Messages.EM_SETLIMITTEXT Constant 5116
Messages.EM_SETMARGINS Constant 5117
Messages.EM_SETMODIFY Constant 5117
Messages.EM_SETPASSWORDCHAR Constant 5117
Messages.EM_SETREADONLY Constant 5118
Messages.EM_SETRECT Constant 5118
Messages.EM_SETRECTNP Constant 5118
Messages.EM_SETSEL Constant 5119
Messages.EM_SETTABSTOPS Constant 5119
Messages.EM_SETWORDBREAKPROC Constant 5119
Messages.EM_UNDO Constant 5120
Messages.EN_CHANGE Constant 5120
Messages.EN_ERRSPACE Constant 5120
Messages.EN_HSCROLL Constant 5121
Messages.EN_KILLFOCUS Constant 5121
Messages.EN_MAXTEXT Constant 5121
Messages.EN_SETFOCUS Constant 5122
Messages.EN_UPDATE Constant 5122
Messages.EN_VSCROLL Constant 5122
Messages.LB_ADDFILE Constant 5123
Messages.LB_ADDSTRING Constant 5123
Messages.LB_DELETESTRING Constant 5123
Messages.LB_DIR Constant 5124
Messages.LB_FINDSTRING Constant 5124
Messages.LB_FINDSTRINGEXACT Constant 5124
Messages.LB_GETANCHORINDEX Constant 5125
Messages.LB_GETCARETINDEX Constant 5125
Messages.LB_GETCOUNT Constant 5125
Messages.LB_GETCURSEL Constant 5126
Messages.LB_GETHORIZONTALEXTENT Constant 5126

Messages.LB_GETITEMDATA Constant 5126
Messages.LB_GETITEMHEIGHT Constant 5127
Messages.LB_GETITEMRECT Constant 5127
Messages.LB_GETLOCALE Constant 5127
Messages.LB_GETSEL Constant 5128
Messages.LB_GETSELCOUNT Constant 5128
Messages.LB_GETSELITEMS Constant 5128
Messages.LB_GETTEXT Constant 5129
Messages.LB_GETTEXTLEN Constant 5129
Messages.LB_GETTOPINDEX Constant 5129
Messages.LB_INITSTORAGE Constant 5130
Messages.LB_INSERTSTRING Constant 5130
Messages.LB_ITEMFROMPOINT Constant 5130
Messages.LB_MSGMAX Constant 5131
Messages.LB_RESETCONTENT Constant 5131
Messages.LB_SELECTSTRING Constant 5131
Messages.LB_SELITEMRANGE Constant 5132
Messages.LB_SELITEMRANGEEX Constant 5132
Messages.LB_SETANCHORINDEX Constant 5132
Messages.LB_SETCARETINDEX Constant 5133
Messages.LB_SETCOLUMNWIDTH Constant 5133
Messages.LB_SETCOUNT Constant 5133
Messages.LB_SETCURSEL Constant 5134
Messages.LB_SETHORIZONTALEXTENT Constant 5134
Messages.LB_SETITEMDATA Constant 5134
Messages.LB_SETITEMHEIGHT Constant 5135
Messages.LB_SETLOCALE Constant 5135
Messages.LB_SETSEL Constant 5135
Messages.LB_SETTABSTOPS Constant 5136
Messages.LB_SETTOPINDEX Constant 5136
Messages.LBN_DBLCLK Constant 5136
Messages.LBN_ERRSPACE Constant 5137
Messages.LBN_KILLFOCUS Constant 5137
Messages.LBN_SELCANCEL Constant 5137
Messages.LBN_SELCHANGE Constant 5138
Messages.LBN_SETFOCUS Constant 5138
Messages.PSM_PAGEINFO Constant 5138
Messages.PSM_SHEETINFO Constant 5139
Messages.SBM_ENABLE_ARROWS Constant 5139

Messages.SBM_GETPOS Constant 5139
Messages.SBM_GETRANGE Constant 5140
Messages.SBM_GETSCROLLBARINFO Constant 5140
Messages.SBM_GETSCROLLINFO Constant 5140
Messages.SBM_SETPOS Constant 5141
Messages.SBM_SETRANGE Constant 5141
Messages.SBM_SETRANGEREDRAW Constant 5141
Messages.SBM_SETSCROLLINFO Constant 5142
Messages.WM_ACTIVATE Constant 5142
Messages.WM_ACTIVATEAPP Constant 5142
Messages.WM_APP Constant 5143
Messages.WM_APPCOMMAND Constant 5143
Messages.WM_ASKCBFORMATNAME Constant 5143
Messages.WM_CANCELJOURNAL Constant 5144
Messages.WM_CANCELMODE Constant 5144
Messages.WM_CAPTURECHANGED Constant 5144
Messages.WM_CHANGECBCHAIN Constant 5145
Messages.WM_CHANGEUISTATE Constant 5145
Messages.WM_CHAR Constant 5145
Messages.WM_CHARTOITEM Constant 5146
Messages.WM_CHILDACTIVATE Constant 5146
Messages.WM_CLEAR Constant 5146
Messages.WM_CLOSE Constant 5147
Messages.WM_COALESCE_FIRST Constant 5147
Messages.WM_COALESCE_LAST Constant 5147
Messages.WM_COMMAND Constant 5148
Messages.WM_COMMNOTIFY Constant 5148
Messages.WM_COMPACTING Constant 5148
Messages.WM_COMPAREITEM Constant 5149
Messages.WM_CONTEXTMENU Constant 5149
Messages.WM_COPY Constant 5149
Messages.WM_COPYDATA Constant 5150
Messages.WM_CREATE Constant 5150
Messages.WM_CTLCOLOR Constant 5150
Messages.WM_CTLCOLORBTN Constant 5151
Messages.WM_CTLCOLORDLG Constant 5151
Messages.WM_CTLCOLOREDIT Constant 5151
Messages.WM_CTLCOLORLISTBOX Constant 5152
Messages.WM_CTLCOLORMSGBOX Constant 5152

Messages.WM_CTLCOLORSCROLLBAR Constant 5152
Messages.WM_CTLCOLORSTATIC Constant 5153
Messages.WM_CUT Constant 5153
Messages.WM_DDE_ACK Constant 5153
Messages.WM_DDE_ADVISE Constant 5154
Messages.WM_DDE_DATA Constant 5154
Messages.WM_DDE_EXECUTE Constant 5154
Messages.WM_DDE_FIRST Constant 5155
Messages.WM_DDE_INITIATE Constant 5155
Messages.WM_DDE_LAST Constant 5155
Messages.WM_DDE_POKE Constant 5156
Messages.WM_DDE_REQUEST Constant 5156
Messages.WM_DDE_TERMINATE Constant 5156
Messages.WM_DDE_UNADVISE Constant 5157
Messages.WM_DEADCHAR Constant 5157
Messages.WM_DELETEITEM Constant 5157
Messages.WM_DESTROY Constant 5158
Messages.WM_DESTROYCLIPBOARD Constant 5158
Messages.WM_DEVICECHANGE Constant 5158
Messages.WM_DEVMODECHANGE Constant 5159
Messages.WM_DISPLAYCHANGE Constant 5159
Messages.WM_DRAWCLIPBOARD Constant 5159
Messages.WM_DRAWITEM Constant 5160
Messages.WM_DROPFILES Constant 5160
Messages.WM_DWMCOLORIZATIONCOLORCHANGED Constant 5160
Messages.WM_DWMCOMPOSITIONCHANGED Constant 5161
Messages.WM_DWMNCRENDERINGCHANGED Constant 5161
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE Constant 5161
Messages.WM_ENABLE Constant 5162
Messages.WM_ENDSESSION Constant 5162
Messages.WM_ENTERIDLE Constant 5162
Messages.WM_ENTERMENULOOP Constant 5163
Messages.WM_ENTERSIZEMOVE Constant 5163
Messages.WM_ERASEBKGND Constant 5163
Messages.WM_EXITMENULOOP Constant 5164
Messages.WM_EXITSIZEMOVE Constant 5164
Messages.WM_FONTCHANGE Constant 5164
Messages.WM_GETDLGCODE Constant 5165
Messages.WM_GETFONT Constant 5165

Messages.WM_GETHOTKEY Constant 5165
Messages.WM_GETICON Constant 5166
Messages.WM_GETMINMAXINFO Constant 5166
Messages.WM_GETOBJECT Constant 5166
Messages.WM_GETTEXT Constant 5167
Messages.WM_GETTEXTLENGTH Constant 5167
Messages.WM_HANDHELDFIRST Constant 5167
Messages.WM_HANDHELDLAST Constant 5168
Messages.WM_HELP Constant 5168
Messages.WM_HOTKEY Constant 5168
Messages.WM_HSCROLL Constant 5169
Messages.WM_HSCROLLCLIPBOARD Constant 5169
Messages.WM_ICONERASEBKGND Constant 5169
Messages.WM_IME_CHAR Constant 5170
Messages.WM_IME_COMPOSITION Constant 5170
Messages.WM_IME_COMPOSITIONFULL Constant 5170
Messages.WM_IME_CONTROL Constant 5171
Messages.WM_IME_ENDCOMPOSITION Constant 5171
Messages.WM_IME_KEYDOWN Constant 5171
Messages.WM_IME_KEYLAST Constant 5172
Messages.WM_IME_KEYUP Constant 5172
Messages.WM_IME_NOTIFY Constant 5172
Messages.WM_IME_REQUEST Constant 5173
Messages.WM_IME_SELECT Constant 5173
Messages.WM_IME_SETCONTEXT Constant 5173
Messages.WM_IME_STARTCOMPOSITION Constant 5174
Messages.WM_INITDIALOG Constant 5174
Messages.WM_INITMENU Constant 5174
Messages.WM_INITMENUPOPUP Constant 5175
Messages.WM_INPUT Constant 5175
Messages.WM_INPUTLANGCHANGE Constant 5175
Messages.WM_INPUTLANGCHANGEREQUEST Constant 5176
Messages.WM_KEYDOWN Constant 5176
Messages.WM_KEYFIRST Constant 5176
Messages.WM_KEYLAST Constant 5177
Messages.WM_KEYUP Constant 5177
Messages.WM_KILLFOCUS Constant 5177
Messages.WM_LBUTTONDBLCLK Constant 5178
Messages.WM_LBUTTONDOWN Constant 5178

Messages.WM_LBUTTONUP Constant 5178
Messages.WM_MBUTTONDBLCLK Constant 5179
Messages.WM_MBUTTONDOWN Constant 5179
Messages.WM_MBUTTONUP Constant 5179
Messages.WM_MDIACTIVATE Constant 5180
Messages.WM_MDICASCADE Constant 5180
Messages.WM_MDICREATE Constant 5180
Messages.WM_MDIDESTROY Constant 5181
Messages.WM_MDIGETACTIVE Constant 5181
Messages.WM_MDIICONARRANGE Constant 5181
Messages.WM_MDIMAXIMIZE Constant 5182
Messages.WM_MDINEXT Constant 5182
Messages.WM_MDIREFRESHMENU Constant 5182
Messages.WM_MDIRESTORE Constant 5183
Messages.WM_MDISETMENU Constant 5183
Messages.WM_MDITILE Constant 5183
Messages.WM_MEASUREITEM Constant 5184
Messages.WM_MENUCHAR Constant 5184
Messages.WM_MENUCOMMAND Constant 5184
Messages.WM_MENUDRAG Constant 5185
Messages.WM_MENUGETOBJECT Constant 5185
Messages.WM_MENURBUTTONUP Constant 5185
Messages.WM_MENUSELECT Constant 5186
Messages.WM_MOUSEACTIVATE Constant 5186
Messages.WM_MOUSEFIRST Constant 5186
Messages.WM_MOUSEHOVER Constant 5187
Messages.WM_MOUSELAST Constant 5187
Messages.WM_MOUSELEAVE Constant 5187
Messages.WM_MOUSEMOVE Constant 5188
Messages.WM_MOUSEWHEEL Constant 5188
Messages.WM_MOVE Constant 5188
Messages.WM_MOVING Constant 5189
Messages.WM_NCACTIVATE Constant 5189
Messages.WM_NCCALCSIZE Constant 5189
Messages.WM_NCCREATE Constant 5190
Messages.WM_NCDESTROY Constant 5190
Messages.WM_NCHITTEST Constant 5190
Messages.WM_NCLBUTTONDBLCLK Constant 5191
Messages.WM_NCLBUTTONDOWN Constant 5191

Messages.WM_NCLBUTTONUP Constant 5191
Messages.WM_NCMBUTTONDBLCLK Constant 5192
Messages.WM_NCMBUTTONDOWN Constant 5192
Messages.WM_NCMBUTTONUP Constant 5192
Messages.WM_NCMOUSEHOVER Constant 5193
Messages.WM_NCMOUSELEAVE Constant 5193
Messages.WM_NCMOUSEMOVE Constant 5193
Messages.WM_NCPAINT Constant 5194
Messages.WM_NCRBUTTONDBLCLK Constant 5194
Messages.WM_NCRBUTTONDOWN Constant 5194
Messages.WM_NCRBUTTONUP Constant 5195
Messages.WM_NCXBUTTONDBLCLK Constant 5195
Messages.WM_NCXBUTTONDOWN Constant 5195
Messages.WM_NCXBUTTONUP Constant 5196
Messages.WM_NEXTDLGCTL Constant 5196
Messages.WM_NEXTMENU Constant 5196
Messages.WM_NOTIFY Constant 5197
Messages.WM_NOTIFYFORMAT Constant 5197
Messages.WM_NULL Constant 5197
Messages.WM_PAINT Constant 5198
Messages.WM_PAINTCLIPBOARD Constant 5198
Messages.WM_PAINTICON Constant 5198
Messages.WM_PALETTECHANGED Constant 5199
Messages.WM_PALETTEISCHANGING Constant 5199
Messages.WM_PARENTNOTIFY Constant 5199
Messages.WM_PASTE Constant 5200
Messages.WM_PENWINFIRST Constant 5200
Messages.WM_PENWINLAST Constant 5200
Messages.WM_POWER Constant 5201
Messages.WM_POWERBROADCAST Constant 5201
Messages.WM_PRINT Constant 5201
Messages.WM_PRINTCLIENT Constant 5202
Messages.WM_QUERYDRAGICON Constant 5202
Messages.WM_QUERYENDSESSION Constant 5202
Messages.WM_QUERYNEWPALETTE Constant 5203
Messages.WM_QUERYOPEN Constant 5203
Messages.WM_QUERYUISTATE Constant 5203
Messages.WM_QUEUESYNC Constant 5204
Messages.WM_QUIT Constant 5204

Messages.WM_RBUTTONDBLCLK Constant 5204
Messages.WM_RBUTTONDOWN Constant 5205
Messages.WM_RBUTTONUP Constant 5205
Messages.WM_RENDERALLFORMATS Constant 5205
Messages.WM_RENDERFORMAT Constant 5206
Messages.WM_SETCURSOR Constant 5206
Messages.WM_SETFOCUS Constant 5206
Messages.WM_SETFONT Constant 5207
Messages.WM_SETHOTKEY Constant 5207
Messages.WM_SETICON Constant 5207
Messages.WM_SETREDRAW Constant 5208
Messages.WM_SETTEXT Constant 5208
Messages.WM_SETTINGCHANGE Constant 5208
Messages.WM_SHOWWINDOW Constant 5209
Messages.WM_SIZE Constant 5209
Messages.WM_SIZECLIPBOARD Constant 5209
Messages.WM_SIZING Constant 5210
Messages.WM_SPOOLERSTATUS Constant 5210
Messages.WM_STYLECHANGED Constant 5210
Messages.WM_STYLECHANGING Constant 5211
Messages.WM_SYSCHAR Constant 5211
Messages.WM_SYSCOLORCHANGE Constant 5211
Messages.WM_SYSCOMMAND Constant 5212
Messages.WM_SYSDEADCHAR Constant 5212
Messages.WM_SYSKEYDOWN Constant 5212
Messages.WM_SYSKEYUP Constant 5213
Messages.WM_SYSTEMERROR Constant 5213
Messages.WM_TABLET_FIRST Constant 5213
Messages.WM_TABLET_LAST Constant 5214
Messages.WM_TCARD Constant 5214
Messages.WM_THEMECHANGED Constant 5214
Messages.WM_TIMECHANGE Constant 5215
Messages.WM_TIMER Constant 5215
Messages.WM_UNDO Constant 5215
Messages.WM_UNICHAR Constant 5216
Messages.WM_UNINITMENUPOPUP Constant 5216
Messages.WM_UPDATEUISTATE Constant 5216
Messages.WM_USER Constant 5217
Messages.WM_USERCHANGED Constant 5217

Messages.WM_VKEYTOITEM Constant 5217
Messages.WM_VSCROLL Constant 5218
Messages.WM_VSCROLLCLIPBOARD Constant 5218
Messages.WM_WINDOWPOSCHANGED Constant 5218
Messages.WM_WINDOWPOSCHANGING Constant 5219
Messages.WM_WININICHANGE Constant 5219
Messages.WM_WTSSESSION_CHANGE Constant 5219

Midas Namespace 5221

Interfaces 5222
IAppServer Interface 5222
IAppServer Members 5223
IAppServer Methods 5223
IAppServerDisp Interface 5229
IAppServerDisp Members 5229
IAppServerDisp Methods 5229

Constants 5232
Midas.IID_IAppServer Constant 5232
Midas.LIBID_Midas Constant 5232

MidasCon Namespace 5233

Classes 5234
TMidasConnection Class 5234
TMidasConnection Members 5234
TMidasConnection.Create Constructor 5235
TMidasConnection Methods 5236
TMidasConnection Properties 5237
TRemoteServer Class 5239

Structs, Records, Enums 5241
MidasCon.TConnectType Enumeration 5241

MidItems Namespace 5243

Classes 5248
TApplyUpdatesButton Class 5250
TApplyUpdatesButton Members 5251
TApplyUpdatesButton.Create Constructor 5251

TApplyUpdatesButton Methods 5252
TApplyUpdatesButton Properties 5253
TCaptionAttributes Class 5256
TCaptionAttributes Members 5256
TCaptionAttributes.Create Constructor 5257
TCaptionAttributes Properties 5257
TCustomDataForm Class 5260
TCustomDataForm Members 5260
TCustomDataForm Methods 5261
TCustomDataGrid Class 5263
TCustomDataGrid Members 5264
TCustomDataGrid.Create Constructor 5266
TCustomDataGrid.Destroy Destructor 5266
TCustomDataGrid Methods 5266
TCustomDataGrid Properties 5269
TCustomDataNavigator Class 5271
TCustomDataNavigator Members 5272
TCustomDataNavigator Methods 5273
TCustomDataNavigator Properties 5275
TCustomFieldGroup Class 5276
TCustomFieldGroup Members 5277
TCustomFieldGroup.Create Constructor 5279
TCustomFieldGroup.Destroy Destructor 5279
TCustomFieldGroup Methods 5280
TCustomFieldGroup Properties 5281
TCustomLayoutGroup Class 5282
TCustomLayoutGroup Members 5283
TCustomLayoutGroup.Create Constructor 5284
TCustomLayoutGroup.Destroy Destructor 5284
TCustomLayoutGroup Methods 5284
TCustomLayoutGroup Properties 5286
TCustomQueryButtons Class 5288
TCustomQueryButtons Members 5288
TCustomQueryButtons Methods 5289
TCustomQueryButtons Properties 5291
TCustomQueryForm Class 5293
TCustomQueryForm Members 5294
TCustomQueryForm.Create Constructor 5295
TCustomQueryForm Methods 5295

TCustomQueryForm Properties 5297
TDataForm Class 5298
TDataForm Members 5299
TDataForm Properties 5300
TDataGrid Class 5302
TDataGrid Members 5303
TDataGrid Properties 5305
TDataNavigator Class 5307
TDataNavigator Members 5307
TDataNavigator Properties 5309
TDataSetButton Class 5311
TDataSetButton Members 5312
TDataSetButton Fields 5313
TDataSetButton Methods 5313
TDataSetButton Properties 5315
TDeleteButton Class 5318
TDeleteButton Members 5319
TDeleteButton.Create Constructor 5320
TFieldGroup Class 5320
TFieldGroup Members 5321
TFieldGroup Methods 5323
TFieldGroup Properties 5324
TFieldRadioGroup Class 5326
TFieldRadioGroup Members 5326
TFieldRadioGroup Properties 5328
TFieldSelectOptions Class 5335
TFieldSelectOptions Members 5336
TFieldSelectOptions Properties 5338
TFieldStatus Class 5344
TFieldStatus Members 5345
TFieldStatus Properties 5346
TFieldText Class 5349
TFieldText Members 5350
TFieldText Properties 5351
TFieldTextArea Class 5355
TFieldTextArea Members 5356
TFieldTextArea Properties 5358
TFirstButton Class 5363
TFirstButton Members 5363

TFirstButton.Create Constructor 5365
TFormLayout Class 5365
TFormLayout Members 5366
TFormLayout.Create Constructor 5366
TFormLayout Methods 5367
TFormLayout Properties 5371
TGridAttributes Class 5373
TGridAttributes Members 5373
TGridAttributes.Create Constructor 5374
TGridAttributes Methods 5374
TGridAttributes Properties 5375
TGridRowAttributes Class 5379
TGridRowAttributes Members 5379
TGridRowAttributes.Create Constructor 5380
TGridRowAttributes Methods 5380
TGridRowAttributes Properties 5381
TInsertButton Class 5385
TInsertButton Members 5385
TInsertButton.Create Constructor 5386
TLastButton Class 5387
TLastButton Members 5387
TLastButton.Create Constructor 5388
TLayoutGroup Class 5389
TLayoutGroup Members 5389
TLayoutGroup Properties 5390
TNextButton Class 5392
TNextButton Members 5393
TNextButton.Create Constructor 5394
TNextPageButton Class 5394
TNextPageButton Members 5395
TNextPageButton.Create Constructor 5396
TPostButton Class 5396
TPostButton Members 5397
TPostButton.Create Constructor 5398
TPriorButton Class 5399
TPriorButton Members 5399
TPriorButton.Create Constructor 5400
TPriorPageButton Class 5401
TPriorPageButton Members 5401

TPriorPageButton.Create Constructor 5403
TQueryButton Class 5403
TQueryButton Members 5404
TQueryButton.Create Constructor 5404
TQueryButton Fields 5405
TQueryButton Methods 5405
TQueryButton Properties 5409
TQueryButtons Class 5412
TQueryButtons Members 5412
TQueryButtons Properties 5413
TQueryFieldGroup Class 5415
TQueryFieldGroup Members 5416
TQueryFieldGroup Methods 5418
TQueryFieldGroup Properties 5419
TQueryForm Class 5421
TQueryForm Members 5422
TQueryForm Properties 5423
TQueryRadioGroup Class 5425
TQueryRadioGroup Members 5426
TQueryRadioGroup Methods 5428
TQueryRadioGroup Properties 5429
TQuerySelectOptions Class 5437
TQuerySelectOptions Members 5438
TQuerySelectOptions Methods 5440
TQuerySelectOptions Properties 5442
TQueryText Class 5449
TQueryText Members 5450
TQueryText.Create Constructor 5451
TQueryText Methods 5452
TQueryText Properties 5453
TQueryTextArea Class 5458
TQueryTextArea Members 5459
TQueryTextArea.Create Constructor 5461
TQueryTextArea.Destroy Destructor 5461
TQueryTextArea Methods 5462
TQueryTextArea Properties 5464
TResetQueryButton Class 5469
TResetQueryButton Members 5470
TResetQueryButton.Create Constructor 5471

TResetQueryButton Methods 5471
TSelectOptionsColumn Class 5472
TSelectOptionsColumn Members 5472
TSelectOptionsColumn Methods 5474
TSelectOptionsColumn Properties 5475
TStatusColumn Class 5481
TStatusColumn Members 5482
TStatusColumn Methods 5483
TStatusColumn Properties 5483
TSubmitQueryButton Class 5486
TSubmitQueryButton Members 5486
TSubmitQueryButton.Create Constructor 5487
TSubmitQueryButton Methods 5487
TTextAreaColumn Class 5488
TTextAreaColumn Members 5489
TTextAreaColumn Methods 5490
TTextAreaColumn Properties 5491
TTextColumn Class 5495
TTextColumn Members 5496
TTextColumn Methods 5498
TTextColumn Properties 5498
TUndoButton Class 5502
TUndoButton Members 5503
TUndoButton.Create Constructor 5504
TWebButton Class 5504
TWebButton Members 5505
TWebButton.Create Constructor 5506
TWebButton Methods 5506
TWebButton Properties 5508
TWebControlGroup Class 5510
TWebControlGroup Members 5511
TWebControlGroup.Create Constructor 5512
TWebControlGroup.Destroy Destructor 5512
TWebControlGroup Methods 5512
TWebControlGroup Properties 5518
TWebDataDisplay Class 5519
TWebDataDisplay Members 5520
TWebDataDisplay.Create Constructor 5521
TWebDataDisplay.Destroy Destructor 5522

TWebDataDisplay Methods 5522
TWebDataDisplay Properties 5529
TWebDataInput Class 5532
TWebDataInput Members 5533
TWebDataInput Methods 5535
TWebDataInput Properties 5539
TWebDisplay Class 5540
TWebDisplay Members 5541
TWebDisplay.Create Constructor 5541
TWebDisplay Methods 5541
TWebDisplay Properties 5543
TWebDisplay Events 5543
TWebDisplayReferenceGroup Class 5544
TWebDisplayReferenceGroup Members 5544
TWebDisplayReferenceGroup.Create Constructor 5545
TWebDisplayReferenceGroup.Destroy Destructor 5546
TWebDisplayReferenceGroup Methods 5546
TWebDisplayReferenceGroup Properties 5547
TWebForm Class 5548
TWebForm Members 5549
TWebForm.Create Constructor 5549
TWebForm.Destroy Destructor 5550
TWebForm Methods 5550
TWebForm Properties 5556
TWebListInput Class 5558
TWebListInput Members 5559
TWebListInput.Create Constructor 5561
TWebListInput.Destroy Destructor 5561
TWebListInput Methods 5562
TWebListInput Properties 5564
TWebRadioGroupInput Class 5567
TWebRadioGroupInput Members 5568
TWebRadioGroupInput.Create Constructor 5570
TWebRadioGroupInput Methods 5570
TWebRadioGroupInput Properties 5572
TWebSelectOptionsInput Class 5573
TWebSelectOptionsInput Members 5574
TWebSelectOptionsInput.Create Constructor 5576
TWebSelectOptionsInput Methods 5576

TWebSelectOptionsInput Properties 5577
TWebStatus Class 5578
TWebStatus Members 5579
TWebStatus.Create Constructor 5580
TWebStatus Methods 5580
TWebStatus Properties 5582
TWebTextAreaInput Class 5583
TWebTextAreaInput Members 5584
TWebTextAreaInput.Create Constructor 5586
TWebTextAreaInput Methods 5586
TWebTextAreaInput Properties 5587
TWebTextInput Class 5589
TWebTextInput Members 5590
TWebTextInput.Create Constructor 5592
TWebTextInput Methods 5592
TWebTextInput Properties 5593
TXMLButton Class 5595
TXMLButton Members 5596
TXMLButton.Create Constructor 5596
TXMLButton.Destroy Destructor 5597
TXMLButton Methods 5597
TXMLButton Properties 5599
TXMLData Class 5601
TXMLData Members 5601
TXMLData.Create Constructor 5602
TXMLData Methods 5602
TXMLData Properties 5604
TXMLData Events 5604
TXMLDataParent Class 5605
TXMLDataParent Members 5605
TXMLDataParent Methods 5606
TXMLDataParent Properties 5607
TXMLDataSet Class 5608
TXMLDataSet Members 5608
TXMLDataSet.Create Constructor 5609
TXMLDataSet.Destroy Destructor 5610
TXMLDataSet Methods 5610
TXMLDataSet Properties 5614
TXMLDataSetParent Class 5616

TXMLDataSetParent Members 5616
TXMLDataSetParent Methods 5617
TXMLDataSetParent Properties 5619
TXMLDisplayGroup Class 5620
TXMLDisplayGroup Members 5621
TXMLDisplayGroup.Create Constructor 5622
TXMLDisplayGroup.Destroy Destructor 5622
TXMLDisplayGroup Methods 5623
TXMLDisplayGroup Properties 5629
TXMLDisplayParent Class 5632
TXMLDisplayParent Members 5632
TXMLDisplayParent.Create Constructor 5633
TXMLDisplayParent Methods 5633
TXMLDisplayParent Properties 5634
TXMLDisplayReferenceButton Class 5635
TXMLDisplayReferenceButton Members 5636
TXMLDisplayReferenceButton.Create Constructor 5637
TXMLDisplayReferenceButton.Destroy Destructor 5637
TXMLDisplayReferenceButton Methods 5637
TXMLDisplayReferenceButton Properties 5639
TXMLDisplayReferenceGroup Class 5640
TXMLDisplayReferenceGroup Members 5641
TXMLDisplayReferenceGroup Methods 5642
TXMLDisplayReferenceGroup Properties 5643

Interfaces 5644

IDataSetComponent Interface 5644
IDataSetComponent Members 5645
IDataSetComponent Methods 5645
IDataSetComponent Properties 5646
IDataSetField Interface 5648
IDataSetField Members 5648
IDataSetField Methods 5648
IDataSetField Properties 5649
IDataSetFields Interface 5650
IDataSetFields Members 5650
IDataSetFields Methods 5650
IDataSetFields Properties 5652
IFormatColumn Interface 5652
IFormatColumn Members 5652

IFormatColumn Methods 5653
IHTMLField Interface 5653
IHTMLField Members 5654
IHTMLField Methods 5654
IHTMLField Properties 5654
IHTMLForm Interface 5655
IHTMLForm Members 5655
IHTMLForm Methods 5656
IHTMLForm Properties 5657
IInetXWebDataSet Interface 5657
IInetXWebDataSet Members 5658
IInetXWebDataSet Methods 5658
IQueryField Interface 5661
IQueryField Members 5661
IQueryField Methods 5662
IQueryField Properties 5663
IQueryFields Interface 5664
IQueryFields Members 5664
IQueryFields Methods 5664
IQueryForm Interface 5665
IRestoreDefaults Interface 5665
IRestoreDefaults Members 5666
IRestoreDefaults Methods 5666
IStatusField Interface 5666
IValidateField Interface 5667
IValidateField Members 5667
IValidateField Methods 5667
IValidateFields Interface 5668
IValidateFields Members 5668
IValidateFields Methods 5668
IValidateFields Properties 5669
IWebDisplay Interface 5670
IWebDisplay Members 5670
IWebDisplay Methods 5670
IWebDisplay Properties 5671
IWebDisplayReference Interface 5671
IWebDisplayReference Members 5671
IWebDisplayReference Methods 5672
IWebDisplayReference Properties 5672

IXMLDisplay Interface 5673
IXMLDisplay Members 5673
IXMLDisplay Methods 5673
IXMLDisplay Properties 5674

Functions 5676
MidItems.AddBoolAttrib Function 5676
MidItems.AddCustomAttrib Function 5676
MidItems.AddIntAttrib Function 5677
MidItems.AddQuotedAttrib Function 5677
MidItems.AddStringAttrib Function 5677
MidItems.CanAddClassHelper Function 5678
MidItems.CreateDefaultButtonClasses Function 5678
MidItems.DeclareSubmitForm Function 5678
MidItems.FindDispatcher Function 5679
MidItems.FindProducer Function 5679
MidItems.FormatColumnData Function 5679
MidItems.FormatColumnHeading Function 5680
MidItems.GetItemValuesFromDataSet Function 5680

Structs, Records, Enums 5681
MidItems.TCaptionPosition Enumeration 5681
MidItems.TFormMethod Enumeration 5682
MidItems.TLayoutState Enumeration 5682
MidItems.TTextAreaWrap Enumeration 5683

Types 5684
MidItems.IMidasWebDataSet Type 5684
MidItems.TWebButtonClass Type 5684
MidItems.TWebDataDisplayClass Type 5685
MidItems.TWebDataInputClass Type 5685
MidItems.TWebStatusClass Type 5685

Constants 5686
MidItems.FormMethodNames Constant 5686
MidItems.HTMLAlignValues Constant 5687
MidItems.HTMLVAlignValues Constant 5687
MidItems.sActionAttr Constant 5687
MidItems.sAlignAttr Constant 5688
MidItems.sBgColorAttr Constant 5688
MidItems.sBorderAttr Constant 5688

MidItems.sCellPaddingAttr Constant 5689
MidItems.sCellSpacingAttr Constant 5689
MidItems.sCheckedAttr Constant 5689
MidItems.sClassAttr Constant 5690
MidItems.sColsAttr Constant 5690
MidItems.sColSpanAttr Constant 5690
MidItems.sEncTypeAttr Constant 5691
MidItems.sImageHeightAttr Constant 5691
MidItems.sImageWidthAttr Constant 5691
MidItems.sMaxLengthAttr Constant 5692
MidItems.sMethodAttr Constant 5692
MidItems.sMultipleAttr Constant 5692
MidItems.sNameAttr Constant 5693
MidItems.sOnClickAttr Constant 5693
MidItems.sReadOnlyAttr Constant 5693
MidItems.sRowsAttr Constant 5694
MidItems.sSelectedAttr Constant 5694
MidItems.sSizeAttr Constant 5694
MidItems.sStyleAttr Constant 5695
MidItems.sTabIndexAttr Constant 5695
MidItems.sTypeAttr Constant 5695
MidItems.sVAlignAttr Constant 5696
MidItems.sValueAttr Constant 5696
MidItems.sWidthAttr Constant 5696
MidItems.sWrapAttr Constant 5697
MidItems.TextAreaWrap Constant 5697

MidProd Namespace 5699

Classes **5701**
TBaseInetXPageProducer Class 5701
TBaseInetXPageProducer Members 5702
TBaseInetXPageProducer.Create Constructor 5703
TBaseInetXPageProducer.Destroy Destructor 5704
TBaseInetXPageProducer Methods 5704
TBaseInetXPageProducer Properties 5715
TBaseInetXPageProducer Events 5717
TCustomInetXPageProducer Class 5720
TCustomInetXPageProducer Members 5720

TCustomInetXPageProducer.Create Constructor 5722
TCustomInetXPageProducer.Destroy Destructor 5723
TCustomInetXPageProducer Methods 5723
TCustomInetXPageProducer Properties 5725
TInetXPageElements Class 5727
TInetXPageElements Members 5728
TInetXPageElements Fields 5728
TInetXPageElements Methods 5729
TInetXPageProducer Class 5730
TInetXPageProducer Members 5730
TInetXPageProducer Properties 5733
TInetXPageProducer Events 5737
TMidasPageProducer Class 5740

Functions **5741**
MidProd.AddStdIncludes Function 5741
MidProd.PathInfoToRelativePath Function 5741

Structs, Records, Enums **5742**
MidProd.TStdInclude Enumeration 5742

Types **5743**
MidProd.TBaseMidasPageProducer Type 5743
MidProd.TCustomMidasPageProducer Type 5743
MidProd.TMidasPageElements Type 5744
MidProd.TStdIncludes Type 5744
MidProd.TXMLDataEvent Type 5744

Constants **5745**
MidProd.sXMLBroker Constant 5745
MidProd.sXMLReadyVar Constant 5745

MPlayer Namespace 5747

Classes **5749**
EMCIDeviceError Class 5749
TMediaPlayer Class 5749
TMediaPlayer Members 5753
TMediaPlayer.Create Constructor 5755
TMediaPlayer.Destroy Destructor 5755
TMediaPlayer Methods 5756
TMediaPlayer Properties 5775

TMediaPlayer Events 5810

Structs, Records, Enums **5819**

MPlayer.TMPBtnType Enumeration 5819

MPlayer.TMPButton Record 5820

MPlayer.TMPDevCaps Enumeration 5821

MPlayer.TMPDeviceTypes Enumeration 5821

MPlayer.TMPGlyph Enumeration 5823

MPlayer.TMPModes Enumeration 5823

MPlayer.TMPNotifyValues Enumeration 5824

MPlayer.TMPTimeFormats Enumeration 5825

Types **5827**

MPlayer.EMPNotify Type 5827

MPlayer.EMPPostNotify Type 5827

MPlayer.TButtonSet Type 5828

MPlayer.TMediaDevice Type 5828

MPlayer.TMPDevCapsSet Type 5828

Mtsobj Namespace 5831

Classes **5832**

TMtsAutoObject Class 5832

TMtsAutoObject Members 5832

TMtsAutoObject Methods 5833

TMtsAutoObject Properties 5838

MtsRdm Namespace 5839

Classes **5840**

TMtsDataModule Class 5840

TMtsDataModule Members 5841

TMtsDataModule.Create Constructor 5841

TMtsDataModule Methods 5842

TMtsDataModule Properties 5852

TMtsDataModule Events 5854

Mtx Namespace 5857

Interfaces **5859**

IEnumNames Interface 5859

IEnumNames Members 5859
IEnumNames Methods 5860
IGetContextProperties Interface 5861
IGetContextProperties Members 5861
IGetContextProperties Methods 5861
IObjectContext Interface 5862
IObjectContext Members 5862
IObjectContext Methods 5863
IObjectContextActivity Interface 5865
IObjectContextActivity Members 5865
IObjectContextActivity Methods 5865
IObjectControl Interface 5866
IObjectControl Members 5866
IObjectControl Methods 5866
ISecurityProperty Interface 5867
ISecurityProperty Members 5868
ISecurityProperty Methods 5868
ISharedProperty Interface 5869
ISharedProperty Members 5869
ISharedProperty Methods 5870
ISharedProperty Properties 5870
ISharedPropertyGroup Interface 5871
ISharedPropertyGroup Members 5871
ISharedPropertyGroup Methods 5871
ISharedPropertyGroup Properties 5873
ISharedPropertyGroupManager Interface 5873
ISharedPropertyGroupManager Members 5874
ISharedPropertyGroupManager Methods 5874
ISharedPropertyGroupManager Properties 5875
ITransactionContext Interface 5875
ITransactionContext Members 5876
ITransactionContext Methods 5876
ITransactionContextEx Interface 5877
ITransactionContextEx Members 5877
ITransactionContextEx Methods 5877

Functions **5879**

Mtx.CreateSharedPropertyGroup Function 5879
Mtx.CreateSharedPropertyGroupManager Function 5880
Mtx.CreateTransactionContext Function 5880

Mtx.CreateTransactionContextEx Function 5881
Mtx.GetObjectContext Function 5882
Mtx.SafeRef Function 5882

Constants **5884**
Mtx.CLASS_SharedPropertyGroupManager Constant 5884
Mtx.CLASS_TransactionContext Constant 5884
Mtx.CLASS_TransactionContextEx Constant 5885
Mtx.LockMethod Constant 5885
Mtx.LockSetGet Constant 5885
Mtx.mtsErrCtxAborted Constant 5886
Mtx.mtsErrCtxAborting Constant 5886
Mtx.mtsErrCtxActivityTimeout Constant 5886
Mtx.mtsErrCtxNoContext Constant 5887
Mtx.mtsErrCtxNoSecurity Constant 5887
Mtx.mtsErrCtxNotRegistered Constant 5887
Mtx.mtsErrCtxOldReference Constant 5888
Mtx.mtsErrCtxRoleNotFound Constant 5888
Mtx.mtsErrCtxTMNotAvailable Constant 5888
Mtx.mtsErrCtxWrongThread Constant 5889
Mtx.Process Constant 5889
Mtx.Standard Constant 5889

ObjAuto Namespace 5891

Interfaces **5893**
IMethodHandler Interface 5893
IMethodHandler Members 5893
IMethodHandler Methods 5893

Functions **5895**
ObjAuto.CreateMethodPointer Function 5895
ObjAuto.CreateMethodPointer Function 5895
ObjAuto.GetInvokeInstance Function 5896
ObjAuto.GetMethodInfo Function 5896
ObjAuto.GetMethods Function 5896
ObjAuto.ObjectInvoke Function 5897
ObjAuto.ReleaseMethodPointer Function 5897

Structs, Records, Enums **5898**
ObjAuto.TCallingConvention Enumeration 5898

ObjAuto.TMethodInfoHeader Record 5898
ObjAuto.TParameters Record 5899
ObjAuto.TParamInfo Record 5899
ObjAuto.TReturnInfo Record 5900

Types **5901**
ObjAuto.PMethodInfoHeader Type 5901
ObjAuto.PParameters Type 5901
ObjAuto.PParamInfo Type 5902
ObjAuto.PPointer Type 5902
ObjAuto.PReturnInfo Type 5902
ObjAuto.PWord Type 5903
ObjAuto.TDynamicInvokeEvent Type 5903
ObjAuto.TMethodInfoArray Type 5903
ObjAuto.TParamFlags Type 5904

Constants **5905**
ObjAuto.paEAX Constant 5905
ObjAuto.paECX Constant 5905
ObjAuto.paEDX Constant 5906
ObjAuto.paStack Constant 5906

ObjBrkr Namespace 5907

Classes **5908**
EBrokerException Class 5908
TServerCollection Class 5908
TServerCollection Members 5909
TServerCollection.Create Constructor 5909
TServerCollection Methods 5910
TServerCollection Properties 5911
TServerItem Class 5912
TServerItem Members 5912
TServerItem.Create Constructor 5913
TServerItem Methods 5913
TServerItem Properties 5914
TSimpleObjectBroker Class 5916
TSimpleObjectBroker Members 5917
TSimpleObjectBroker.Create Constructor 5918
TSimpleObjectBroker.Destroy Destructor 5918
TSimpleObjectBroker Methods 5918

TSimpleObjectBroker Properties 5922

ObjComAuto Namespace 5925

Classes **5926**

TObjectDispatch Class 5926

TObjectDispatch Members 5926

TObjectDispatch.Create Constructor 5927

TObjectDispatch.Destroy Destructor 5927

TObjectDispatch Methods 5927

TObjectDispatch Properties 5929

Functions **5931**

ObjComAuto.CreateMethodPointer Function 5931

ObjComAuto.CreateMethodPointer Function 5931

ObjComAuto.ReleaseMethodPointer Function 5932

Structs, Records, Enums **5933**

ObjComAuto.TDispatchInfo Record 5933

ObjComAuto.TDispatchKind Enumeration 5933

Types **5935**

ObjComAuto.TDispatchInfos Type 5935

OleAuto Namespace 5937

Classes **5939**

EOleError Class 5939

EOleException Class 5939

EOleException Members 5940

EOleException.Create Constructor 5940

EOleException Properties 5940

EOleSysError Class 5941

EOleSysError Members 5941

EOleSysError.Create Constructor 5942

EOleSysError Properties 5942

TAutoDispatch Class 5942

TAutoDispatch Members 5943

TAutoDispatch.Create Constructor 5943

TAutoDispatch Methods 5943

TAutoDispatch Properties 5945

TAutomation Class 5946

TAutomation Members 5946
TAutomation.Create Constructor 5947
TAutomation.Destroy Destructor 5947
TAutomation Methods 5947
TAutomation Properties 5948
TAutomation Events 5948
TAutoObject Class 5949
TAutoObject Members 5949
TAutoObject.Create Constructor 5950
TAutoObject.Destroy Destructor 5950
TAutoObject Methods 5951
TAutoObject Properties 5952
TRegistryClass Class 5953
TRegistryClass Members 5953
TRegistryClass.Create Constructor 5953
TRegistryClass.Destroy Destructor 5953
TRegistryClass Methods 5954

Functions **5955**
OleAuto.ClassIDToProgID Function 5955
OleAuto.ClassIDToString Function 5955
OleAuto.CreateOleObject Function 5956
OleAuto.DispInvoke Function 5956
OleAuto.DispInvokeError Function 5956
OleAuto.DllCanUnloadNow Function 5957
OleAuto.DllGetClassObject Function 5957
OleAuto.DllRegisterServer Function 5957
OleAuto.DllUnregisterServer Function 5958
OleAuto.GetActiveOleObject Function 5958
OleAuto.OleCheck Function 5958
OleAuto.OleError Function 5959
OleAuto.ProgIDToClassID Function 5959
OleAuto.StringToClassID Function 5959
OleAuto.VarFromInterface Function 5960
OleAuto.VarToAutoObject Function 5960
OleAuto.VarToInterface Function 5960

Structs, Records, Enums **5961**
OleAuto.TAutoClassInfo Record 5961
OleAuto.TAutoClassInstancing Enumeration 5961

OleAuto.TStartMode Enumeration 5962

Types **5963**

OleAuto.TAutoClass Type 5963

OleAuto.TLastReleaseEvent Type 5963

Variables **5964**

OleAuto.Automation Variable 5964

Constants **5965**

OleAuto.MaxDispArgs Constant 5965

OleCtnrs Namespace 5967

Classes **5969**

TOleContainer Class 5969

TOleContainer Members 5970

TOleContainer.Create Constructor 5972

TOleContainer.Destroy Destructor 5973

TOleContainer Methods 5973

TOleContainer Properties 5990

TOleContainer Events 6021

TOleForm Class 6050

TOleForm Members 6051

TOleForm.Create Constructor 6051

TOleForm.Destroy Destructor 6051

Interfaces **6053**

IVCLFrameForm Interface 6053

IVCLFrameForm Members 6053

IVCLFrameForm Methods 6053

Functions **6055**

OleCtnrs.DestroyMetaPict Function 6055

Structs, Records, Enums **6056**

OleCtnrs.TAutoActivate Enumeration 6056

OleCtnrs.TCreateInfo Record 6056

OleCtnrs.TCreateType Enumeration 6057

OleCtnrs.TObjectState Enumeration 6057

OleCtnrs.TSizeMode Enumeration 6058

OleCtnrs.TVerbInfo Record 6058

Types **6060**

OleCtnrs.TObjectMoveEvent Type 6060

Constants **6061**

OleCtnrs.ovDiscardUndoState Constant 6061
OleCtnrs.ovHide Constant 6061
OleCtnrs.ovInPlaceActivate Constant 6062
OleCtnrs.ovOpen Constant 6062
OleCtnrs.ovPrimary Constant 6062
OleCtnrs.ovShow Constant 6063
OleCtnrs.ovUIActivate Constant 6063

OleCtrls Namespace 6065

Classes **6067**

EOleCtrlError Class 6067
TEnumPropDesc Class 6067
TEnumPropDesc Members 6068
TEnumPropDesc.Create Constructor 6068
TEnumPropDesc.Destroy Destructor 6069
TEnumPropDesc Methods 6069
TEventDispatch Class 6070
TEventDispatch Members 6071
TEventDispatch.Create Constructor 6071
TEventDispatch Methods 6071
TEventDispatch Properties 6073
TOleControl Class 6074
TOleControl Members 6074
TOleControl.Create Constructor 6077
TOleControl.Destroy Destructor 6078
TOleControl Methods 6078
TOleControl Properties 6119
TOleControl Events 6121

Functions **6122**

OleCtrls.EmptyParam Function 6122
OleCtrls.FontToOleFont Function 6122
OleCtrls.OleFontToFont Function 6123

Structs, Records, Enums **6124**

OleCtrls.TControlData Record 6124
OleCtrls.TControlData2 Record 6125

OleCtrls.TEnumValue Record 6125

Types **6127**

OleCtrls.PControlData Type 6127

OleCtrls.PControlData2 Type 6127

OleCtrls.PEnumValueList Type 6128

OleCtrls.TEnumValueList Type 6128

OleCtrls.TOleEnum Type 6128

OleCtrls.TServiceQuery Type 6129

OleServer Namespace 6131

Classes **6132**

TOleServer Class 6132

TOleServer Members 6132

TOleServer.Create Constructor 6133

TOleServer.Destroy Destructor 6134

TOleServer Methods 6135

TOleServer Properties 6141

TServerEventDispatch Class 6144

TServerEventDispatch Members 6144

TServerEventDispatch.Create Constructor 6144

TServerEventDispatch Methods 6145

TServerEventDispatch Properties 6147

Structs, Records, Enums **6148**

OleServer.TConnectKind Enumeration 6148

OleServer.TServerData Record 6148

Types **6150**

OleServer.PServerData Type 6150

OleServer.TVariantArray Type 6150

OPConvert Namespace 6151

Interfaces **6152**

IOPConvert Interface 6152

IOPConvert Members 6153

IOPConvert Methods 6153

IOPConvert Properties 6160

Structs, Records, Enums **6162**

OPConvert.TSOAPConvertOption Enumeration 6162

Types **6163**

OPConvert.TSOAPConvertOptions Type 6163

OPToSOAPDomConv Namespace 6165

Classes **6167**

ESOAPDomConvertError Class 6167

TOPToSoapDomConvert Class 6167

TOPToSoapDomConvert Members 6168

TOPToSoapDomConvert.Create Constructor 6171

TOPToSoapDomConvert.Destroy Destructor 6171

TOPToSoapDomConvert Methods 6171

TOPToSoapDomConvert Properties 6176

TSOAPDomConv Class 6178

TSOAPDomConv Members 6179

TSOAPDomConv.Create Constructor 6180

TSOAPDomConv.Destroy Destructor 6181

TSOAPDomConv Methods 6181

TSOAPDomConv Properties 6201

TSOAPDomConv Events 6202

Functions **6203**

OPToSOAPDomConv.GetOrdPropEx Function 6203

Structs, Records, Enums **6204**

OPToSOAPDomConv.ConvNodeState Enumeration 6204

OPToSOAPDomConv.TMultiRefNodeElem Record 6204

OPToSOAPDomConv.TMultiRefNodeMapElem Record 6205

OPToSOAPDomConv.TSOAPArrayElemDesc Record 6205

Types **6206**

OPToSOAPDomConv.TMemberDataNotReceivedEvent Type 6206

OPToSOAPDomConv.TMultiRefNodeMap Type 6206

OPToSOAPDomConv.TMultiRefNodes Type 6207

OPToSOAPDomConv.TSOAPArrayDesc Type 6207

OPToSOAPDomConv.TUnhandledNodeEvent Type 6207

OPToSOAPDomConv.TXMLNodeArray Type 6208

Variables **6209**

OPToSOAPDomConv.DefArrayElemName Variable 6209

Constants **6210**
OPToSOAPDomConv.SVarArrayType Constant 6210

OrbPas Namespace 6211

Interfaces **6213**
IBOA Interface 6213
IBOA Members 6213
IBOA Methods 6214
ICorbaObj Interface 6215
ICorbaObj Members 6215
ICorbaObj Methods 6215
IMarshalInBuffer Interface 6216
IMarshalInBuffer Members 6216
IMarshalInBuffer Methods 6217
IMarshalOutBuffer Interface 6220
IMarshalOutBuffer Members 6221
IMarshalOutBuffer Methods 6221
IORB Interface 6225
IORB Members 6225
IORB Methods 6225
ISkeleton Interface 6229
ISkeleton Members 6229
ISkeleton Methods 6230
ISkeletonObject Interface 6230
ISkeletonObject Members 6231
ISkeletonObject Methods 6231
IStub Interface 6232
IStub Members 6232
IStub Methods 6232
IStubObject Interface 6235
IStubObject Members 6235
IStubObject Methods 6235
ITypeCode Interface 6236
ITypeCode Members 6236
ITypeCode Methods 6237

Functions **6241**
OrbPas.AnyToVariant Function 6241
OrbPas.BindStub Function 6241

OrbPas.CorbaAnyType Function 6242
OrbPas.CorbaDuplicateAny Function 6242
OrbPas.CorbaReleaseAny Function 6242
OrbPas.CorbaStringFree Function 6243
OrbPas.CorbaWStringFree Function 6243
OrbPas.CreateSkeleton Function 6243
OrbPas.CreateUserException Function 6244
OrbPas.InitORB Function 6244
OrbPas.RegisterUserException Function 6244
OrbPas.SequenceToVariant Function 6245
OrbPas.UnRegisterUserException Function 6245
OrbPas.VariantToAny Function 6245

Structs, Records, Enums **6246**
OrbPas.TCKind Enumeration 6246
OrbPas.TStructMember Record 6247

Types **6248**
OrbPas.CorbaBoolean Type 6248
OrbPas.CorbaULong Type 6248
OrbPas.PCorbaAny Type 6249
OrbPas.PCorbaInterfaceDef Type 6249
OrbPas.PCorbaObject Type 6249
OrbPas.PCorbaOrb Type 6250
OrbPas.PExceptionDescription Type 6250
OrbPas.PMarshalInbuffer Type 6250
OrbPas.PMarshalOutbuffer Type 6251
OrbPas.PUserExceptionProxy Type 6251
OrbPas.TArgv Type 6251
OrbPas.TCopyUserExceptionProc Type 6252
OrbPas.TStructMembers Type 6252
OrbPas.TThrowUserExceptionProc Type 6252
OrbPas.TUserExceptionFactoryProc Type 6253

Outline Namespace 6255

Classes **6257**
EOutlineError Class 6257
OutlineError Class 6258
TCustomOutline Class 6258
TCustomOutline Members 6259

TCustomOutline.Create Constructor 6262
TCustomOutline.Destroy Destructor 6263
TCustomOutline Methods 6263
TCustomOutline Properties 6284
TCustomOutline Events 6294
TOutline Class 6296
TOutline Members 6296
TOutline Properties 6301
TOutline Events 6330
TOutlineNode Class 6363
TOutlineNode Members 6364
TOutlineNode.Create Constructor 6365
TOutlineNode.Destroy Destructor 6365
TOutlineNode Methods 6366
TOutlineNode Properties 6374

Structs, Records, Enums **6380**
Outline.TAttachMode Enumeration 6380
Outline.TOutlineBitmap Enumeration 6381
Outline.TOutlineNodeCompare Enumeration 6381
Outline.TOutlineOption Enumeration 6382
Outline.TOutlineStyle Enumeration 6382
Outline.TOutlineType Enumeration 6383

Types **6385**
Outline.EOutlineChange Type 6385
Outline.TBitmapArray Type 6385
Outline.TBitmapArrayRange Type 6386
Outline.TChangeRange Type 6386
Outline.TOutlineBitmaps Type 6386
Outline.TOutlineOptions Type 6387

PagItems Namespace 6389

Classes **6390**
TBasePageItemsProducer Class 6390
TBasePageItemsProducer Members 6391
TBasePageItemsProducer.Create Constructor 6391
TBasePageItemsProducer.Destroy Destructor 6391
TBasePageItemsProducer Methods 6392
TBasePageItemsProducer Properties 6393

TPageItemsProducer Class 6394
TPageItemsProducer Members 6394
TPageItemsProducer.Create Constructor 6395
TPageItemsProducer.Destroy Destructor 6395
TPageItemsProducer Methods 6395
TPageItemsProducer Properties 6396

PlatformDefaultStyleActnCtrls Namespace 6399

Classes **6400**
TPlatformDefaultStyleActionBars Class 6400
TPlatformDefaultStyleActionBars Members 6400
TPlatformDefaultStyleActionBars Methods 6400

Variables **6403**
PlatformDefaultStyleActnCtrls.PlatformDefaultStyle Variable 6403

Printers Namespace 6405

Classes **6406**
EPrinter Class 6406
TPrinter Class 6406
TPrinter Members 6407
TPrinter.Create Constructor 6408
TPrinter.Destroy Destructor 6408
TPrinter Methods 6408
TPrinter Properties 6418

Functions **6434**
Printers.AssignPrn Function 6434
Printers.Printer Function 6434
Printers.SetPrinter Function 6436

Structs, Records, Enums **6438**
Printers.TPrinterCapability Enumeration 6438
Printers.TPrinterOrientation Enumeration 6438
Printers.TPrinterState Enumeration 6439

Types **6440**
Printers.TPrinterCapabilities Type 6440

Provider Namespace 6441

Classes **6443**
EDSWriter Class 6443
EDSWriter Members 6444
EDSWriter.Create Constructor 6444
EDSWriter Properties 6444
TBaseProvider Class 6445
TBaseProvider Members 6446
TBaseProvider.Create Constructor 6447
TBaseProvider.Destroy Destructor 6448
TBaseProvider Methods 6448
TBaseProvider Properties 6455
TBaseProvider Events 6457
TCustomPacketWriter Class 6460
TCustomPacketWriter Members 6461
TCustomPacketWriter.Create Constructor 6461
TCustomPacketWriter.Destroy Destructor 6461
TCustomPacketWriter Fields 6462
TCustomPacketWriter Methods 6462
TCustomPacketWriter Properties 6463
TCustomProvider Class 6464
TCustomProvider Members 6465
TCustomProvider.Create Constructor 6466
TCustomProvider.Destroy Destructor 6467
TCustomProvider Methods 6467
TCustomProvider Properties 6481
TCustomProvider Events 6483
TCustomResolver Class 6491
TCustomResolver Members 6492
TCustomResolver.Create Constructor 6493
TCustomResolver.Destroy Destructor 6493
TCustomResolver Methods 6493
TCustomResolver Properties 6500
TDataPacketWriter Class 6501
TDataPacketWriter Members 6502
TDataPacketWriter.Destroy Destructor 6503
TDataPacketWriter Methods 6503
TDataPacketWriter Properties 6509
TDataPacketWriter Events 6510
TDataSetProvider Class 6510

TDataSetProvider Members 6511
TDataSetProvider.Create Constructor 6513
TDataSetProvider.Destroy Destructor 6513
TDataSetProvider Methods 6514
TDataSetProvider Properties 6524
TDataSetProvider Events 6529
TDataSetResolver Class 6540
TDataSetResolver Members 6541
TDataSetResolver.Create Constructor 6541
TDataSetResolver Methods 6542
TDataSetResolver Properties 6545
TLocalAppServer Class 6546
TLocalAppServer Members 6546
Create Constructor 6547
TLocalAppServer.Destroy Destructor 6547
TLocalAppServer Methods 6548
TPacketDataSet Class 6555
TPacketDataSet Members 6556
TPacketDataSet.Create Constructor 6556
TPacketDataSet Methods 6557
TPacketDataSet Properties 6562
TProvider Class 6563
TSQLResolver Class 6564
TSQLResolver Members 6564
TSQLResolver.Create Constructor 6565
TSQLResolver.Destroy Destructor 6565
TSQLResolver Methods 6566
TSQLResolver Properties 6571
TUpdateTree Class 6572
TUpdateTree Members 6572
TUpdateTree.Create Constructor 6573
TUpdateTree.Destroy Destructor 6573
TUpdateTree Methods 6574
TUpdateTree Properties 6576

Interfaces 6580

IProviderContainer Interface 6580
IProviderContainer Members 6580
IProviderContainer Methods 6580

Functions **6582**
Provider.GetObjectProperty Function 6582
Provider.GetStringProperty Function 6582
Provider.VarArrayFromStrings Function 6582
Provider.VarArrayFromWideStrings Function 6583

Structs, Records, Enums **6584**
Provider.TGetRecordOption Enumeration 6584
Provider.TProviderOption Enumeration 6585
Provider.TPutFieldInfo Record 6587
Provider.TResolverResponse Enumeration 6587

Types **6589**
Provider.PPutFieldInfo Type 6589
Provider.TAfterUpdateRecordEvent Type 6589
Provider.TBeforeCommitEvent Type 6590
Provider.TBeforeUpdateRecordEvent Type 6590
Provider.TDataRequestEvent Type 6590
Provider.TGetDSProps Type 6591
Provider.TGetParamsEvent Type 6591
Provider.TGetRecordOptions Type 6591
Provider.TGetTableNameEvent Type 6592
Provider.TInfoArray Type 6592
Provider.TOnValidate Type 6592
Provider.TProviderDataEvent Type 6593
Provider.TProviderOptions Type 6593
Provider.TPutFieldProc Type 6593
Provider.TResolverErrorEvent Type 6594

Variables **6595**
Provider.InformixLob Variable 6595

Constants **6596**
Provider.MetaDataOption Constant 6596
Provider.ResetOption Constant 6597
Provider.XMLOption Constant 6597
Provider.XMLUTF8Option Constant 6598

Registry Namespace 6599

Classes **6600**

ERegistryException Class 6600
TRegIniFile Class 6601
TRegIniFile Members 6601
Create Constructor 6603
TRegIniFile Methods 6604
TRegIniFile Properties 6610
TRegistry Class 6610
TRegistry Members 6613
Create Constructor 6615
TRegistry.Destroy Destructor 6617
TRegistry Methods 6618
TRegistry Properties 6656
TRegistryIniFile Class 6663
TRegistryIniFile Members 6667
Create Constructor 6668
TRegistryIniFile.Destroy Destructor 6668
TRegistryIniFile Methods 6669
TRegistryIniFile Properties 6690

Structs, Records, Enums 6691
Registry.TRegDataInfo Record 6691
Registry.TRegDataType Enumeration 6691
Registry.TRegKeyInfo Record 6692

ReqFiles Namespace 6695

Classes 6696
TWebRequestFile Class 6696
TWebRequestFile Members 6697
TWebRequestFile.Create Constructor 6697
TWebRequestFile.Destroy Destructor 6698
TWebRequestFile Methods 6698
TWebRequestFiles Class 6700
TWebRequestFiles Members 6700
TWebRequestFiles.Create Constructor 6701
TWebRequestFiles.Destroy Destructor 6701
TWebRequestFiles Methods 6701
TWebRequestFileStream Class 6703
TWebRequestFileStream Members 6704
TWebRequestFileStream.Create Constructor 6704

TWebRequestFileStream Methods 6704

ReqMulti Namespace 6705

Classes 6706

TMultipartContentParser Class 6706

TMultipartContentParser Members 6707

TMultipartContentParser.Destroy Destructor 6707

TMultipartContentParser Methods 6707

Ribbon Namespace 6713

Classes 6715

ERibbonException Class 6715

TApplicationMenu Class 6716

TApplicationMenu Members 6716

TApplicationMenu.Create Constructor 6717

TApplicationMenu.Destroy Destructor 6717

TApplicationMenu Methods 6717

TApplicationMenu Properties 6719

TCustomActionControlBar Class 6721

TCustomActionControlBar Members 6721

TCustomActionControlBar.Create Constructor 6722

TCustomActionControlBar Methods 6722

TCustomActionControlBar Properties 6723

TCustomRibbon Class 6724

TCustomRibbon Members 6725

TCustomRibbon.Create Constructor 6728

TCustomRibbon.Destroy Destructor 6728

TCustomRibbon Constants 6729

TCustomRibbon Methods 6733

TCustomRibbon Properties 6758

TCustomRibbon Nested Types 6765

TCustomRibbon Events 6766

TCustomRibbonColorMap Class 6767

TCustomRibbonColorMap Members 6767

TCustomRibbonColorMap Properties 6768

TCustomRibbonGroup Class 6771

TCustomRibbonGroup Members 6771

TCustomRibbonGroup.Create Constructor 6773

TCustomRibbonGroup.Destroy Destructor 6773
TCustomRibbonGroup Enumerations 6773
TCustomRibbonGroup Methods 6774
TCustomRibbonGroup Properties 6786
TCustomRibbonGroup Nested Types 6790
TCustomRibbonPage Class 6790
TCustomRibbonPage Members 6790
TCustomRibbonPage.Create Constructor 6792
TCustomRibbonPage.Destroy Destructor 6792
TCustomRibbonPage Constants 6793
TCustomRibbonPage Methods 6793
TCustomRibbonPage Properties 6802
TCustomRibbonQuickAccessPopupToolbar Class 6804
TCustomRibbonQuickAccessPopupToolbar Members 6804
TCustomRibbonQuickAccessPopupToolbar.Create Constructor 6805
TCustomRibbonQuickAccessPopupToolbar Methods 6805
TCustomRibbonQuickAccessToolbar Class 6807
TCustomRibbonQuickAccessToolbar Members 6808
TCustomRibbonQuickAccessToolbar.Create Constructor 6809
TCustomRibbonQuickAccessToolbar Methods 6809
TCustomRibbonQuickAccessToolbar Properties 6817
TCustomRibbonQuickAccessToolbar Nested Types 6819
TCustomTabData Class 6819
TCustomTabData Members 6820
TCustomTabData.Create Constructor 6820
TCustomTabData.Destroy Destructor 6821
TCustomTabData Methods 6821
TCustomTabData Properties 6822
THelpButton Class 6825
THelpButton Members 6825
THelpButton Methods 6826
TOffice2007Button Class 6826
TOffice2007Button Members 6827
TOffice2007Button.Create Constructor 6827
TOffice2007Button.Destroy Destructor 6828
TOffice2007Button Methods 6828
TOffice2007Button Properties 6832
TOffice2007Button Events 6833
TQuickAccessToolbar Class 6835

TQuickAccessToolbar Members 6835
TQuickAccessToolbar.Create Constructor 6836
TQuickAccessToolbar Enumerations 6836
TQuickAccessToolbar Methods 6837
TQuickAccessToolbar Properties 6838
TQuickAccessToolbar Events 6840
TQuickAccessToolbarGroupItem Class 6841
TQuickAccessToolbarGroupItem Members 6841
TQuickAccessToolbarGroupItem.Create Constructor 6841
TQuickAccessToolbarGroupItem.Destroy Destructor 6842
TQuickAccessToolbarGroupItem Properties 6842
TQuickAccessToolbarGroups Class 6843
TQuickAccessToolbarGroups Members 6843
TQuickAccessToolbarGroups.Create Constructor 6843
TQuickAccessToolbarGroups.Destroy Destructor 6844
TQuickAccessToolbarGroups Methods 6844
TRibbon Class 6845
TRibbon Members 6846
TRibbon Properties 6849
TRibbon Events 6863
TRibbonCustomizeDlg Class 6864
TRibbonCustomizeDlg Members 6864
TRibbonCustomizeDlg Methods 6865
TRibbonGroup Class 6865
TRibbonGroup Members 6866
TRibbonGroup Properties 6868
TRibbonGroupCollapsedControl Class 6871
TRibbonGroupCollapsedControl Members 6871
TRibbonGroupCollapsedControl.Create Constructor 6872
TRibbonGroupCollapsedControl.Destroy Destructor 6873
TRibbonGroupCollapsedControl Methods 6873
TRibbonGroupCollapsedControl Properties 6880
TRibbonGroupDockForm Class 6884
TRibbonGroupDockForm Members 6884
TRibbonGroupDockForm.Create Constructor 6885
TRibbonGroupDockForm Methods 6885
TRibbonGroupDockForm Properties 6886
TRibbonGroupScrollButton Class 6886
TRibbonGroupScrollButton Members 6887

TRibbonGroupScrollButton.Create Constructor 6887
TRibbonGroupScrollButton Methods 6888
TRibbonGroupScrollButton Properties 6889
TRibbonLunaColorMap Class 6890
TRibbonLunaColorMap Members 6890
TRibbonLunaColorMap Methods 6891
TRibbonLunaColorMap Properties 6891
TRibbonObsidianColorMap Class 6896
TRibbonObsidianColorMap Members 6896
TRibbonObsidianColorMap Methods 6897
TRibbonObsidianColorMap Properties 6897
TRibbonPage Class 6901
TRibbonPage Members 6902
TRibbonPage Properties 6903
TRibbonQuickAccessPopupToolbar Class 6911
TRibbonQuickAccessToolbar Class 6911
TRibbonQuickAccessToolbar Members 6912
TRibbonQuickAccessToolbar Properties 6913
TRibbonSilverColorMap Class 6914
TRibbonSilverColorMap Members 6914
TRibbonSilverColorMap Methods 6915
TRibbonSilverColorMap Properties 6915
TRibbonTabItem Class 6920
TRibbonTabItem Members 6920
TRibbonTabItem.Destroy Destructor 6920
TRibbonTabItem Methods 6921
TRibbonTabItem Properties 6922
TRibbonTabs Class 6925
TRibbonTabs Members 6925
TRibbonTabs.Create Constructor 6926
TRibbonTabs Methods 6926
TRibbonTabs Properties 6929

Interfaces 6931
IRibbon Interface 6931
IRibbon Members 6931
IRibbon Methods 6932
IRibbon Properties 6932
IRibbonGroup Interface 6933
IRibbonGroup Members 6933

IRibbonGroup Methods 6933
IRibbonQuickAccessToolbar Interface 6934
IRibbonQuickAccessToolbar Members 6935
IRibbonQuickAccessToolbar Methods 6935

Structs, Records, Enums 6936
Ribbon.TCommandType Enumeration 6936
Ribbon.TIconSize Enumeration 6936
Ribbon.TRibbonGroupVariant Enumeration 6937
Ribbon.TRibbonMetric Enumeration 6937
Ribbon.TRibbonScrollButton Enumeration 6938
Ribbon.TTabActiveScrollButton Enumeration 6938
Ribbon.TTabSeparatorState Enumeration 6939

Types 6940
Ribbon.TCustomRibbonClass Type 6940
Ribbon.TRecentItemClickEvent Type 6940
Ribbon.TRibbonScrollButtons Type 6941

RibbonActnCtrls Namespace 6943

Classes 6946
TCustomRibbonComboBox Class 6947
TCustomRibbonComboBox Members 6947
TCustomRibbonComboBox.Create Constructor 6948
TCustomRibbonComboBox.Destroy Destructor 6948
TCustomRibbonComboBox Methods 6948
TCustomRibbonComboBox Properties 6954
TKeyTipList Class 6956
TKeyTipList Members 6956
TKeyTipList.Create Constructor 6957
TKeyTipList.Destroy Destructor 6957
TKeyTipList Methods 6957
TKeyTipList Properties 6958
TKeyTipListEnumerator Class 6960
TKeyTipListEnumerator Members 6960
TKeyTipListEnumerator.Create Constructor 6961
TKeyTipListEnumerator Methods 6961
TKeyTipListEnumerator Properties 6961
TKeyTipMatchItem Class 6962
TKeyTipMatchItem Members 6962

TKeyTipMatchItem.Create Constructor 6963
TKeyTipMatchItem Properties 6963
TKeyTipValidator Class 6964
TKeyTipValidator Members 6965
TKeyTipValidator Methods 6965
TRibbonAddRemoveItem Class 6965
TRibbonAddRemoveItem Members 6966
TRibbonAddRemoveItem Methods 6966
TRibbonApplicationButton Class 6967
TRibbonApplicationButton Members 6967
TRibbonApplicationButton Methods 6967
TRibbonApplicationMenuButton Class 6968
TRibbonApplicationMenuButton Members 6968
TRibbonApplicationMenuButton.Create Constructor 6969
TRibbonApplicationMenuButton.Destroy Destructor 6969
TRibbonApplicationMenuButton Methods 6970
TRibbonApplicationMenuButton Properties 6975
TRibbonApplicationMenuItem Class 6976
TRibbonApplicationMenuItem Members 6977
TRibbonApplicationMenuItem.Create Constructor 6977
TRibbonApplicationMenuItem Methods 6978
TRibbonBaseButtonControl Class 6980
TRibbonBaseButtonControl Members 6981
TRibbonBaseButtonControl.Create Constructor 6981
TRibbonBaseButtonControl.Destroy Destructor 6982
TRibbonBaseButtonControl Methods 6983
TRibbonBaseButtonControl Properties 6989
TRibbonCheckBoxControl Class 6990
TRibbonCheckBoxControl Members 6990
TRibbonCheckBoxControl Methods 6991
TRibbonComboBox Class 6992
TRibbonComboBox Members 6993
TRibbonComboBox Properties 6994
TRibbonComboBox Events 7012
TRibbonComboBoxItem Class 7045
TRibbonComboBoxItem Members 7045
TRibbonComboBoxItem Methods 7045
TRibbonComboControl Class 7047
TRibbonComboControl Members 7047

TRibbonComboControl.Create Constructor 7048
TRibbonComboControl.Destroy Destructor 7049
TRibbonComboControl Methods 7049
TRibbonComboControl Properties 7060
TRibbonCustomControl Class 7061
TRibbonCustomControl Members 7061
TRibbonCustomControl.Create Constructor 7062
TRibbonCustomControl.Destroy Destructor 7062
TRibbonCustomControl Methods 7063
TRibbonCustomControl Properties 7073
TRibbonCustomizeItem Class 7073
TRibbonCustomizeItem Members 7074
TRibbonCustomizeItem Methods 7074
TRibbonCustomStatusControl Class 7076
TRibbonCustomStatusControl Members 7076
TRibbonCustomStatusControl Methods 7077
TRibbonDropDownButton Class 7078
TRibbonDropDownButton Members 7078
TRibbonDropDownButton Methods 7079
TRibbonGallery Class 7083
TRibbonGalleryControl Class 7083
TRibbonGalleryControl Members 7083
TRibbonGalleryControl Methods 7084
TRibbonGalleryControl Properties 7085
TRibbonGalleryControlMenuItem Class 7085
TRibbonGalleryControlMenuItem Members 7086
TRibbonGalleryControlMenuItem Methods 7086
TRibbonGalleryControlMenuItem Properties 7091
TRibbonGalleryItem Class 7091
TRibbonGalleryItem Members 7092
TRibbonGalleryItem Methods 7093
TRibbonGalleryMenuItem Class 7093
TRibbonGalleryMenuItem Members 7094
TRibbonGalleryMenuItem Methods 7094
TRibbonGallerySplitButton Class 7096
TRibbonGallerySplitButton Members 7096
TRibbonGallerySplitButton Methods 7098
TRibbonGripItem Class 7098
TRibbonGripItem Members 7099

TRibbonGripItem.Create Constructor 7099
TRibbonGripItem Methods 7100
TRibbonGripItem Properties 7101
TRibbonGroupDropDown Class 7101
TRibbonGroupDropDown Members 7102
TRibbonGroupDropDown Methods 7103
TRibbonGroupDropDown Properties 7105
TRibbonLargeButtonControl Class 7105
TRibbonLargeButtonControl Members 7105
TRibbonLargeButtonControl Methods 7106
TRibbonLargeSplitButton Class 7108
TRibbonLargeSplitButton Members 7109
TRibbonLargeSplitButton Methods 7110
TRibbonLargeSplitButton Properties 7115
TRibbonMainMenuItem Class 7115
TRibbonMainMenuItem Members 7115
TRibbonMainMenuItem Methods 7116
TRibbonMainMenuSeparator Class 7117
TRibbonMainMenuSeparator Members 7117
TRibbonMainMenuSeparator Methods 7118
TRibbonMenuItem Class 7118
TRibbonMenuItem Members 7118
TRibbonMenuItem.Create Constructor 7119
TRibbonMenuItem Methods 7119
TRibbonMenuItem Properties 7123
TRibbonMenuSeparatorItem Class 7123
TRibbonMenuSeparatorItem Members 7124
TRibbonMenuSeparatorItem Methods 7124
TRibbonMenuSmallButtonControl Class 7125
TRibbonMenuSmallButtonControl Members 7125
TRibbonMenuSmallButtonControl.Create Constructor 7126
TRibbonMenuSmallButtonControl Methods 7127
TRibbonOptionButtonControl Class 7128
TRibbonOptionButtonControl Members 7128
TRibbonOptionButtonControl Methods 7129
TRibbonQATScrollButton Class 7130
TRibbonQATScrollButton Members 7130
TRibbonQATScrollButton Methods 7130
TRibbonQATScrollButton Properties 7133

TRibbonRadioButtonControl Class 7134
TRibbonRadioButtonControl Members 7135
TRibbonRadioButtonControl Methods 7136
TRibbonRecentButtonControl Class 7137
TRibbonRecentButtonControl Members 7137
TRibbonRecentButtonControl Methods 7138
TRibbonRichContentItem Class 7139
TRibbonRichContentItem Members 7139
TRibbonRichContentItem.Create Constructor 7140
TRibbonRichContentItem Methods 7140
TRibbonRichContentMainMenuItem Class 7143
TRibbonRichContentMainMenuItem Members 7143
TRibbonRichContentMainMenuItem.Create Constructor 7144
TRibbonRichContentMainMenuItem Methods 7145
TRibbonSeparator Class 7147
TRibbonSeparator Members 7147
TRibbonSeparator.Create Constructor 7147
TRibbonSeparator Methods 7148
TRibbonSmallButtonControl Class 7150
TRibbonSmallButtonControl Members 7150
TRibbonSmallButtonControl Methods 7151
TRibbonSmallSplitButton Class 7152
TRibbonSmallSplitButton Members 7152
TRibbonSmallSplitButton.Create Constructor 7153
TRibbonSmallSplitButton Fields 7154
TRibbonSmallSplitButton Methods 7155
TRibbonSmallSplitButton Properties 7158
TRibbonSpinButton Class 7159
TRibbonSpinButton Members 7159
TRibbonSpinButton.Create Constructor 7160
TRibbonSpinButton Methods 7161
TRibbonSpinButton Properties 7163
TRibbonSpinButton Events 7183
TRibbonSpinEdit Class 7198
TRibbonSpinEdit Members 7198
TRibbonSpinEdit.Create Constructor 7200
TRibbonSpinEdit.Destroy Destructor 7200
TRibbonSpinEdit Methods 7201
TRibbonSpinEdit Properties 7206

TRibbonSpinEdit Events 7227
TRibbonSplitMenuItem Class 7257
TRibbonSplitMenuItem Members 7257
TRibbonSplitMenuItem.Create Constructor 7258
TRibbonSplitMenuItem Fields 7258
TRibbonSplitMenuItem Methods 7259
TRibbonSplitMenuItem Properties 7262
TRibbonStyleMenuButton Class 7262
TRibbonStyleMenuButton Members 7262
TRibbonStyleMenuButton Methods 7263
TRibbonTextControl Class 7263
TRibbonTextControl Members 7264
TRibbonTextControl Methods 7264
TRibbonToolScrollButton Class 7265
TRibbonToolScrollButton Members 7266
TRibbonToolScrollButton Methods 7266
TTimerSpeedButton Class 7267
TTimerSpeedButton Members 7267
TTimerSpeedButton.Create Constructor 7267
TTimerSpeedButton.Destroy Destructor 7268
TTimerSpeedButton Methods 7268
TTimerSpeedButton Properties 7270

Interfaces 7272
IRibbonBarKeyTip Interface 7272
IRibbonBarKeyTip Members 7272
IRibbonBarKeyTip Methods 7273
IRibbonContainedControl Interface 7274
IRibbonContainedControl Members 7274
IRibbonContainedControl Methods 7275
IRibbonContainedControl Properties 7275
IRibbonControl Interface 7276
IRibbonControl Members 7276
IRibbonControl Methods 7276
IRibbonGroupDropdown Interface 7277
IRibbonGroupDropdown Members 7277
IRibbonGroupDropdown Methods 7277
IRibbonKeyTip Interface 7278
IRibbonKeyTip Members 7279
IRibbonKeyTip Methods 7279

IRibbonKeyTip Properties 7282
IRibbonSplitButton Interface 7282
IRibbonSplitButton Members 7283
IRibbonSplitButton Methods 7283

Functions **7284**
RibbonActnCtrls.DrawLine Function 7284
RibbonActnCtrls.IsLeftMouseButtonPressed Function 7284
RibbonActnCtrls.ShowKeyTip Function 7284

Structs, Records, Enums **7286**
RibbonActnCtrls.TKeyTipHandled Enumeration 7286
RibbonActnCtrls.TKeyTipMatchType Enumeration 7287
RibbonActnCtrls.TKeyTipPosition Enumeration 7287
RibbonActnCtrls.TKeyTipState Enumeration 7288
RibbonActnCtrls.TQATScrollMode Enumeration 7289
RibbonActnCtrls.TSpinButtonStyle Enumeration 7290

Types **7291**
RibbonActnCtrls.TNumGlyphs Type 7291
RibbonActnCtrls.TTimeBtnState Type 7291

Constants **7292**
RibbonActnCtrls.UM_FADECONTROL Constant 7292

RibbonActnMenus Namespace 7293

Classes **7294**
TOptionItem Class 7294
TOptionItem Members 7294
TOptionItem Methods 7295
TOptionItem Properties 7296
TOptionItems Class 7298
TOptionItems Members 7298
TOptionItems Methods 7298
TOptionItems Properties 7299
TRibbonActionPopupMenu Class 7300
TRibbonActionPopupMenu Members 7301
TRibbonActionPopupMenu Methods 7301
TRibbonActionPopupMenu Properties 7302
TRibbonApplicationMenuBar Class 7302
TRibbonApplicationMenuBar Members 7303

TRibbonApplicationMenuBar.Create Constructor 7304
TRibbonApplicationMenuBar.Destroy Destructor 7304
TRibbonApplicationMenuBar Methods 7305
TRibbonApplicationMenuBar Properties 7309
TRibbonApplicationPopupMenu Class 7321
TRibbonApplicationPopupMenu Members 7321
TRibbonApplicationPopupMenu.Create Constructor 7322
TRibbonApplicationPopupMenu.Destroy Destructor 7322
TRibbonApplicationPopupMenu Methods 7322
TRibbonQATCustomizeMenu Class 7328
TRibbonQATCustomizeMenu Members 7328
TRibbonQATCustomizeMenu Methods 7328
TRibbonRecentDocuments Class 7330
TRibbonRecentDocuments Members 7331
TRibbonRecentDocuments.Create Constructor 7331
TRibbonRecentDocuments Methods 7332
TRibbonRecentDocuments Properties 7332
TRibbonStyleMainPopupMenu Class 7335
TRibbonStyleMainPopupMenu Members 7335
TRibbonStyleMainPopupMenu.Create Constructor 7335
TRibbonStyleMainPopupMenu Methods 7336
TRibbonStylePopupMenu Class 7336
TRibbonStylePopupMenu Members 7337
TRibbonStylePopupMenu.Create Constructor 7337
TRibbonStylePopupMenu.Destroy Destructor 7338
TRibbonStylePopupMenu Methods 7338
TRibbonStylePopupMenu Properties 7340

RibbonConsts Namespace 7341

Constants 7342

RibbonConsts.SAddToQuickAccessToolbar Constant 7342
RibbonConsts.SCommandIsDisabled Constant 7343
RibbonConsts.SCustomizeQuickAccessToolbar Constant 7343
RibbonConsts.SDefaultFooterText Constant 7343
RibbonConsts.SDuplicateRibbonComponentsNotAllowed Constant 7344
RibbonConsts.SGenerateScreenTipsError Constant 7344
RibbonConsts.SInvalidRibbonGroupParent Constant 7344
RibbonConsts.SInvalidRibbonPageChild Constant 7345

RibbonConsts.SInvalidRibbonPageParent Constant 7345
RibbonConsts.SInvalidScreenTipItemAssign Constant 7345
RibbonConsts.SMinimizeTheRibbon Constant 7346
RibbonConsts.SMoreCommands Constant 7346
RibbonConsts.SMoreControls Constant 7346
RibbonConsts.SPageCaptionExists Constant 7347
RibbonConsts.SQATShowAboveTheRibbon Constant 7347
RibbonConsts.SQATShowBelowTheRibbon Constant 7347
RibbonConsts.SQuickAccessToolbarGlyphRequired Constant 7348
RibbonConsts.SRecentDocuments Constant 7348
RibbonConsts.SRemoveFromQuickAccessToolbar Constant 7348
RibbonConsts.SRibbonControlInvalidParent Constant 7349
RibbonConsts.SRibbonTabIndexInvalid Constant 7349
RibbonConsts.SShowQuickAccessToolbarAbove Constant 7349
RibbonConsts.SShowQuickAccessToolbarBelow Constant 7350
RibbonConsts.SUseTheGlyphProperty Constant 7350

RibbonGalleryBar Namespace 7351

Classes 7352

TCustomRibbonGalleryBar Class 7352
TCustomRibbonGalleryBar Members 7352
TCustomRibbonGalleryBar.Create Constructor 7353
TCustomRibbonGalleryBar Methods 7353
TCustomRibbonGalleryBar Properties 7354
TCustomRibbonGalleryPopupBar Class 7358
TCustomRibbonGalleryPopupBar Members 7359
TCustomRibbonGalleryPopupBar Methods 7359
TRibbonGalleryBar Class 7360
TRibbonGalleryBar Members 7361
TRibbonGalleryBar Properties 7361
TRibbonGalleryPopupBar Class 7362
TRibbonPopupGalleryBar Class 7362
TRibbonPopupGalleryBar Members 7363
TRibbonPopupGalleryBar Methods 7363

RibbonStyleActnCtrls Namespace 7367

Classes 7369

TAlphaBitmap Class 7369

TAlphaBitmap Members 7369
TAlphaBitmap.Create Constructor 7371
TCustomRibbonSkin Class 7372
TCustomRibbonSkin Members 7372
TCustomRibbonSkin.Create Constructor 7373
TCustomRibbonSkin.Destroy Destructor 7373
TCustomRibbonSkin Enumerations 7373
TCustomRibbonSkin Methods 7374
TCustomRibbonSkin Properties 7375
TRibbonStyleActionBars Class 7376
TRibbonStyleActionBars Members 7376
TRibbonStyleActionBars.Create Constructor 7377
TRibbonStyleActionBars.Destroy Destructor 7377
TRibbonStyleActionBars Methods 7377
TRibbonStyleActionBars Properties 7381

Structs, Records, Enums 7383
RibbonStyleActnCtrls.TRibbonElement Enumeration 7383
RibbonStyleActnCtrls.TSkinApplicationMenu Enumeration 7384
RibbonStyleActnCtrls.TSkinButton Enumeration 7386
RibbonStyleActnCtrls.TSkinCheckBox Enumeration 7390
RibbonStyleActnCtrls.TSkinComboBox Enumeration 7391
RibbonStyleActnCtrls.TSkinForm Enumeration 7391
RibbonStyleActnCtrls.TSkinGallery Enumeration 7394
RibbonStyleActnCtrls.TSkinMenu Enumeration 7395
RibbonStyleActnCtrls.TSkinQuickAccessToolbar Enumeration 7397
RibbonStyleActnCtrls.TSkinRadioButton Enumeration 7398
RibbonStyleActnCtrls.TSkinRibbon Enumeration 7399
RibbonStyleActnCtrls.TSkinRibbonGroup Enumeration 7400
RibbonStyleActnCtrls.TSkinScrollBar Enumeration 7402
RibbonStyleActnCtrls.TSkinSpinEdit Enumeration 7403
RibbonStyleActnCtrls.TSkinStatusBar Enumeration 7403
RibbonStyleActnCtrls.TSkinTab Enumeration 7404

Types 7406
RibbonStyleActnCtrls.TCustomRibbonSkinClass Type 7406

Rio Namespace 7407

Classes 7408
TRIO Class 7408

TRIO Members 7409
TRIO.Create Constructor 7410
TRIO.Destroy Destructor 7410
TRIO Fields 7410
TRIO Methods 7412
TRIO Properties 7415
TRIO Events 7417

Interfaces **7420**
IRIOAccess Interface 7420
IRIOAccess Members 7420
IRIOAccess Methods 7421
IRIOAccess Properties 7421

Types **7422**
Rio.TAfterExecuteEvent Type 7422
Rio.TBeforeExecuteEvent Type 7422

Constants **7423**
Rio.StubSize Constant 7423

RTLConsts Namespace 7425

Constants **7431**
RTLConsts.hNoContext Constant 7436
RTLConsts.hNoContextFound Constant 7436
RTLConsts.hNoFilterViewer Constant 7436
RTLConsts.hNoSystem Constant 7437
RTLConsts.hNoTableOfContents Constant 7437
RTLConsts.hNothingFound Constant 7437
RTLConsts.hNoTopics Constant 7438
RTLConsts.SAcreFeetDescription Constant 7438
RTLConsts.SAcreInchesDescription Constant 7438
RTLConsts.SAcresDescription Constant 7439
RTLConsts.SAncestorNotFound Constant 7439
RTLConsts.SAngstromsDescription Constant 7439
RTLConsts.SAnsiUTF8Expected Constant 7440
RTLConsts.SAreaDescription Constant 7440
RTLConsts.SAresDescription Constant 7440
RTLConsts.sArgument_InvalidHighSurrogate Constant 7441
RTLConsts.sArgument_InvalidLowSurrogate Constant 7441

RTLConsts.sArgumentOutOfRange_Index Constant 7441
RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate Constant 7442
RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate Constant 7442
RTLConsts.sArgumentOutOfRange_InvalidUTF32 Constant 7443
RTLConsts.sArgumentOutOfRange_StringIndex Constant 7443
RTLConsts.SAssignError Constant 7443
RTLConsts.SAstronomicalUnitsDescription Constant 7444
RTLConsts.sAsyncSocketError Constant 7444
RTLConsts.SBitsIndexError Constant 7444
RTLConsts.SBucketListLocked Constant 7445
RTLConsts.sCannotCallAcquireOnConditionVar Constant 7445
RTLConsts.sCannotCreateSocket Constant 7445
RTLConsts.sCannotListenOnOpen Constant 7446
RTLConsts.sCantChangeWhileActive Constant 7446
RTLConsts.SCantWriteResourceStreamError Constant 7446
RTLConsts.SCelsiusDescription Constant 7447
RTLConsts.SCentaresDescription Constant 7447
RTLConsts.SCentigramsDescription Constant 7447
RTLConsts.SCentiLitersDescription Constant 7448
RTLConsts.SCentimetersDescription Constant 7448
RTLConsts.SCenturiesDescription Constant 7448
RTLConsts.SChainsDescription Constant 7449
RTLConsts.SCharExpected Constant 7449
RTLConsts.SCheckSynchronizeError Constant 7449
RTLConsts.SClassNotFound Constant 7450
RTLConsts.SCmplxCouldNotParseImaginary Constant 7450
RTLConsts.SCmplxCouldNotParsePlus Constant 7450
RTLConsts.SCmplxCouldNotParseReal Constant 7451
RTLConsts.SCmplxCouldNotParseSymbol Constant 7451
RTLConsts.SCmplxErrorSuffix Constant 7451
RTLConsts.SCmplxUnexpectedChars Constant 7452
RTLConsts.SCmplxUnexpectedEOS Constant 7452
RTLConsts.SComponentNameTooLong Constant 7452
RTLConsts.SConvDuplicateFamily Constant 7453
RTLConsts.SConvDuplicateType Constant 7453
RTLConsts.SConvFactorZero Constant 7453
RTLConsts.SConvIllegalFamily Constant 7454
RTLConsts.SConvIllegalType Constant 7454
RTLConsts.SConvIncompatibleTypes2 Constant 7454

RTLConsts.SConvIncompatibleTypes3 Constant 7455
RTLConsts.SConvIncompatibleTypes4 Constant 7455
RTLConsts.SConvStrParseError Constant 7455
RTLConsts.SConvUnknownDescription Constant 7456
RTLConsts.SConvUnknownDescriptionWithPrefix Constant 7456
RTLConsts.SConvUnknownFamily Constant 7456
RTLConsts.SConvUnknownType Constant 7457
RTLConsts.SCordFeetDescription Constant 7457
RTLConsts.SCordsDescription Constant 7457
RTLConsts.SCubicCentimetersDescription Constant 7458
RTLConsts.SCubicDecametersDescription Constant 7458
RTLConsts.SCubicDecimetersDescription Constant 7458
RTLConsts.SCubicFeetDescription Constant 7459
RTLConsts.SCubicHectometersDescription Constant 7459
RTLConsts.SCubicInchesDescription Constant 7459
RTLConsts.SCubicKilometersDescription Constant 7460
RTLConsts.SCubicMetersDescription Constant 7460
RTLConsts.SCubicMilesDescription Constant 7460
RTLConsts.SCubicMillimetersDescription Constant 7461
RTLConsts.SCubicYardsDescription Constant 7461
RTLConsts.SCubitsDescription Constant 7461
RTLConsts.SDateTimeDescription Constant 7462
RTLConsts.SDaysDescription Constant 7462
RTLConsts.SDecadesDescription Constant 7462
RTLConsts.SDecagramsDescription Constant 7463
RTLConsts.SDecaLitersDescription Constant 7463
RTLConsts.SDecametersDescription Constant 7463
RTLConsts.SDecasteresDescription Constant 7464
RTLConsts.SDecigramsDescription Constant 7464
RTLConsts.SDeciLitersDescription Constant 7464
RTLConsts.SDecimetersDescription Constant 7465
RTLConsts.SDecisteresDescription Constant 7465
RTLConsts.SDelimiterQuoteCharError Constant 7465
RTLConsts.SDistanceDescription Constant 7466
RTLConsts.SDramsDescription Constant 7466
RTLConsts.SDryBucketsDescription Constant 7466
RTLConsts.SDryBushelsDescription Constant 7467
RTLConsts.SDryGallonsDescription Constant 7467
RTLConsts.SDryPecksDescription Constant 7467

RTLConsts.SDryPintsDescription Constant 7468
RTLConsts.SDryQuartsDescription Constant 7468
RTLConsts.SDuplicateClass Constant 7468
RTLConsts.SDuplicateItem Constant 7469
RTLConsts.SDuplicateName Constant 7469
RTLConsts.SDuplicateString Constant 7469
RTLConsts.SFahrenheitDescription Constant 7470
RTLConsts.SFailedToCallConstructor Constant 7470
RTLConsts.SFathomsDescription Constant 7470
RTLConsts.SFCreateError Constant 7471
RTLConsts.SFCreateErrorEx Constant 7471
RTLConsts.SFeetDescription Constant 7471
RTLConsts.SFixedColTooBig Constant 7472
RTLConsts.SFixedRowTooBig Constant 7472
RTLConsts.SFluidCupsDescription Constant 7472
RTLConsts.SFluidGallonsDescription Constant 7473
RTLConsts.SFluidGillsDescription Constant 7473
RTLConsts.SFluidOuncesDescription Constant 7473
RTLConsts.SFluidPintsDescription Constant 7474
RTLConsts.SFluidQuartsDescription Constant 7474
RTLConsts.SFluidTablespoonsDescription Constant 7474
RTLConsts.SFluidTeaspoonsDescription Constant 7475
RTLConsts.SFOpenError Constant 7475
RTLConsts.SFOpenErrorEx Constant 7475
RTLConsts.SFortnightsDescription Constant 7476
RTLConsts.SFurlongsDescription Constant 7476
RTLConsts.SGigametersDescription Constant 7476
RTLConsts.SGrainsDescription Constant 7477
RTLConsts.SGramsDescription Constant 7477
RTLConsts.SGridTooLarge Constant 7477
RTLConsts.SHandsDescription Constant 7478
RTLConsts.SHectaresDescription Constant 7478
RTLConsts.SHectogramsDescription Constant 7478
RTLConsts.SHectoLitersDescription Constant 7479
RTLConsts.SHectometersDescription Constant 7479
RTLConsts.SHoursDescription Constant 7479
RTLConsts.SIdentifierExpected Constant 7480
RTLConsts.SInchesDescription Constant 7480
RTLConsts.SIndexOutOfRange Constant 7480

RTLConsts.SIniFileWriteError Constant 7481
RTLConsts.SInvalidActionCreation Constant 7481
RTLConsts.SInvalidActionEnumeration Constant 7481
RTLConsts.SInvalidActionRegistration Constant 7482
RTLConsts.SInvalidActionUnregistration Constant 7482
RTLConsts.SInvalidBinary Constant 7482
RTLConsts.SInvalidDate Constant 7483
RTLConsts.SInvalidDateDay Constant 7483
RTLConsts.SInvalidDateMonthWeek Constant 7483
RTLConsts.SInvalidDateTime Constant 7484
RTLConsts.SInvalidDateWeek Constant 7484
RTLConsts.SInvalidDayOfWeekInMonth Constant 7484
RTLConsts.SInvalidFileName Constant 7485
RTLConsts.SInvalidImage Constant 7485
RTLConsts.SInvalidInteger Constant 7485
RTLConsts.SInvalidJulianDate Constant 7486
RTLConsts.SInvalidMask Constant 7486
RTLConsts.SInvalidName Constant 7486
RTLConsts.SInvalidProperty Constant 7487
RTLConsts.SInvalidPropertyElement Constant 7487
RTLConsts.SInvalidPropertyPath Constant 7487
RTLConsts.SInvalidPropertyType Constant 7488
RTLConsts.SInvalidPropertyValue Constant 7488
RTLConsts.SInvalidRegType Constant 7488
RTLConsts.SInvalidString Constant 7489
RTLConsts.SInvalidStringGridOp Constant 7489
RTLConsts.SInvalidTime Constant 7489
RTLConsts.SItemNotFound Constant 7490
RTLConsts.SJulianDateDescription Constant 7490
RTLConsts.SKelvinDescription Constant 7490
RTLConsts.SKilogramsDescription Constant 7491
RTLConsts.SKiloLitersDescription Constant 7491
RTLConsts.SKilometersDescription Constant 7491
RTLConsts.SLightYearsDescription Constant 7492
RTLConsts.SLineTooLong Constant 7492
RTLConsts.SLinksDescription Constant 7492
RTLConsts.SListCapacityError Constant 7493
RTLConsts.SListCountError Constant 7493
RTLConsts.SListIndexError Constant 7493

RTLConsts.SLitersDescription Constant 7494
RTLConsts.SLongTonsDescription Constant 7494
RTLConsts.SMaskEditErr Constant 7494
RTLConsts.SMaskErr Constant 7495
RTLConsts.SMassDescription Constant 7495
RTLConsts.SMegametersDescription Constant 7495
RTLConsts.SMemoryStreamError Constant 7496
RTLConsts.SMetersDescription Constant 7496
RTLConsts.SMetricTonsDescription Constant 7496
RTLConsts.SMicrogramsDescription Constant 7497
RTLConsts.SMicromicronsDescription Constant 7497
RTLConsts.SMicronsDescription Constant 7497
RTLConsts.SMilesDescription Constant 7498
RTLConsts.SMillenniaDescription Constant 7498
RTLConsts.SMilligramsDescription Constant 7498
RTLConsts.SMilliLitersDescription Constant 7499
RTLConsts.SMillimetersDescription Constant 7499
RTLConsts.SMillimicronsDescription Constant 7499
RTLConsts.SMilliSecondsDescription Constant 7500
RTLConsts.SMinutesDescription Constant 7500
RTLConsts.SMissingDateTimeField Constant 7500
RTLConsts.SModifiedJulianDateDescription Constant 7501
RTLConsts.SMonthsDescription Constant 7501
RTLConsts.SNanogramsDescription Constant 7501
RTLConsts.SNauticalMilesDescription Constant 7502
RTLConsts.sNoAddress Constant 7502
RTLConsts.SNoComSupport Constant 7502
RTLConsts.sNoConstruct Constant 7503
RTLConsts.SNotPrinting Constant 7503
RTLConsts.SNumberExpected Constant 7503
RTLConsts.SOuncesDescription Constant 7504
RTLConsts.SPacesDescription Constant 7504
RTLConsts.SParsecsDescription Constant 7504
RTLConsts.SParseError Constant 7505
RTLConsts.SPicasDescription Constant 7505
RTLConsts.SPointsDescription Constant 7505
RTLConsts.SPoundsDescription Constant 7506
RTLConsts.SPrinting Constant 7506
RTLConsts.SPropertyException Constant 7506

RTLConsts.SRankineDescription Constant 7507
RTLConsts.SReadError Constant 7507
RTLConsts.SReadOnlyProperty Constant 7507
RTLConsts.SReaumurDescription Constant 7508
RTLConsts.SRegCreateFailed Constant 7508
RTLConsts.SRegGetDataFailed Constant 7508
RTLConsts.SRegisterError Constant 7509
RTLConsts.SRegSetDataFailed Constant 7509
RTLConsts.SResNotFound Constant 7509
RTLConsts.SRodsDescription Constant 7510
RTLConsts.SSecondsDescription Constant 7510
RTLConsts.SSeekNotImplemented Constant 7510
RTLConsts.sSocketAlreadyOpen Constant 7511
RTLConsts.sSocketIOError Constant 7511
RTLConsts.sSocketMustBeBlocking Constant 7511
RTLConsts.sSocketRead Constant 7512
RTLConsts.sSocketWrite Constant 7512
RTLConsts.SSortedListError Constant 7512
RTLConsts.SSquareCentimetersDescription Constant 7513
RTLConsts.SSquareDecametersDescription Constant 7513
RTLConsts.SSquareDecimetersDescription Constant 7513
RTLConsts.SSquareFeetDescription Constant 7514
RTLConsts.SSquareHectometersDescription Constant 7514
RTLConsts.SSquareInchesDescription Constant 7514
RTLConsts.SSquareKilometersDescription Constant 7515
RTLConsts.SSquareMetersDescription Constant 7515
RTLConsts.SSquareMilesDescription Constant 7515
RTLConsts.SSquareMillimetersDescription Constant 7516
RTLConsts.SSquareRodsDescription Constant 7516
RTLConsts.SSquareYardsDescription Constant 7516
RTLConsts.SSteresDescription Constant 7517
RTLConsts.SStonesDescription Constant 7517
RTLConsts.SStreamSetSize Constant 7517
RTLConsts.SStringExpected Constant 7518
RTLConsts.SSymbolExpected Constant 7518
RTLConsts.STemperatureDescription Constant 7518
RTLConsts.SThreadCreateError Constant 7519
RTLConsts.SThreadError Constant 7519
RTLConsts.SThreadExternalTerminate Constant 7519

RTLConsts.SThreadExternalWait Constant 7520
RTLConsts.STimeDescription Constant 7520
RTLConsts.STimeEncodeError Constant 7520
RTLConsts.STonsDescription Constant 7521
RTLConsts.STooManyDeleted Constant 7521
RTLConsts.SUKBucketsDescription Constant 7521
RTLConsts.SUKBushelsDescription Constant 7522
RTLConsts.SUKGallonsDescription Constant 7522
RTLConsts.SUKGillsDescription Constant 7522
RTLConsts.SUKOuncesDescription Constant 7523
RTLConsts.SUKPecksDescription Constant 7523
RTLConsts.SUKPintsDescription Constant 7523
RTLConsts.SUKPottlesDescription Constant 7524
RTLConsts.SUKQuartsDescription Constant 7524
RTLConsts.SUnknownGroup Constant 7524
RTLConsts.SUnknownProperty Constant 7525
RTLConsts.SVolumeDescription Constant 7525
RTLConsts.SWeeksDescription Constant 7525
RTLConsts.sWindowsSocketError Constant 7526
RTLConsts.SWriteError Constant 7526
RTLConsts.SYardsDescription Constant 7526
RTLConsts.SYearsDescription Constant 7527

Index a

1 ExptIntf Namespace

This is namespace ExptIntf.

Topics

Name	Description
Classes (see page 3)	The following table lists classes in this documentation.
Functions (see page 7)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 8)	The following table lists structs, records, enums in this documentation.
Types (see page 9)	The following table lists types in this documentation.
Variables (see page 11)	The following table lists variables in this documentation.
Constants (see page 12)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIExpert (see page 3)	This is class ExptIntf.TIExpert.

Constants

Name	Description
ExpertEntryPoint (see page 12)	This is constant ExptIntf.ExpertEntryPoint.
isExperts (see page 12)	This is constant ExptIntf.isExperts.
ValidExpertVersion (see page 12)	This is constant ExptIntf.ValidExpertVersion.

Functions

	Name	Description
	RegisterLibraryExpert (see page 7)	Registers an expert so that it is installed when a package is installed.

Structs, Records, Enums

	Name	Description
	TExpertStyle (see page 8)	This is record ExptIntf.TExpertStyle.

Types

Name	Description
TExpertInitProc (see page 9)	This is type ExptIntf.TExpertInitProc.
TExpertRegisterProc (see page 9)	This is type ExptIntf.TExpertRegisterProc.
TExpertState (see page 10)	This is type ExptIntf.TExpertState.
TExpertTerminateProc (see page 10)	This is type ExptIntf.TExpertTerminateProc.

Variables

Name	Description
LibraryExpertProc (see page 11)	This is variable ExptIntf.LibraryExpertProc.
ToolServices (see page 11)	This is variable ExptIntf.ToolServices.

1.1 Classes

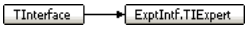
The following table lists classes in this documentation.

Classes

	Name	Description
	TIExpert (see page 3)	This is class ExptIntf.TIExpert.

1.1.1 TIExpert Class

Class Hierarchy



Pascal

```
TIExpert = class(TInterface);
```

C++

```
class TIExpert : public TInterface;
```

File

ExptIntf

Description

This is class ExptIntf.TIExpert.

1.1.1.1 TIExpert Members

The following tables list the members exposed by TIExpert.

Public Methods

	Name	Description
	Execute (see page 4)	This is Execute, a member of class TIExpert.
	GetAuthor (see page 4)	This is GetAuthor, a member of class TIExpert.
	GetComment (see page 4)	This is GetComment, a member of class TIExpert.
	GetGlyph (see page 4)	This is GetGlyph, a member of class TIExpert.
	GetIDString (see page 5)	This is GetIDString, a member of class TIExpert.
	GetMenuText (see page 5)	This is GetMenuText, a member of class TIExpert.
	GetName (see page 5)	This is GetName, a member of class TIExpert.
	GetPage (see page 5)	This is GetPage, a member of class TIExpert.
	GetState (see page 6)	This is GetState, a member of class TIExpert.
	GetStyle (see page 6)	This is GetStyle, a member of class TIExpert.

1.1.1.2 TIExpert Methods

The methods of the TIExpert class are listed here.

Public Methods

	Name	Description
	Execute (see page 4)	This is Execute, a member of class TIExpert.
	GetAuthor (see page 4)	This is GetAuthor, a member of class TIExpert.
	GetComment (see page 4)	This is GetComment, a member of class TIExpert.
	GetGlyph (see page 4)	This is GetGlyph, a member of class TIExpert.
	GetIDString (see page 5)	This is GetIDString, a member of class TIExpert.
	GetMenuText (see page 5)	This is GetMenuText, a member of class TIExpert.
	GetName (see page 5)	This is GetName, a member of class TIExpert.
	GetPage (see page 5)	This is GetPage, a member of class TIExpert.
	GetState (see page 6)	This is GetState, a member of class TIExpert.
	GetStyle (see page 6)	This is GetStyle, a member of class TIExpert.

1.1.1.2.1 TIExpert.Execute Method

Pascal

```
procedure Execute; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall Execute() = 0;
```

Description

This is Execute, a member of class TIExpert.

1.1.1.2.2 TIExpert.GetAuthor Method

Pascal

```
function GetAuthor: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetAuthor() = 0;
```

Description

This is GetAuthor, a member of class TIExpert.

1.1.1.2.3 TIExpert.GetComment Method

Pascal

```
function GetComment: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetComment() = 0;
```

Description

This is GetComment, a member of class TIExpert.

1.1.1.2.4 TIExpert.GetGlyph Method

Pascal

```
function GetGlyph: HICON; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HICON GetGlyph() = 0;
```

Description

This is GetGlyph, a member of class TIExpert.

1.1.1.2.5 TIExpert.GetIDString Method

Pascal

```
function GetIDString: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetIDString() = 0;
```

Description

This is GetIDString, a member of class TIExpert.

1.1.1.2.6 TIExpert.GetMenuText Method

Pascal

```
function GetMenuText: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetMenuText() = 0;
```

Description

This is GetMenuText, a member of class TIExpert.

1.1.1.2.7 TIExpert.GetName Method

Pascal

```
function GetName: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetName() = 0;
```

Description

This is GetName, a member of class TIExpert.

1.1.1.2.8 TIExpert.GetPage Method

Pascal

```
function GetPage: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall UnicodeString GetPage() = 0;
```

Description

This is GetPage, a member of class TIExpert.

1.1.1.2.9 TIExpert.GetState Method

Pascal

```
function GetState: TExpertState; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall TExpertState GetState() = 0;
```

Description

This is GetState, a member of class TIExpert.

1.1.1.2.10 TIExpert.GetStyle Method

Pascal

```
function GetStyle: TExpertStyle; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall TExpertStyle GetStyle() = 0;
```

Description

This is GetStyle, a member of class TIExpert.

1.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterLibraryExpert (see page 7)	Registers an expert so that it is installed when a package is installed.

1.2.1 ExptIntf.RegisterLibraryExpert Function

Registers an expert so that it is installed when a package is installed.

Pascal

```
procedure RegisterLibraryExpert(Expert: TIExpert);
```

C++

```
RegisterLibraryExpert(TIExpert Expert);
```

File

ExptIntf

Description

Call RegisterLibrary to register an expert for a package so that it appears in the New Items dialog. If the package already has a registered expert, RegisterLibraryExpert does nothing.

1.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TExpertStyle (see page 8)	This is record ExptIntf.TExpertStyle.

1.3.1 ExptIntf.TExpertStyle Enumeration

Pascal

```
TExpertStyle = (
  esStandard,
  esForm,
  esProject,
  esAddIn
);
```

C++

```
enum TExpertStyle {
  esStandard,
  esForm,
  esProject,
  esAddIn
};
```

File

ExptIntf

Description

This is record ExptIntf.TExpertStyle.

1.4 Types

The following table lists types in this documentation.

Types

Name	Description
TExpertInitProc (see page 9)	This is type ExptIntf.TExpertInitProc.
TExpertRegisterProc (see page 9)	This is type ExptIntf.TExpertRegisterProc.
TExpertState (see page 10)	This is type ExptIntf.TExpertState.
TExpertTerminateProc (see page 10)	This is type ExptIntf.TExpertTerminateProc.

1.4.1 ExptIntf.TExpertInitProc Type

Pascal

```
TExpertInitProc = function (ToolServices: TIToolServices; RegisterProc: TExpertRegisterProc;
var Terminate: TExpertTerminateProc): Boolean stdcall;
```

C++

```
(ToolServices: TIToolServices; RegisterProc: TExpertRegisterProc; var Terminate:
TExpertTerminateProc): Boolean stdcall ( TExpertInitProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertInitProc.

1.4.2 ExptIntf.TExpertRegisterProc Type

Pascal

```
TExpertRegisterProc = function (Expert: TIExpert): Boolean;
```

C++

```
(Expert: TIExpert): Boolean ( TExpertRegisterProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertRegisterProc.

1.4.3 ExptIntf.TExpertState Type

Pascal

```
TExpertState = set of (esEnabled, esChecked);
```

C++

```
(esEnabled, esChecked) TExpertState;
```

File

ExptIntf

Description

This is type ExptIntf.TExpertState.

1.4.4 ExptIntf.TExpertTerminateProc Type

Pascal

```
TExpertTerminateProc = procedure;
```

C++

```
( TExpertTerminateProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertTerminateProc.

1.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
LibraryExpertProc (see page 11)	This is variable ExptIntf.LibraryExpertProc.
ToolServices (see page 11)	This is variable ExptIntf.ToolServices.

1.5.1 ExptIntf.LibraryExpertProc Variable

Pascal

```
LibraryExpertProc: TExpertRegisterProc = nil;
```

C++

```
TExpertRegisterProc LibraryExpertProc = nil;
```

File

ExptIntf

Description

This is variable ExptIntf.LibraryExpertProc.

1.5.2 ExptIntf.ToolServices Variable

Pascal

```
ToolServices: TIToolServices = nil;
```

C++

```
TIToolServices ToolServices = nil;
```

File

ExptIntf

Description

This is variable ExptIntf.ToolServices.

1.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ExpertEntryPoint (see page 12)	This is constant ExptIntf.ExpertEntryPoint.
isExperts (see page 12)	This is constant ExptIntf.isExperts.
ValidExpertVersion (see page 12)	This is constant ExptIntf.ValidExpertVersion.

1.6.1 ExptIntf.ExpertEntryPoint Constant

Pascal

```
ExpertEntryPoint = 'INITEXPERT0017';
```

C++

```
ExpertEntryPoint = 'INITEXPERT0017';
```

File

ExptIntf

Description

This is constant ExptIntf.ExpertEntryPoint.

1.6.2 ExptIntf.isExperts Constant

Pascal

```
isExperts = 'Experts';
```

C++

```
isExperts = 'Experts';
```

File

ExptIntf

Description

This is constant ExptIntf.isExperts.

1.6.3 ExptIntf.ValidExpertVersion Constant

Pascal

```
ValidExpertVersion = 3;
```

C++

```
ValidExpertVersion = 3;
```

File

ExptIntf

Description

This is constant ExptIntf.ValidExpertVersion.

2 ExtActns Namespace

This is namespace ExtActns.

Topics

Name	Description
Classes (see page 17)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 157)	The following table lists structs, records, enums in this documentation.
Types (see page 158)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomFileRun (see page 17)	TCustomFileRun is the base class for actions that perform a specified operation on a file.
	TFileRun (see page 27)	TFileRun is the standard action for performing a specified operation on a file.
	TListControlAction (see page 37)	TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).
	TListControlClearSelection (see page 41)	TListControlClearSelection deselects all selected items in a list control (list view or list box).
	TListControlCopySelection (see page 50)	TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlDeleteSelection (see page 60)	TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).
	TListControlMoveSelection (see page 68)	TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlSelectAll (see page 71)	TListControlSelectAll selects all the items in a list control (list view or listbox).
	TNextTab (see page 80)	TNextTab moves a tabbed control to its next page.
	TOpenPicture (see page 92)	TOpenPicture is the standard action for displaying the open picture dialog.
	TPreviousTab (see page 101)	TPreviousTab moves a page control to its previous page.
	TRichEditAction (see page 112)	TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.
	TRichEditAlignCenter (see page 115)	TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.
	TRichEditAlignLeft (see page 117)	TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.
	TRichEditAlignRight (see page 119)	TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.
	TRichEditBold (see page 121)	TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.

	TRichEditBullets (see page 123)	TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.
	TRichEditItalic (see page 125)	TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.
	TRichEditStrikeOut (see page 127)	TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.
	TRichEditUnderline (see page 130)	TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.
	TSavePicture (see page 132)	TSavePicture is the standard action for displaying the save picture dialog.
	TSendMail (see page 141)	TSendMail is the standard action for sending an email message.
	TTabAction (see page 150)	TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.

Structs, Records, Enums

	Name	Description
	TShowCmd (see page 157)	This is record ExtActns.TShowCmd.

Types

Name	Description
TValidateTabEvent (see page 158)	This is type ExtActns.TValidateTabEvent.

2.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomFileRun (see page 17)	TCustomFileRun is the base class for actions that perform a specified operation on a file.
	TFileRun (see page 27)	TFileRun is the standard action for performing a specified operation on a file.
	TListControlAction (see page 37)	TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).
	TListControlClearSelection (see page 41)	TListControlClearSelection deselects all selected items in a list control (list view or list box).
	TListControlCopySelection (see page 50)	TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlDeleteSelection (see page 60)	TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).
	TListControlMoveSelection (see page 68)	TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlSelectAll (see page 71)	TListControlSelectAll selects all the items in a list control (list view or listbox).
	TNextTab (see page 80)	TNextTab moves a tabbed control to its next page.
	TOpenPicture (see page 92)	TOpenPicture is the standard action for displaying the open picture dialog.
	TPreviousTab (see page 101)	TPreviousTab moves a page control to its previous page.
	TRichEditAction (see page 112)	TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.
	TRichEditAlignCenter (see page 115)	TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.
	TRichEditAlignLeft (see page 117)	TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.
	TRichEditAlignRight (see page 119)	TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.
	TRichEditBold (see page 121)	TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.
	TRichEditBullets (see page 123)	TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.
	TRichEditItalic (see page 125)	TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.
	TRichEditStrikeOut (see page 127)	TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.
	TRichEditUnderline (see page 130)	TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.
	TSavePicture (see page 132)	TSavePicture is the standard action for displaying the save picture dialog.
	TSendMail (see page 141)	TSendMail is the standard action for sending an email message.
	TTabAction (see page 150)	TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.

2.1.1 TCustomFileRun Class

TCustomFileRun is the base class for actions that perform a specified operation on a file.

Class Hierarchy

Pascal

```
TCustomFileRun = class(TCustomAction);
```

C++

```
class TCustomFileRun : public TCustomAction;
```

File

ExtActns

Description

Use TCustomFileRun as a base class when defining a pre-defined action that uses ShellExecute to perform an operation on a file. TCustomFileRun implements properties for specifying

A target file or the option of letting the user select one using a browse dialog.

The operation to perform on the target file.

Parameters (see page 25) to supply to an external application that performs the desired operation.

The startup directory for the external application.

The state of the window that appears when performing the specified operation.

When the ExecuteTarget (see page 20) method is called, TCustomFileRun uses these properties to call the Windows API ShellExecute, which performs the specified operation if it is listed as applicable to the target file type in the system registry. Depending on the operation, this may involve launching an external application, calling a dynamic-link library (DLL), or displaying a system dialog (such as the file properties dialog).

Do not add TCustomFileRun to an action list or action manager. To add a generic file run action, use TFileRun (see page 27) instead.

See Also

TFileRun (see page 27)

2.1.1.1 TCustomFileRun Members

The following tables list the members exposed by TCustomFileRun.

Protected Methods

	Name	Description
	Notification (see page 21)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 19)	Creates an instance of TCustomFileRun (see page 17).
	ExecuteTarget (see page 20)	Performs the specified operation on a target file.
	HandlesTarget (see page 20)	Indicates whether the action can execute.
	SetupBrowseDialog (see page 21)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 23) property.
	UpdateTarget (see page 21)	Enables or disables the action, depending on current conditions.

Public Properties

	Name	Description
	Browse (see page 22)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 23)	Specifies the dialog that allows users to select the target file to run.

	Directory (see page 23)	Specifies the default directory for the externally-launched application.
	FileName (see page 23)	Specifies the target file to run.
R	HInst (see page 24)	Indicates the Windows error code that is returned when TCustomFileRun (see page 17) launches the external application.
	Operation (see page 25)	Indicates what TCustomFileRun (see page 17) should do with the target file.
	Parameters (see page 25)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 26)	This is ParentControl, a member of class TCustomFileRun.
	ShowCmd (see page 26)	Specifies how the launched application window appears.

2.1.1.2 TCustomFileRun.Create Constructor

Creates an instance of TCustomFileRun (see page 17).

Pascal

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

C++

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

Description

Most applications do not need to call the TCustomFileRun (see page 17) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomFileRun (see page 17) instance.

Create initializes the action's properties and creates a default TOpenDialog instance to act as the value of the BrowseDlg (see page 23) property.

See Also

Owner

SetupBrowseDialog (see page 21)

BrowseDlg (see page 23)

2.1.1.3 TCustomFileRun Methods

The methods of the TCustomFileRun class are listed here.

Protected Methods

	Name	Description
V	Notification (see page 21)	Responds when components are created or destroyed.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20)	Performs the specified operation on a target file.
V	HandlesTarget (see page 20)	Indicates whether the action can execute.
	SetupBrowseDialog (see page 21)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 23) property.
V	UpdateTarget (see page 21)	Enables or disables the action, depending on current conditions.

2.1.1.3.1 TCustomFileRun.ExecuteTarget Method

Performs the specified operation on a target file.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and performs the operation specified by the Operation (see page 25) property on the target file specified by the FileName (see page 23) property or chosen by the user if the value of the Browse (see page 22) property is true. The values of Parameters (see page 25), ShowCmd (see page 26), and Directory (see page 23), as well as the window handle for ParentControl (see page 26) are passed to Windows with the request to perform the specified operation.

Note: If the attempt to perform the operation fails, the HInst (see page 24) property contains an error code that indicates what was wrong. If the attempt succeeds, the HInst (see page 24) property is greater than 32.

See Also

FileName (see page 23)

Browse (see page 22)

Operation (see page 25)

Parameters (see page 25)

ShowCmd (see page 26)

Directory (see page 23)

parentcontrol (see page 26)

HInst (see page 24)

2.1.1.3.2 TCustomFileRun.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget always returns true, because TCustomFileRun (see page 17) does not use the target object.

2.1.1.3.3 TCustomFileRun.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

2.1.1.3.4 TCustomFileRun.SetupBrowseDialog Method

Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 23) property.

Pascal

```
procedure SetupBrowseDialog;
```

C++

```
__fastcall SetupBrowseDialog();
```

Description

SetupBrowseDialog generates the default open dialog instance for the dialog that appears when the Browse (see page 22) property is true. It assigns this newly created dialog component as the value of the BrowseDlg (see page 23) property.

Most applications do not need to call SetupBrowseDialog because it is called automatically by the TCustomFileRun (see page 17) constructor.

See Also

BrowseDlg (see page 23)

TOpenDialog

Browse (see page 22)

2.1.1.3.5 TCustomFileRun.UpdateTarget Method

Enables or disables the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when application is idle so that the action can update itself to reflect current conditions.

In TCustomFileRun (see page 17), UpdateTarget checks whether it can obtain a target file. If the FileName (see page 23) property specifies a target file, or if the Browse (see page 22) property is true, indicating that users can select a target file, then UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

FileName (see page 23)

Browse (see page 22)

2.1.1.4 TCustomFileRun Properties

The properties of the TCustomFileRun class are listed here.

Public Properties

	Name	Description
	Browse (see page 22)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 23)	Specifies the dialog that allows users to select the target file to run.
	Directory (see page 23)	Specifies the default directory for the externally-launched application.
	FileName (see page 23)	Specifies the target file to run.
R	HInst (see page 24)	Indicates the Windows error code that is returned when TCustomFileRun (see page 17) launches the external application.
	Operation (see page 25)	Indicates what TCustomFileRun (see page 17) should do with the target file.
	Parameters (see page 25)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 26)	This is ParentControl, a member of class TCustomFileRun.
	ShowCmd (see page 26)	Specifies how the launched application window appears.

2.1.1.4.1 TCustomFileRun.Browse Property

Controls whether an open dialog automatically appears to let users browse for the target file.

Pascal

```
property Browse: Boolean;
```

C++

```
__property Boolean Browse;
```

Description

Browse controls whether TCustomFileRun (see page 17) displays the dialog specified by BrowseDlg (see page 23) when it fires, letting users select the target file to be launched. When Browse is true, firing TCustomFileRun (see page 17) displays the open dialog specified by BrowseDlg (see page 23), and then (if the user does not cancel) performs the operation specified by Operation (see page 25) on the chosen file. When Browse is false, your application must specify the target file by explicitly setting the FileName (see page 23) property.

Note: When Browse is true, TCustomFileRun (see page 17) ignores the FileName (see page 23) property.

See Also

BrowseDlg (see page 23)

FileName (see page 23)

Operation (see page 25)

2.1.1.4.2 TCustomFileRun.BrowseDlg Property

Specifies the dialog that allows users to select the target file to run.

Pascal

```
property BrowseDlg: TOpenDialog;
```

C++

```
__property TOpenDialog BrowseDlg;
```

Description

Use BrowseDlg when the Browse (see page 22) property is true to configure the dialog that users use to select a target file. By expanding BrowseDlg in the Object Inspector at design time, or by setting its properties at runtime, you can customize the way the dialog first appears when the action fires.

By default, BrowseDlg is an internally-generated TOpenDialog object. When using the default dialog, there is no need to explicitly add a dialog component to your application. If you want to use a specialized TOpenDialog descendant (such as TOpenPictureDialog (see page 1560)), however, set BrowseDlg to an open dialog component you place in the same form or data module as the TCustomFileRun (see page 17) action.

See Also

Browse (see page 22)

FileName (see page 23)

TOpenDialog

2.1.1.4.3 TCustomFileRun.Directory Property

Specifies the default directory for the externally-launched application.

Pascal

```
property Directory: TFileName;
```

C++

```
__property TFileName Directory;
```

Description

Set Directory to specify a default (startup) directory for the external application that is launched when the action fires.

See Also

Parameters (see page 25)

FileName (see page 23)

2.1.1.4.4 TCustomFileRun.FileName Property

Specifies the target file to run.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file on which TCustomFileRun (see page 17) performs the operation specified by Operation (see page 25) when the action fires. FileName can specify a file that is opened using an external application (such as a .txt file that causes a text editor to launch), or it can specify an executable file for the application itself. When the action fires in response to a user action, TCustomFileRun (see page 17) launches an external application or dll to perform Operation (see page 25) on the file specified by FileName.

Note:

See Also

Browse (see page 22)

BrowseDlg (see page 23)

Parameters (see page 25)

Operation (see page 25)

2.1.1.4.5 TCustomFileRun.HInst Property

Indicates the Windows error code that is returned when TCustomFileRun (see page 17) launches the external application.

Pascal

```
property HInst: HInst;
```

C++

```
__property HInst HInst;
```

Description

TCustomFileRun (see page 17) sets HInst to the Windows return value when it launches the external application. If this value is less than 32, it represents an error code that indicates why the application failed to launch. This value is only available after the action has executed.

The following table lists the possible error values (defined in shellapi.pas) that can be the value of HInst when it is less than 32.

Value	Meaning
0	The operating system is out of memory or resources.
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.
ERROR_BAD_FORMAT	The .exe file is invalid (does not have the correct internal format).
SE_ERR_ACCESSDENIED	The operating system denied access to the specified file.
SE_ERR_ASSOCINCOMPLETE	The file name association is incomplete or invalid.
SE_ERR_DDEBUSY	A DDE transaction could not be completed because other DDE transactions were being processed.
SE_ERR_DDEFAIL	A DDE transaction failed.
SE_ERR_DDETIMEOUT	A DDE transaction could not be completed because the request timed out.

SE_ERR_DLLNOTFOUND	A specified dynamic-link library was not found.
SE_ERR_FNF	The specified file was not found.
SE_ERR_NOASSOC	There is no application associated with the given file name extension or TCustomFileRun (see page 17) attempted to print an unprintable file.
SE_ERR_OOM	There was not enough memory to complete the operation.
SE_ERR_PNF	The specified path was not found.
SE_ERR_SHARE	A sharing violation occurred.

2.1.1.4.6 TCustomFileRun.Operation Property

Indicates what TCustomFileRun (see page 17) should do with the target file.

Pascal

```
property Operation: String;
```

C++

```
__property UnicodeString Operation;
```

Description

Set Operation to indicate what should be done with the target file when TCustomFileRun (see page 17) fires. The possible values for Operation depend on the particular file or folder specified by FileName (see page 23) or selected by the user when Browse (see page 22) is true. The possible operations for a particular file or folder are listed in the system Registry (see page 6599).

The following strings are usually valid:

Value	Meaning
edit	TCustomFileRun (see page 17) opens an editor. If FileName (see page 23) is not an editable file, the action fails.
explore	TCustomFileRun (see page 17) launches the Windows explorer for the folder specified by FileName (see page 23).
open	TCustomFileRun (see page 17) opens the file specified by FileName (see page 23). FileName (see page 23) can be an executable file, a file associated with an application that can open it, or a folder.
print	TCustomFileRun (see page 17) prints the file specified by FileName (see page 23). If FileName (see page 23) is a binary file (for example, if it is an executable), the action fails.
properties	TCustomFileRun (see page 17) displays the file or folder's properties in a Properties dialog.

Note: If Operation is an empty string, Windows tries to use the "open" operation. If the specified file does not support the "open" operation, Windows tries to use the default operation. If there is no "open" operation and no default operation, and if the application is running on Windows 2000 or later, Windows uses the first verb listed in the Registry (see page 6599).

See Also

FileName (see page 23)

Browse (see page 22)

2.1.1.4.7 TCustomFileRun.Parameters Property

Specifies parameters to be passed to the application when the target file is an executable.

Pascal

```
property Parameters: String;
```

C++

```
__property UnicodeString Parameters;
```

Description

Use Parameters to specify any parameters that should be passed to the external application when it is launched. Only set Parameters if FileName (see page 23) is the name of an Executable file or the user is browsing for an executable file. If the target file holds data associated with the application to launch, Parameters should be an empty string.

See Also

FileName (see page 23)

Operation (see page 25)

2.1.1.4.8 TCustomFileRun.ParentControl Property

Pascal

```
property ParentControl: TWinControl;
```

C++

```
__property TWinControl * ParentControl;
```

Description

This is ParentControl, a member of class TCustomFileRun.

2.1.1.4.9 TCustomFileRun.ShowCmd Property

Specifies how the launched application window appears.

Pascal

```
property ShowCmd: TShowCmd;
```

C++

```
__property TShowCmd ShowCmd;
```

Description

Use ShowCmd to indicate how you want the application window to appear when TCustomFileRun (see page 17) performs the specified operation on its target file. ShowCmd can have any of the following values:

Value	Meaning
scHide	The external application window is hidden. If it was active, another window is activated.
scMaximize	The external application window is maximized (but not necessarily activated)
scMinimize	The external application window is minimized (but not necessarily activated).
scRestore	The external application window is activated. If it was minimized or maximized, Windows restores it to its original size and position.
scShow	The external application window is activated in its current size and position.
scShowDefault	The external application window is shown in the default startup state specified by that external application.
scShowMaximized	The external application window is activated and maximized.

scShowMinimized	The external application window is activated and minimized.
scShowMinNoActive	The external application window is minimized. The active window does not change.
scShowNA	The external application window is displayed (after being hidden) but activation does not change.
scShowNoActivate	The external application window is displayed in its most recent size and position, but activation does not change.
scShowNormal	The external application window is activated and restored to its original size and position if it was maximized or minimized.

See Also

Operation (see page 25)

2.1.2 TFileRun Class

TFileRun is the standard action for performing a specified operation on a file.

Class Hierarchy

Pascal

```
TFileRun = class(TCustomFileRun);
```

C++

```
class TFileRun : public TCustomFileRun;
```

File

ExtActns

Description

Add TFileRun to an action list to let your application launch an external application or dll in response to user actions. Controls linked to this action cause your application to perform the operation specified by the Operation (see page 33) property on the file specified by the FileName (see page 31) property.

Before this action fires, configure its properties to indicate what application to launch and how to launch it:

To indicate the target file to run, set the FileName (see page 31) property to the name of a file on which the external application operates or to the name of the application's executable file. If you want to let the user select the file name instead when the action fires, set the Browse (see page 29) property to true, and an open dialog appears when the action fires.

To indicate what action to perform on the specified file, set the Operation (see page 33) property.

To supply parameters to the external application when it is launched, use the Parameters (see page 33) property.

To specify a default directory for the external application, use the Directory (see page 30) property.

To control how the external application's window appears (minimized, maximized, hidden, and so on), use the ShowCmd (see page 35) property.

If the external application needs a parent window (to parent a dialog such as a properties dialog), or if you want to allow TFileRun to display error messages, use the ParentControl (see page 34) property.

See Also

TBrowseURL

TSendMail (see page 141)

TFileOpenWith

TDownloadURL

2.1.2.1 TFileRun Members

The following tables list the members exposed by TFileRun.

Protected Methods

	Name	Description
	Notification (see page 21)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 19)	Creates an instance of TCustomFileRun (see page 17).
	ExecuteTarget (see page 20)	Performs the specified operation on a target file.
	HandlesTarget (see page 20)	Indicates whether the action can execute.
	SetupBrowseDialog (see page 21)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 23) property.
	UpdateTarget (see page 21)	Enables or disables the action, depending on current conditions.

Public Properties

	Name	Description
	HInst (see page 24)	Indicates the Windows error code that is returned when TCustomFileRun (see page 17) launches the external application.

Published Events

	Name	Description
	OnHint (see page 36)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 36)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Browse (see page 29)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 29)	Specifies the dialog that allows users to select the target file to run.
	Caption (see page 30)	Represents the caption of client controls and menu items.
	Directory (see page 30)	Specifies the default directory for the externally-launched application.
	Enabled (see page 31)	Indicates whether client controls and menu items are enabled.
	FileName (see page 31)	Specifies the target file to run.
	HelpContext (see page 31)	Indicates the help context ID for client controls and menu items.
	Hint (see page 32)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 32)	Indicates the ImageIndex property for client controls and menu items.
	Operation (see page 33)	Indicates what TCustomFileRun should do with the target file.
	Parameters (see page 33)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 34)	This is ParentControl, a member of class TFileRun.
	SecondaryShortCuts (see page 34)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 34)	Specifies the ShortCut property for client menu items.
	ShowCmd (see page 35)	Specifies how the launched application window appears.
	Visible (see page 35)	Specifies the Visible property for client controls and menu items.

2.1.2.2 TFileRun Properties

The properties of the TFileRun class are listed here.

Published Properties

	Name	Description
	Browse (see page 29)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 29)	Specifies the dialog that allows users to select the target file to run.
	Caption (see page 30)	Represents the caption of client controls and menu items.
	Directory (see page 30)	Specifies the default directory for the externally-launched application.
	Enabled (see page 31)	Indicates whether client controls and menu items are enabled.
	FileName (see page 31)	Specifies the target file to run.
	HelpContext (see page 31)	Indicates the help context ID for client controls and menu items.
	Hint (see page 32)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 32)	Indicates the ImageIndex property for client controls and menu items.
	Operation (see page 33)	Indicates what TCustomFileRun should do with the target file.
	Parameters (see page 33)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 34)	This is ParentControl, a member of class TFileRun.
	SecondaryShortCuts (see page 34)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 34)	Specifies the ShortCut property for client menu items.
	ShowCmd (see page 35)	Specifies how the launched application window appears.
	Visible (see page 35)	Specifies the Visible property for client controls and menu items.

2.1.2.2.1 TFileRun.Browse Property

Controls whether an open dialog automatically appears to let users browse for the target file.

Pascal

```
property Browse: Boolean;
```

C++

```
__property Boolean Browse;
```

Description

Browse controls whether TCustomFileRun displays the dialog specified by BrowseDlg when it fires, letting users select the target file to be launched. When Browse is true, firing TCustomFileRun displays the open dialog specified by BrowseDlg, and then (if the user does not cancel) performs the operation specified by Operation on the chosen file. When Browse is false, your application must specify the target file by explicitly setting the FileName property.

Note: When Browse is true, TCustomFileRun ignores the FileName property.

See Also

BrowseDlg (see page 23)

FileName (see page 23)

Operation (see page 25)

2.1.2.2.2 TFileRun.BrowseDlg Property

Specifies the dialog that allows users to select the target file to run.

Pascal

```
property BrowseDlg: TOpenDialog;
```

C++

```
__property TOpenDialog BrowseDlg;
```

Description

Use BrowseDlg when the Browse property is true to configure the dialog that users use to select a target file. By expanding BrowseDlg in the Object Inspector at design time, or by setting its properties at runtime, you can customize the way the dialog first appears when the action fires.

By default, BrowseDlg is an internally-generated TOpenDialog object. When using the default dialog, there is no need to explicitly add a dialog component to your application. If you want to use a specialized TOpenDialog descendant (such as TOpenPictureDialog), however, set BrowseDlg to an open dialog component you place in the same form or data module as the TCustomFileRun action.

See Also

Browse (see page 22)

FileName (see page 23)

TOpenDialog

2.1.2.2.3 TFileRun.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.2.2.4 TFileRun.Directory Property

Specifies the default directory for the externally-launched application.

Pascal

```
property Directory: TFileName;
```

C++

```
__property TFileName Directory;
```

Description

Set Directory to specify a default (startup) directory for the external application that is launched when the action fires.

See Also

Parameters (see page 25)

FileName (see page 23)

2.1.2.2.5 TFileRun.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.2.2.6 TFileRun.FileName Property

Specifies the target file to run.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file on which TCustomFileRun performs the operation specified by Operation when the action fires. FileName can specify a file that is opened using an external application (such as a .txt file that causes a text editor to launch), or it can specify an executable file for the application itself. When the action fires in response to a user action, TCustomFileRun launches an external application or dll to perform Operation on the file specified by FileName.

Note:

See Also

Browse (see page 22)

BrowseDlg (see page 23)

Parameters (see page 25)

Operation (see page 25)

2.1.2.2.7 TFileRun.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.2.2.8 TFileRun.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.2.2.9 TFileRun.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.2.2.10 TFileRun.Operation Property

Indicates what TCustomFileRun should do with the target file.

Pascal

```
property Operation: String;
```

C++

```
__property UnicodeString Operation;
```

Description

Set Operation to indicate what should be done with the target file when TCustomFileRun fires. The possible values for Operation depend on the particular file or folder specified by FileName or selected by the user when Browse is true. The possible operations for a particular file or folder are listed in the system Registry.

The following strings are usually valid:

Value	Meaning
edit	TCustomFileRun opens an editor. If FileName is not an editable file, the action fails.
explore	TCustomFileRun launches the Windows explorer for the folder specified by FileName.
open	TCustomFileRun opens the file specified by FileName. FileName can be an executable file, a file associated with an application that can open it, or a folder.
print	TCustomFileRun prints the file specified by FileName. If FileName is a binary file (for example, if it is an executable), the action fails.
properties	TCustomFileRun displays the file or folder's properties in a Properties dialog.

Note: If Operation is an empty string, Windows tries to use the "open" operation. If the specified file does not support the "open" operation, Windows tries to use the default operation. If there is no "open" operation and no default operation, and if the application is running on Windows 2000 or later, Windows uses the first verb listed in the Registry.

See Also

FileName (see page 23)

Browse (see page 22)

2.1.2.2.11 TFileRun.Parameters Property

Specifies parameters to be passed to the application when the target file is an executable.

Pascal

```
property Parameters: String;
```

C++

```
__property UnicodeString Parameters;
```

Description

Use Parameters to specify any parameters that should be passed to the external application when it is launched. Only set Parameters if FileName is the name of an Executable file or the user is browsing for an executable file. If the target file holds data associated with the application to launch, Parameters should be an empty string.

See Also

FileName (see page 23)

Operation (see page 25)

2.1.2.2.12 TFileRun.ParentControl Property

Pascal

```
property ParentControl: TWinControl;
```

C++

```
__property TWinControl * ParentControl;
```

Description

This is ParentControl, a member of class TFileRun.

2.1.2.2.13 TFileRun.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.2.2.14 TFileRun.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.2.2.15 TFileRun.ShowCmd Property

Specifies how the launched application window appears.

Pascal

```
property ShowCmd: TShowCmd;
```

C++

```
__property TShowCmd ShowCmd;
```

Description

Use ShowCmd to indicate how you want the application window to appear when TCustomFileRun performs the specified operation on its target file. ShowCmd can have any of the following values:

Value	Meaning
scHide	The external application window is hidden. If it was active, another window is activated.
scMaximize	The external application window is maximized (but not necessarily activated)
scMinimize	The external application window is minimized (but not necessarily activated).
scRestore	The external application window is activated. If it was minimized or maximized, Windows restores it to its original size and position.
scShow	The external application window is activated in its current size and position.
scShowDefault	The external application window is shown in the default startup state specified by that external application.
scShowMaximized	The external application window is activated and maximized.
scShowMinimized	The external application window is activated and minimized.
scShowMinNoActive	The external application window is minimized. The active window does not change.
scShowNA	The external application window is displayed (after being hidden) but activation does not change.
scShowNoActivate	The external application window is displayed in its most recent size and position, but activation does not change.
scShowNormal	The external application window is activated and restored to its original size and position if it was maximized or minimized.

See Also

Operation (see page 25)

2.1.2.2.16 TFileRun.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.2.3 TFileRun Events

The events of the TFileRun class are listed here.

Published Events

	Name	Description
	OnHint (see page 36)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 36)	Occurs when the application is idle or when the action list updates.

2.1.2.3.1 TFileRun.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.2.3.2 TFileRun.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
```

```
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.3 TListControlAction Class

TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).

Class Hierarchy

Pascal

```
TListControlAction = class(TCustomAction);
```

C++

```
class TListControlAction : public TCustomAction;
```

File

ExtActns

Description

TListControlAction implements the common behavior for actions that work with list controls (list boxes and list views). Each descendant class performs a specific operation on the target list control. Do not use TListControlAction directly in an application. Instead, use a TListControlAction descendant such as TListControlSelectAll (see page 71), TListControlClearSelection (see page 41), TListControlDeleteSelection (see page 60), TListControlCopySelection (see page 50), or TListControlMoveSelection (see page 68).

TListControlAction makes the target list control available to descendant classes as the value of the ListControl (see page 40) property. TListControlAction descendants then override the ExecuteTarget method to perform their specific function on the target list control.

See Also

TListControlClearSelection (see page 41)

TListControlCopySelection (see page 50)

TListControlMoveSelection (see page 68)

TListCOntrolDeleteSelection

TListControlSelectAll (see page 71)

TCustomListControl

2.1.3.1 TListControlAction Members

The following tables list the members exposed by TListControlAction.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
V	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
V	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
V	HandlesTarget (see page 39)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
	ListControl (see page 40)	Provides access to the target of the action.

2.1.3.2 TListControlAction.Create Constructor

Creates an instance of TListControlAction (see page 37).

Pascal

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

C++

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

Description

Most applications do not need to call the TListControlAction (see page 37) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TListControlAction (see page 37) instance.

See Also

Owner

2.1.3.3 TListControlAction Methods

The methods of the TListControlAction class are listed here.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
V	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
V	HandlesTarget (see page 39)	Indicates whether the action can execute its function.

2.1.3.3.1 TListControlAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target is a TCustomListControl descendant (or that the ListControl (see page 40) property is set if Target is nil (Delphi) or NULL (C++)). HandlesTarget returns true if there is a list control to act as target, false otherwise.

See Also

ExecuteTarget

TCustomListControl

ListControl (see page 40)

2.1.3.3.2 TListControlAction.IsEnabled Method

Pascal

```
function IsEnabled(Target: TObject): Boolean;
```

C++

```
__fastcall Boolean IsEnabled(TObject * Target);
```

Description

This is IsEnabled, a member of class TListControlAction.

2.1.3.3.3 TListControlAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

2.1.3.4 TListControlAction Properties

The properties of the TListControlAction class are listed here.

Public Properties

	Name	Description
	ListControl (see page 40)	Provides access to the target of the action.

2.1.3.4.1 TListControlAction.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView

TListBox

2.1.4 TListControlClearSelection Class

TListControlClearSelection deselects all selected items in a list control (list view or list box).

Class Hierarchy

Pascal

```
TListControlClearSelection = class(TListControlAction);
```

C++

```
class TListControlClearSelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlClearSelection to an action list to allow users to deselect any selected items of a list control by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to deselect all selected items in the target control.

TListControlClearSelection is designed to work with a list control (TCustomListControl descendant) as a target. TListControlClearSelection automatically disables itself if the target control is not a list control or if that list box or list view has no selected items.

See Also

TListControlCopySelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl

2.1.4.1 TListControlClearSelection Members

The following tables list the members exposed by TListControlClearSelection.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
	ExecuteTarget (see page 42)	Deselects the currently selected items in the target list control.
	HandlesTarget (see page 39)	Indicates whether the action can execute its function.
	UpdateTarget (see page 43)	Enables or disables the action, depending on the state of the target control.

Published Events

	Name	Description
	OnHint (see page 48)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 48)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 44)	Represents the caption of client controls and menu items.
	Enabled (see page 44)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 44)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 45)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 45)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 46)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 46)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 46)	Provides access to the target of the action.
	SecondaryShortCuts (see page 47)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 47)	Specifies the ShortCut property for client menu items.
	Visible (see page 47)	Specifies the Visible property for client controls and menu items.

2.1.4.2 TListControlClearSelection Methods

The methods of the TListControlClearSelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 42)	Deselects the currently selected items in the target list control.
	UpdateTarget (see page 43)	Enables or disables the action, depending on the state of the target control.

2.1.4.2.1 TListControlClearSelection.ExecuteTarget Method

Deselects the currently selected items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 39) returns false if it is not).

ExecuteTarget sets the ListControl (see page 46) property to Target it is not already set. It then calls the list control's

ClearSelection method, deselecting any selected items.

See Also

TCustomListControl

ListControl (see page 40)

ClearSelection

2.1.4.2.2 TListControlClearSelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlClearSelection (see page 41), UpdateTarget checks whether the current target object (specified by the ListControl (see page 46) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 44) to true. If not, UpdateTarget sets Enabled (see page 44) to false.

See Also

Enabled

SelCount

ListControl (see page 40)

2.1.4.3 TListControlClearSelection Properties

The properties of the TListControlClearSelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 44)	Represents the caption of client controls and menu items.
	Enabled (see page 44)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 44)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 45)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 45)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 46)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 46)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 46)	Provides access to the target of the action.
	SecondaryShortCuts (see page 47)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 47)	Specifies the ShortCut property for client menu items.
	Visible (see page 47)	Specifies the Visible property for client controls and menu items.

2.1.4.3.1 TListControlClearSelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.4.3.2 TListControlClearSelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.4.3.3 TListControlClearSelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.4.3.4 TListControlClearSelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.4.3.5 TListControlClearSelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.4.3.6 TListControlClearSelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.4.3.7 TListControlClearSelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.4.3.8 TListControlClearSelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView

or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView

TListBox

2.1.4.3.9 TListControlClearSelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.4.3.10 TListControlClearSelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.4.3.11 TListControlClearSelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.4.4 TListControlClearSelection Events

The events of the TListControlClearSelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 48)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 48)	Occurs when the application is idle or when the action list updates.

2.1.4.4.1 TListControlClearSelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.4.4.2 TListControlClearSelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.5 TListControlCopySelection Class

TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.

Class Hierarchy

Pascal

```
TListControlCopySelection = class(TListControlAction);
```

C++

```
class TListControlCopySelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlCopySelection to an action list to allow users to copy the selected items of a list control to another list control by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to copy all selected items in the target control to another (destination) list control.

Your application must explicitly set the Destination (see page 53) property to indicate the list to which items are copied.

TListControlCopySelection is designed to work with a list control (TCustomListControl descendant) as a target. This target is the source list control, not the destination. TListControlCopySelection automatically disables itself if the target control is not a list control, if that list box or list view has no selected items, or if there is no destination list control assigned.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl

2.1.5.1 TListControlCopySelection Members

The following tables list the members exposed by TListControlCopySelection.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
V	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
V	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
V	ExecuteTarget (see page 51)	Copies the selected items in the target list control to the list control specified by Destination (see page 53).
V	HandlesTarget (see page 52)	Indicates whether the action can execute its function.

	UpdateTarget (see page 52)	Enables or disables the action, depending on the state of the target control.

Published Events

	Name	Description
	OnHint (see page 58)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 58)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 53)	Represents the caption of client controls and menu items.
	Destination (see page 53)	Specifies the list control to which items are copied.
	Enabled (see page 54)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 54)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 55)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 55)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 56)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 56)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 56)	Provides access to the target of the action.
	SecondaryShortCuts (see page 57)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 57)	Specifies the ShortCut property for client menu items.
	Visible (see page 57)	Specifies the Visible property for client controls and menu items.

2.1.5.2 TListControlCopySelection Methods

The methods of the TListControlCopySelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 51)	Copies the selected items in the target list control to the list control specified by Destination (see page 53).
	HandlesTarget (see page 52)	Indicates whether the action can execute its function.
	UpdateTarget (see page 52)	Enables or disables the action, depending on the state of the target control.

2.1.5.2.1 TListControlCopySelection.ExecuteTarget Method

Copies the selected items in the target list control to the list control specified by Destination (see page 53).

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 52) returns false if it is not).

ExecuteTarget sets the ListControl (see page 56) property to Target it is not already set. It then calls the list control's ClearSelection method, copying any selected items to the list box or list view specified by the Destination (see page 53) property.

See Also

TCustomListControl

ListControl (see page 40)

CopySelection

Destination

2.1.5.2.2 TListControlCopySelection.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target is a TCustomListControl descendant (or that the ListControl (see page 56) property is set if Target is nil (Delphi) or NULL (C++)). It also ensures that the Destination (see page 53) property has been asssigned. HandlesTarget returns true if there is a list control to act as target and a destination control, false otherwise.

See Also

ExecuteTarget

TCustomListControl

ListControl (see page 40)

Destination

2.1.5.2.3 TListControlCopySelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlCopySelection (see page 50), UpdateTarget checks whether the current target object (specified by the ListControl (see page 56) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 54) to true. If not, UpdateTarget sets Enabled (see page 54) to

false.

See Also

Enabled

SelCount

ListControl (see page 40)

2.1.5.3 TListControlCopySelection Properties

The properties of the TListControlCopySelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 53)	Represents the caption of client controls and menu items.
	Destination (see page 53)	Specifies the list control to which items are copied.
	Enabled (see page 54)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 54)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 55)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 55)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 56)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 56)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 56)	Provides access to the target of the action.
	SecondaryShortCuts (see page 57)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 57)	Specifies the ShortCut property for client menu items.
	Visible (see page 57)	Specifies the Visible property for client controls and menu items.

2.1.5.3.1 TListControlCopySelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.5.3.2 TListControlCopySelection.Destination Property

Specifies the list control to which items are copied.

Pascal

```
property Destination: TCustomListControl;
```

C++

```
__property TCustomListControl Destination;
```

Description

Destination is the list control to which the action copies selected items from its target. It is a TCustomListControl descendant, such a TListView or TListBox.

You must explicitly set Destination to indicate the list control that receives the new items. These items are copied from the list control specified by the ListControl (see page 56) property. Destination and ListControl (see page 56) do not need to be the same type of control, as long as they are both TCustomListControl descendants.

See Also

TListView

TListBox

ListControl (see page 40)

2.1.5.3.3 TListControlCopySelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.5.3.4 TListControlCopySelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.5.3.5 TListControlCopySelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.5.3.6 TListControlCopySelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.5.3.7 TListControlCopySelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.5.3.8 TListControlCopySelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.5.3.9 TListControlCopySelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView

or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView

TListBox

2.1.5.3.10 TListControlCopySelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.5.3.11 TListControlCopySelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.5.3.12 TListControlCopySelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.5.4 TListControlCopySelection Events

The events of the TListControlCopySelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 58)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 58)	Occurs when the application is idle or when the action list updates.

2.1.5.4.1 TListControlCopySelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.5.4.2 TListControlCopySelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.6 TListControlDeleteSelection Class

TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).

Class Hierarchy

Pascal

```
TListControlDeleteSelection = class(TListControlAction);
```

C++

```
class TListControlDeleteSelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlDeleteSelection to an action list to allow users to delete the selected items of a list control by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to delete all selected items in the target control (a listbox or list view).

TListControlDeleteSelection is designed to work with a list control (TCustomListControl descendant) as a target. TListControlDeleteSelection automatically disables itself if the target control is not a list control or if that list box or list view has no selected items.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl

2.1.6.1 TListControlDeleteSelection Members

The following tables list the members exposed by TListControlDeleteSelection.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
	ExecuteTarget (see page 61)	Deletes the selected items in the target list control.
	HandlesTarget (see page 39)	Indicates whether the action can execute its function.
	UpdateTarget (see page 62)	Enables or disables the action, depending on the state of the target control.

Published Events

	Name	Description
	OnHint (see page 67)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 67)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 62)	Represents the caption of client controls and menu items.
	Enabled (see page 63)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 63)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 64)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 64)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 65)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 65)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 65)	Provides access to the target of the action.
	SecondaryShortCuts (see page 66)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 66)	Specifies the ShortCut property for client menu items.
	Visible (see page 66)	Specifies the Visible property for client controls and menu items.

2.1.6.2 TListControlDeleteSelection Methods

The methods of the TListControlDeleteSelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 61)	Deletes the selected items in the target list control.
	UpdateTarget (see page 62)	Enables or disables the action, depending on the state of the target control.

2.1.6.2.1 TListControlDeleteSelection.ExecuteTarget Method

Deletes the selected items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 39) returns false if it is not).

ExecuteTarget sets the ListControl (see page 65) property to Target it is not already set. It then calls the list control's DeleteSelected method, deleting any selected items.

See Also

TCustomListControl

ListControl (see page 40)

DeleteSelected

2.1.6.2.2 TListControlDeleteSelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlDeleteSelection (see page 60), UpdateTarget checks whether the current target object (specified by the ListControl (see page 65) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 63) to true. If not, UpdateTarget sets Enabled (see page 63) to false.

See Also

Enabled

SelCount

ListControl (see page 40)

2.1.6.3 TListControlDeleteSelection Properties

The properties of the TListControlDeleteSelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 62)	Represents the caption of client controls and menu items.
	Enabled (see page 63)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 63)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 64)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 64)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 65)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 65)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 65)	Provides access to the target of the action.
	SecondaryShortCuts (see page 66)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 66)	Specifies the ShortCut property for client menu items.
	Visible (see page 66)	Specifies the Visible property for client controls and menu items.

2.1.6.3.1 TListControlDeleteSelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.6.3.2 TListControlDeleteSelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.6.3.3 TListControlDeleteSelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.6.3.4 TListControlDeleteSelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.6.3.5 TListControlDeleteSelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.6.3.6 TListControlDeleteSelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.6.3.7 TListControlDeleteSelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.6.3.8 TListControlDeleteSelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView

TListBox

2.1.6.3.9 TListControlDeleteSelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.6.3.10 TListControlDeleteSelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.6.3.11 TListControlDeleteSelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.6.4 TListControlDeleteSelection Events

The events of the TListControlDeleteSelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 67)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 67)	Occurs when the application is idle or when the action list updates.

2.1.6.4.1 TListControlDeleteSelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.6.4.2 TListControlDeleteSelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
```

```
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.7 TListControlMoveSelection Class

TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.

Class Hierarchy

Pascal

```
TListControlMoveSelection = class(TListControlCopySelection);
```

C++

```
class TListControlMoveSelection : public TListControlCopySelection;
```

File

ExtActns

Description

Add TListControlMoveSelection to an action list to allow users to move the selected items of a list control to another list control by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to move all selected items in the target control to another (destination) list control.

Your application must explicitly set the Destination (see page 53) property to indicate the list to which items are copied.

TListControlMoveSelection is designed to work with a list control (TCustomListControl descendant) as a target. This target is the source list control, not the destination. TListControlMoveSelection automatically disables itself if the target control is not a list control, if that list box or list view has no selected items, or if there is no destination list control assigned.

See Also

TListControlClearSelection

TListControlCopySelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl

2.1.7.1 TListControlMoveSelection Members

The following tables list the members exposed by TListControlMoveSelection.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
	ExecuteTarget (see page 70)	Moves the selected items in the target list control to the list control specified by Destination (see page 53).
	HandlesTarget (see page 39)	Indicates whether the action can execute its function.
	HandlesTarget (see page 52)	Indicates whether the action can execute its function.
	UpdateTarget (see page 52)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ListControl (see page 40)	Provides access to the target of the action.

Published Events

	Name	Description
	OnHint (see page 58)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 58)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 53)	Represents the caption of client controls and menu items.
	Destination (see page 53)	Specifies the list control to which items are copied.
	Enabled (see page 54)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 54)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 55)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 55)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 56)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 56)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 56)	Provides access to the target of the action.
	SecondaryShortCuts (see page 57)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 57)	Specifies the ShortCut property for client menu items.
	Visible (see page 57)	Specifies the Visible property for client controls and menu items.

2.1.7.2 TListControlMoveSelection Methods

The methods of the TListControlMoveSelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 70)	Moves the selected items in the target list control to the list control specified by Destination (see page 53).

2.1.7.2.1 TListControlMoveSelection.ExecuteTarget Method

Moves the selected items in the target list control to the list control specified by Destination (see page 53).

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 52) returns false if it is not).

ExecuteTarget sets the ListControl (see page 56) property to Target it is not already set. It then calls the list control's MoveSelection method, copying any selected items to the list box or list view specified by the Destination (see page 53) property and removing them from the target (source list control).

See Also

TCustomListControl

ListControl (see page 40)

MoveSelection

Destination

2.1.8 TListControlSelectAll Class

TListControlSelectAll selects all the items in a list control (list view or listbox).

Class Hierarchy

Pascal

```
TListControlSelectAll = class(TListControlAction);
```

C++

```
class TListControlSelectAll : public TListControlAction;
```

File

ExtActns

Description

Add TListControlSelectAll to an action list to allow users to select all the items of a list control by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to select all items in the target control (a listbox or list view).

TListControlSelectAll is designed to work with a list control (TCustomListControl descendant) as a target. TListControlSelectAll automatically disables itself if the target control is not a list control, if that target control does not permit multiple selection, or if all of the items in the target control are already selected.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlSelectAll

TListControlSelectAll

TCustomListControl

2.1.8.1 TListControlSelectAll Members

The following tables list the members exposed by TListControlSelectAll.

Protected Methods

	Name	Description
	IsEnabled (see page 40)	This is IsEnabled, a member of class TListControlAction.
	Notification (see page 40)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 38)	Creates an instance of TListControlAction (see page 37).
	ExecuteTarget (see page 72)	Selects all the items in the target list control.
	HandlesTarget (see page 39)	Indicates whether the action can execute its function.
	UpdateTarget (see page 73)	Enables or disables the action, depending on the state of the target control.

Published Events

	Name	Description
	OnHint (see page 78)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 78)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 74)	Represents the caption of client controls and menu items.
	Enabled (see page 74)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 74)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 75)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 75)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 76)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 76)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 76)	Provides access to the target of the action.
	SecondaryShortCuts (see page 77)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 77)	Specifies the ShortCut property for client menu items.
	Visible (see page 77)	Specifies the Visible property for client controls and menu items.

2.1.8.2 TListControlSelectAll Methods

The methods of the TListControlSelectAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 72)	Selects all the items in the target list control.
	UpdateTarget (see page 73)	Enables or disables the action, depending on the state of the target control.

2.1.8.2.1 TListControlSelectAll.ExecuteTarget Method

Selects all the items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 39) returns false if it is not).

ExecuteTarget sets the ListControl (see page 76) property to Target it is not already set. It then calls the list control's SelectAll

method, selecting all of its items.

See Also

TCustomListControl

ListControl (see page 40)

SelectAll

2.1.8.2.2 TListControlSelectAll.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlSelectAll (see page 71), UpdateTarget checks whether the current target object (specified by the ListControl (see page 76) property if it is set, otherwise by the Target parameter) is a list control that includes allows multiple selections and that includes at least one unselected item. If so, UpdateTarget sets Enabled (see page 74) to true. If not, UpdateTarget sets Enabled (see page 74) to false.

See Also

Enabled

MultiSelect

ListControl (see page 40)

2.1.8.3 TListControlSelectAll Properties

The properties of the TListControlSelectAll class are listed here.

Published Properties

	Name	Description
	Caption (see page 74)	Represents the caption of client controls and menu items.
	Enabled (see page 74)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 74)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 75)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 75)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 76)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 76)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 76)	Provides access to the target of the action.
	SecondaryShortCuts (see page 77)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 77)	Specifies the ShortCut property for client menu items.
	Visible (see page 77)	Specifies the Visible property for client controls and menu items.

2.1.8.3.1 TListControlSelectAll.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.8.3.2 TListControlSelectAll.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.8.3.3 TListControlSelectAll.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.8.3.4 TListControlSelectAll.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.8.3.5 TListControlSelectAll.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.8.3.6 TListControlSelectAll.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.8.3.7 TListControlSelectAll.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.8.3.8 TListControlSelectAll.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView

or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView

TListBox

2.1.8.3.9 TListControlSelectAll.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.8.3.10 TListControlSelectAll.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.8.3.11 TListControlSelectAll.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.8.4 TListControlSelectAll Events

The events of the TListControlSelectAll class are listed here.

Published Events

	Name	Description
	OnHint (see page 78)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 78)	Occurs when the application is idle or when the action list updates.

2.1.8.4.1 TListControlSelectAll.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.8.4.2 TListControlSelectAll.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.9 TNextTab Class

TNextTab moves a tabbed control to its next page.

Class Hierarchy

Pascal

```
TNextTab = class(TTabAction);
```

C++

```
class TNextTab : public TTabAction;
```

File

ExtActns

Description

Add TNextTab to an action list to provide the response to the "Next" and "Finish" buttons or menu items in a tabbed dialog. Controls such as menu items and tool buttons linked to this action cause a different response, depending on the value of the Wrap (see page 88) property and on whether the target control is on the last page:

If the target page control is not on the last page, or if Wrap (see page 88) is true, client controls cause this action to change the current page to the next page. (When Wrap (see page 88) is true, the next page after the last tab is the first tabbed page).

If Wrap (see page 88) is false and the target page control is on the last page, this action changes the caption of client controls (for example, from "Next" to "Finish") and, when fired, responds by generating an OnFinish (see page 90) event.

Before you can use TNextTab, you must assign a target page control or tab control as the value of the TabControl (see page 87) property. Once the target control is assigned, TNextTab automatically disables itself if the target control is not enabled.

See Also

TPreviousTab (see page 101)

TPageControl

TTabControl

2.1.9.1 TNextTab Members

The following tables list the members exposed by TNextTab.

Protected Methods

	Name	Description
	Notification (see page 152)	Responds when components are created or destroyed.
	SelectNextTab (see page 152)	This is SelectNextTab, a member of class TTabAction.

Public Methods

	Name	Description
	Create (see page 151)	Creates an instance of TTabAction (see page 150).
	ExecuteTarget (see page 81)	Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 90) event.
	HandlesTarget (see page 152)	Indicates whether the action can execute its function.

	UpdateTarget (see page 82)	Updates the action to reflect current conditions.

Published Events

	Name	Description
	AfterTabChange (see page 89)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 89)	Occurs immediately before the action changes the page of the associated tab control.
	OnFinish (see page 90)	Occurs when the action fires and the target page control is on its last page.
	OnHint (see page 90)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 90)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 92)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 83)	Represents the caption of client controls and menu items.
	Enabled (see page 83)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 84)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 84)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 85)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 85)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 85)	Indicates the ImageIndex property for client controls and menu items.
	LastTabCaption (see page 86)	Specifies the caption when the target page control is on its last page and Wrap (see page 88) is false.
	ShortCut (see page 86)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 87)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 87)	Specifies the target of the action.
	Visible (see page 88)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 88)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.9.2 TNextTab Methods

The methods of the TNextTab class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 81)	Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 90) event.
	UpdateTarget (see page 82)	Updates the action to reflect current conditions.

2.1.9.2.1 TNextTab.ExecuteTarget Method

Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 90) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget causes the action to perform its default response based on the state of the target tab control.

Target is the currently active object when the action fires. TNextTab (see page 80) ignores this parameter and uses the TabControl (see page 87) property to identify the target control.

ExecuteTarget performs differently, depending on whether the target control is on the last page and on the value of the Wrap (see page 88) property:

If Wrap (see page 88) is true or the target control is not on the last page, ExecuteTarget generates a BeforeTabChange (see page 89) event, changes the target control to the next page, and generates an AfterTabChange (see page 89) event. If Wrap (see page 88) is true and the target control is on the last page, this causes the target control to move to the first page. Note that the SkipHiddenTabs (see page 87) property determines whether the next page can include a hidden page.

If Wrap (see page 88) is false and the target control is on the last page, ExecuteTarget generates an OnFinish (see page 90) event.

See Also

ExecuteTarget

ActivePageIndex

TabControl (see page 154)

OnFinish (see page 90)

BeforeTabChange (see page 155)

AfterTabChange (see page 155)

SkipHiddenTabs (see page 153)

Wrap (see page 154)

TabIndex

2.1.9.2.2 TNextTab.UpdateTarget Method

Updates the action to reflect current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TNextTab (see page 80), UpdateTarget first checks whether the TabControl (see page 87) property specifies an enabled control that can act as a target. If not, UpdateTarget sets the Enabled (see page 83) property to false.

If the target control is enabled, UpdateTarget sets the Enabled (see page 83) property to true and checks whether the target control is on its last page. If so, (and if Wrap (see page 88) is false) it changes the Caption (see page 83) property to the value of LastTabCaption (see page 86). If not, it restores the Caption (see page 83) property to its original value.

See Also

Enabled

Caption

LastTabCaption (see page 86)

TabControl (see page 154)

Wrap (see page 154)

2.1.9.3 TNextTab Properties

The properties of the TNextTab class are listed here.

Published Properties

	Name	Description
	Caption (see page 83)	Represents the caption of client controls and menu items.
	Enabled (see page 83)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 84)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 84)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 85)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 85)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 85)	Indicates the ImageIndex property for client controls and menu items.
	LastTabCaption (see page 86)	Specifies the caption when the target page control is on its last page and Wrap (see page 88) is false.
	ShortCut (see page 86)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 87)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 87)	Specifies the target of the action.
	Visible (see page 88)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 88)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.9.3.1 TNextTab.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.9.3.2 TNextTab.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.9.3.3 TNextTab.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.9.3.4 TNextTab.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be

associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.9.3.5 TNextTab.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.9.3.6 TNextTab.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.9.3.7 TNextTab.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.9.3.8 TNextTab.LastTabCaption Property

Specifies the caption when the target page control is on its last page and Wrap (see page 88) is false.

Pascal

```
property LastTabCaption: String;
```

C++

```
__property UnicodeString LastTabCaption;
```

Description

Set LastTabCaption to the caption of target controls when the control specified by the TabControl (see page 87) property is on its last page and the action can't tab beyond the last page. When the target page control is not on the last page, the caption for client controls is given by the Caption (see page 83) property. When the target page control is on the last page (and Wrap (see page 88) is false), TNextTab (see page 80) caches the original value of Caption (see page 83) and changes the Caption (see page 83) property to LastTabCaption. If the target page control returns to a previous page, TNextTab (see page 80) then restores the original value of the Caption (see page 83) property.

LastTabCaption is designed for client controls such as the button in a tabbed dialog that changes from a "Next" button to a "Finish" button when the tabbed dialog reaches the last page.

See Also

Caption

TabControl (see page 154)

Wrap (see page 154)

2.1.9.3.9 TNextTab.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.9.3.10 TNextTab.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabVisible

TPageControl

2.1.9.3.11 TNextTab.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

2.1.9.3.12 TNextTab.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.9.3.13 TNextTab.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange and an AfterTabChange event or in an OnFinish event.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabControl (see page 154)

BeforeTabChange (see page 155)

AfterTabChange (see page 155)

OnFinish (see page 90)

2.1.9.4 TNextTab Events

The events of the TNextTab class are listed here.

Published Events

	Name	Description
	AfterTabChange (see page 89)	Occurs immediately after the action changes the page of the associated tab control.

	BeforeTabChange (see page 89)	Occurs immediately before the action changes the page of the associated tab control.
	OnFinish (see page 90)	Occurs when the action fires and the target page control is on its last page.
	OnHint (see page 90)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 90)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 92)	Occurs when the application is idle or when the action list updates.

2.1.9.4.1 TNextTab.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.9.4.2 TNextTab.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.9.4.3 TNextTab.OnFinish Event

Occurs when the action fires and the target page control is on its last page.

Pascal

```
property OnFinish: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFinish;
```

Description

Write an OnFinish event handler to respond when the user indicates that they are finished with the tabbed dialog by clicking a client control when the tabbed dialog is on its last page. OnFinish only occurs when the target control is on its last page and the Wrap (see page 88) property is false. When the target page control is on any other page, or when Wrap (see page 88) is true, TNextTab (see page 80) generates an BeforeTabChange (see page 89) event, changes the current page of the target control, and generates an AfterTabChange (see page 89) event.

Note: Use the OnFinish event rather than the action's OnExecute event. If you supply an event handler to the OnExecute event, it blocks the default response by this action, and the action won't change the current page when the target tabbed dialog is not on its last page.

See Also

AfterTabChange (see page 155)

BeforeTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.9.4.4 TNextTab.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.9.4.5 TNextTab.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
```

```
end;
```

See Also

UpdateAction

2.1.9.4.6 TNextTab.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget method, OnValidateTab occurs after UpdateTarget has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate

UpdateTarget (see page 153)

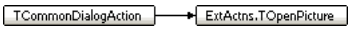
TabControl (see page 154)

Enabled

2.1.10 TOpenPicture Class

TOpenPicture is the standard action for displaying the open picture dialog.

Class Hierarchy



Pascal

```
TOpenPicture = class(TCommonDialogAction);
```

C++

```
class TOpenPicture : public TCommonDialogAction;
```

File

ExtActns

Description

Add TOpenPicture to an action list to add a graphics file selection dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the open picture dialog when invoked. Use the OnAccept (see page 99) event to respond after the user selects a file name in the dialog or the OnCancel (see page 99) event to respond when the user cancels from the dialog. In the event handler, you can read the file name from the dialog specified by the Dialog (see page 94) property.

See Also

TFileOpen

TOpenPictureDialog (see page 1560)

TSavePicture (see page 132)

2.1.10.1 TOpenPicture Members

The following tables list the members exposed by TOpenPicture.

Protected Methods

	Name	Description
	GetDialogClass (see page 94)	This is GetDialogClass, a member of class TOpenPicture.

Published Events

	Name	Description
	OnAccept (see page 99)	This is OnAccept, a member of class TOpenPicture.
	OnCancel (see page 99)	This is OnCancel, a member of class TOpenPicture.
	OnHint (see page 99)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 99)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 94)	Represents the caption of client controls and menu items.
	Dialog (see page 94)	Provides access the open picture dialog that the action executes.
	Enabled (see page 95)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 95)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 96)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 96)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 97)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 97)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 97)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 98)	Specifies the ShortCut property for client menu items.
	Visible (see page 98)	Specifies the Visible property for client controls and menu items.

2.1.10.2 TOpenPicture Methods

The methods of the TOpenPicture class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 94)	This is GetDialogClass, a member of class TOpenPicture.

2.1.10.2.1 TOpenPicture.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TOpenPicture.

2.1.10.3 TOpenPicture Properties

The properties of the TOpenPicture class are listed here.

Published Properties

	Name	Description
	Caption (see page 94)	Represents the caption of client controls and menu items.
R	Dialog (see page 94)	Provides access the open picture dialog that the action executes.
	Enabled (see page 95)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 95)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 96)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 96)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 97)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 97)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 97)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 98)	Specifies the ShortCut property for client menu items.
	Visible (see page 98)	Specifies the Visible property for client controls and menu items.

2.1.10.3.1 TOpenPicture.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.10.3.2 TOpenPicture.Dialog Property

Provides access the open picture dialog that the action executes.

Pascal

```
property Dialog: TOpenPictureDialog;
```

C++

```
__property TOpenPictureDialog Dialog;
```

Description

Use Dialog to configure the graphics file selection dialog before the action fires or to read the selected file name after the action executes the dialog. For example, you can set the dialog's default extension or initially selected file at design time using the Object Inspector or read the dialog's FileName property from an OnAccept (see page 99) event handler to use the file that the user selected.

When using an open picture action, you do not place a TOpenPicture (see page 92) Dialog component on the form or data module at design time. Instead, you add the TOpenPicture (see page 92) object to your action list and then use the Dialog property. The dialog component is created by the TOpenPicture (see page 92) constructor.

See Also

onaccept

oncancel

FileName

2.1.10.3.3 TOpenPicture.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.10.3.4 TOpenPicture.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated

action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.10.3.5 TOpenPicture.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.10.3.6 TOpenPicture.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.10.3.7 TOpenPicture.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.10.3.8 TOpenPicture.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.10.3.9 TOpenPicture.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.10.3.10 TOpenPicture.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.10.3.11 TOpenPicture.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.10.4 TOpenPicture Events

The events of the TOpenPicture class are listed here.

Published Events

	Name	Description
	OnAccept (see page 99)	This is OnAccept, a member of class TOpenPicture.
	OnCancel (see page 99)	This is OnCancel, a member of class TOpenPicture.
	OnHint (see page 99)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 99)	Occurs when the application is idle or when the action list updates.

2.1.10.4.1 TOpenPicture.OnAccept Event

Pascal

```
property OnAccept;
```

C++

```
__property OnAccept;
```

Description

This is OnAccept, a member of class TOpenPicture.

2.1.10.4.2 TOpenPicture.OnCancel Event

Pascal

```
property OnCancel;
```

C++

```
__property OnCancel;
```

Description

This is OnCancel, a member of class TOpenPicture.

2.1.10.4.3 TOpenPicture.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.10.4.4 TOpenPicture.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
```

```
end;
```

See Also

UpdateAction

2.1.11 TPreviousTab Class

TPreviousTab moves a page control to its previous page.

Class Hierarchy

Pascal

```
TPreviousTab = class(TTabAction);
```

C++

```
class TPreviousTab : public TTabAction;
```

File

ExtActns

Description

Add TPreviousTab to an action list to allow users to move to the prior page of a page control using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application change the current page of a page control to the previous one. Typically, these client controls are the "Prior" button or menu item in a tabbed dialog.

Before you can use TPreviousTab, you must assign a target tab control as the value of the TabControl (see page 107) property. Once the target tab control is assigned, TPreviousTab automatically disables itself if the target control is not enabled or if it is on the first page and wrapping is not permitted.

See Also

TNextTab (see page 80)

TPageControl

TTabControl

2.1.11.1 TPreviousTab Members

The following tables list the members exposed by TPreviousTab.

Protected Methods

	Name	Description
	Notification (see page 152)	Responds when components are created or destroyed.
	SelectNextTab (see page 152)	This is SelectNextTab, a member of class TTabAction.

Public Methods

	Name	Description
	Create (see page 151)	Creates an instance of TTabAction (see page 150).
	ExecuteTarget (see page 102)	Changes the selected page in the target control to the previous one.
	HandlesTarget (see page 152)	Indicates whether the action can execute its function.
	UpdateTarget (see page 103)	Enables or disables the action, depending on the status of the target control.

Published Events

	Name	Description
	AfterTabChange (see page 109)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 109)	Occurs immediately before the action changes the page of the associated tab control.
	OnHint (see page 110)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 110)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 111)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 104)	Represents the caption of client controls and menu items.
	Enabled (see page 104)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 105)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 105)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 106)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 106)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 106)	Indicates the ImageIndex property for client controls and menu items.
	ShortCut (see page 107)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 107)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 107)	Specifies the target of the action.
	Visible (see page 108)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 108)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.11.2 TPreviousTab Methods

The methods of the TPreviousTab class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 102)	Changes the selected page in the target control to the previous one.
	UpdateTarget (see page 103)	Enables or disables the action, depending on the status of the target control.

2.1.11.2.1 TPreviousTab.ExecuteTarget Method

Changes the selected page in the target control to the previous one.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget causes the action to perform its default response based on the state of the target tab control.

Target is the currently active object when the action fires. TPreviousTab (see page 101) ignores this parameter and uses the TabControl (see page 107) property to identify the target control.

ExecuteTarget

1. Generates a BeforeTabChange (see page 109) event.
2. Moves the control identified by TabControl (see page 107) to the previous page. If TabControl (see page 107) is on the first page and the Wrap (see page 108) property is true, this moves to the last page. If Wrap (see page 108) is false, ExecuteTarget does not change the page when the tab control is on the first page. Note that the SkipHiddenTabs (see page 107) property controls whether hidden pages are considered when locating the previous page.
3. Generates an AfterTabChange (see page 109) event.

See Also

ExecuteTarget

ActivePageIndex

TabControl (see page 154)

BeforeTabChange (see page 155)

AfterTabChange (see page 155)

Wrap (see page 154)

TabIndex

SkipHiddenTabs (see page 153)

2.1.11.2.2 TPreviousTab.UpdateTarget Method

Enables or disables the action, depending on the status of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object.

In TPreviousTab (see page 101), UpdateTarget ignores the Target parameter and checks whether the target control specified by the TabControl (see page 107) property is enabled. If so, it sets Enabled (see page 104) to true if the first page is not currently selected or if Wrap (see page 108) is true. It sets Enabled (see page 104) to false if TabControl (see page 107) is not enabled or if the first page is selected and Wrap (see page 108) is false.

See Also

Enabled

TabControl (see page 154)

Wrap (see page 154)

2.1.11.3 TPreviousTab Properties

The properties of the TPreviousTab class are listed here.

Published Properties

	Name	Description
	Caption (see page 104)	Represents the caption of client controls and menu items.
	Enabled (see page 104)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 105)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 105)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 106)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 106)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 106)	Indicates the ImageIndex property for client controls and menu items.
	ShortCut (see page 107)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 107)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 107)	Specifies the target of the action.
	Visible (see page 108)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 108)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.11.3.1 TPreviousTab.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.11.3.2 TPreviousTab.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.11.3.3 TPreviousTab.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.11.3.4 TPreviousTab.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.11.3.5 TPreviousTab.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.11.3.6 TPreviousTab.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.11.3.7 TPreviousTab.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.11.3.8 TPreviousTab.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.11.3.9 TPreviousTab.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabVisible

TPageControl

2.1.11.3.10 TPreviousTab.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

2.1.11.3.11 TPreviousTab.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.11.3.12 TPreviousTab.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange and an AfterTabChange event or in an OnFinish event.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabControl (see page 154)

BeforeTabChange (see page 155)

AfterTabChange (see page 155)

OnFinish (see page 90)

2.1.11.4 TPreviousTab Events

The events of the TPreviousTab class are listed here.

Published Events

	Name	Description
	AfterTabChange (see page 109)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 109)	Occurs immediately before the action changes the page of the associated tab control.
	OnHint (see page 110)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 110)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 111)	Occurs when the application is idle or when the action list updates.

2.1.11.4.1 TPreviousTab.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.11.4.2 TPreviousTab.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.11.4.3 TPreviousTab.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.11.4.4 TPreviousTab.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
```

```
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

2.1.11.4.5 TPreviousTab.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget method, OnValidateTab occurs after UpdateTarget has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate

UpdateTarget (see page 153)

TabControl (see page 154)

Enabled

2.1.12 TRichEditAction Class

TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAction = class(TEditAction);
```

C++

```
class TRichEditAction : public TEditAction;
```

File

ExtActns

Description

TRichEditAction implements the common behavior for actions that assign text attributes to a rich edit control. Each descendant class represents a specific text attribute. Do not use TRichEditAction directly in an application. Instead, use a TRichEditAction descendant such as TRichEditBold (see page 121), TRichEditItalic (see page 125), TRichEditUnderline (see page 130), TRichEditStrikeOut (see page 127), TRichEditAlignCenter (see page 115), TRichEditAlignLeft (see page 117), TRichEditAlignRight (see page 119), or TRichEditBullets (see page 123).

TRichEditAction ensures that the action is only works with a TCustomRichEdit descendant as a target. TRichEditAction descendants toggle their specific text attribute in the target rich edit control: if the attribute is set, it is removed, and if it is not set (or set for only some of the text), the action sets it. If the target rich edit control has a selection, the attribute is toggled for the selected text only. If there is no selection, the attribute is toggled for all of the text.

See Also

TCustomRichEdit

TRichEditBold (see page 121)

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditAlignCenter (see page 115)

TRichEditAlignLeft (see page 117)

TRichEditAlignRight (see page 119)

TRichEditBullets (see page 123)

TRichEditStrikeOut (see page 127)

2.1.12.1 TRichEditAction Members

The following tables list the members exposed by TRichEditAction.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.

2.1.12.2 TRichEditAction.Create Constructor

Creates an instance of TRichEditAction (see page 112).

Pascal

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

C++

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

Description

Most applications do not need to call the TRichEditAction (see page 112) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TRichEditAction (see page 112) instance.

See Also

Owner

2.1.12.3 TRichEditAction Methods

The methods of the TRichEditAction class are listed here.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.

2.1.12.3.1 TRichEditAction.CurrText Method

Pascal

```
function CurrText(Edit: TCustomRichEdit): TTextAttributes;
```

C++

```
__fastcall TTextAttributes CurrText(TCustomRichEdit Edit);
```

Description

This is CurrText, a member of class TRichEditAction.

2.1.12.3.2 TRichEditAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget returns true if Target is a TCustomRichEdit descendant, false otherwise.

See Also

ExecuteTarget

TCustomRichEdit

2.1.12.3.3 TRichEditAction.SetFontStyle Method

Pascal

```
procedure SetFontStyle(Edit: TCustomRichEdit; Style: TFontStyle);
```

C++

```
__fastcall SetFontStyle(TCustomRichEdit Edit, TFontStyle Style);
```

Description

This is SetFontStyle, a member of class TRichEditAction.

2.1.13 TRichEditAlignCenter Class

TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAlignCenter = class(TRichEditAction);
```

C++

```
class TRichEditAlignCenter : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignCenter to an action list to let users center the text in a rich edit control. Controls such as menu items and tool buttons linked to this action cause the application to center the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected).

TRichEditAlignCenter is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to reflect whether the text in the target edit control is centered.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditStrikeOut (see page 127)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditBullets (see page 123)

TRichEditBold (see page 121)

2.1.13.1 TRichEditAlignCenter Members

The following tables list the members exposed by TRichEditAlignCenter.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 116)	Centers the text in the current paragraph of the target rich edit control.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 116)	Configures the action, depending on current conditions.

2.1.13.2 TRichEditAlignCenter Methods

The methods of the TRichEditAlignCenter class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 116)	Centers the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 116)	Configures the action, depending on current conditions.

2.1.13.2.1 TRichEditAlignCenter.ExecuteTarget Method

Centers the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taCenter.

See Also

Paragraph

Alignment

2.1.13.2.2 TRichEditAlignCenter.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TRichEditAlignCenter (see page 115), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in

the current paragraph is centered.

See Also

Enabled

Checked

2.1.14 TRichEditAlignLeft Class

TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAlignLeft = class(TRichEditAction);
```

C++

```
class TRichEditAlignLeft : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignLeft to an action list to let users left-justify the text in a rich edit control. Controls such as menu items and tool buttons linked to this action cause the application to align the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected) to the left margin.

TRichEditAlignLeft is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the text in the target control is left-aligned.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditStrikeOut (see page 127)

TRichEditAlignRight (see page 119)

TRichEditAlignCenter (see page 115)

TRichEditBullets (see page 123)

TRichEditBold (see page 121)

2.1.14.1 TRichEditAlignLeft Members

The following tables list the members exposed by TRichEditAlignLeft.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 118)	Left-justifies the text in the current paragraph of the target rich edit control.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 118)	Configures the action, depending on current conditions.

2.1.14.2 TRichEditAlignLeft Methods

The methods of the TRichEditAlignLeft class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 118)	Left-justifies the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 118)	Configures the action, depending on current conditions.

2.1.14.2.1 TRichEditAlignLeft.ExecuteTarget Method

Left-justifies the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taLeftJustify.

See Also

Paragraph

Alignment

2.1.14.2.2 TRichEditAlignLeft.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TRichEditAlignLeft (see page 117), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in the current paragraph is left justified.

See Also

Enabled

Checked

2.1.15 TRichEditAlignRight Class

TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAlignRight = class(TRichEditAction);
```

C++

```
class TRichEditAlignRight : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignRight to an action list to let users right-justify the text in a rich edit control. Controls such as menu items and tool buttons linked to this action cause the application to align the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected) to the right margin.

TRichEditAlignRight is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the current paragraph of the target control is right-aligned.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditStrikeOut (see page 127)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

TRichEditBullets (see page 123)

TRichEditBold (see page 121)

2.1.15.1 TRichEditAlignRight Members

The following tables list the members exposed by TRichEditAlignRight.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 120)	Right-justifies the text in the current paragraph of the target rich edit control.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 120)	Configures the action, depending on current conditions.

2.1.15.2 TRichEditAlignRight Methods

The methods of the TRichEditAlignRight class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 120)	Right-justifies the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 120)	Configures the action, depending on current conditions.

2.1.15.2.1 TRichEditAlignRight.ExecuteTarget Method

Right-justifies the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taRightJustify.

See Also

Paragraph

Alignment

2.1.15.2.2 TRichEditAlignRight.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TRichEditAlignRight (see page 119), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in the current paragraph is right justified.

See Also

Enabled

Checked

2.1.16 TRichEditBold Class

TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditBold = class(TRichEditAction);
```

C++

```
class TRichEditBold : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditBold to an action list to let users control whether the selected text in a rich edit control is bold. Controls such as menu items and tool buttons linked to this action cause the application to toggle the bold attribute of selected text (or of all text if there is no text selected).

TRichEditBold is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the target control's text is bold.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditBullets (see page 123)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

TRichEditStrikeOut (see page 127)

2.1.16.1 TRichEditBold Members

The following tables list the members exposed by TRichEditBold.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 122)	Toggles the fsBold style of the target rich edit control's text attributes.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 123)	Sets the Checked property to reflect whether the target control's text is bold.

2.1.16.2 TRichEditBold Methods

The methods of the TRichEditBold class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 122)	Toggles the fsBold style of the target rich edit control's text attributes.
	UpdateTarget (see page 123)	Sets the Checked property to reflect whether the target control's text is bold.

2.1.16.2.1 TRichEditBold.ExecuteTarget Method

Toggles the fsBold style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsBold style of the selected text attributes (SelAttributes property), adding the fsBold property if the selected text is not all bold and removing it if the selected text is entirely bold. Note that if the selected text contains both bold and non-bold text, this results in the text becoming entirely bold.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsBold style of the control. This changes the bold attribute of all text that has not had a specific text attribute applied.

See Also

Style

SelAttributes

DefAttributes

2.1.16.2.2 TRichEditBold.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text is bold.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget sets the Checked property to true if Target is a rich edit control whose selected text is bold, otherwise it sets Checked to false.

See Also

Style

SelAttributes

Checked

2.1.17 TRichEditBullets Class

TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.

Class Hierarchy

Pascal

```
TRichEditBullets = class(TRichEditAction);
```

C++

```
class TRichEditBullets : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditBullets to an action list to let users control whether the current paragraphs in a rich edit control are bulleted. Controls such as menu items and tool buttons linked to this action cause the application to toggle the bulleted attribute of all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected).

TRichEditBullets is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control and adds or removes a check mark to indicate whether that control's paragraphs are bulleted.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditStrikeOut (see page 127)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

TRichEditBold (see page 121)

2.1.17.1 TRichEditBullets Members

The following tables list the members exposed by TRichEditBullets.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 124)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 125)	Configures the action, depending on current conditions.

2.1.17.2 TRichEditBullets Methods

The methods of the TRichEditBullets class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 124)	Introduces an interface for invoking an action on a target client component or control.
	UpdateTarget (see page 125)	Configures the action, depending on current conditions.

2.1.17.2.1 TRichEditBullets.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute

HandlesTarget

2.1.17.2.2 TRichEditBullets.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In UpdateTarget, UpdateTarget sets Enabled to T true and sets the Checked property to T true if the current paragraph is bulleted or to F false if the current paragraph is not bulleted.

See Also

Enabled

Checked

2.1.18 TRichEditItalic Class

TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditItalic = class(TRichEditAction);
```

C++

```
class TRichEditItalic : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditItalic to an action list to let users control whether the selected text in a rich edit control is italic. Controls such as menu items and tool buttons linked to this action cause the application to toggle the italic attribute of selected text (or of all text if there is no text selected).

TRichEditItalic is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to reflect whether the target control's text is italic.

See Also

TCustomRichEdit

TRichEditBold (see page 121)

TRichEditUnderline (see page 130)

TRichEditBullets (see page 123)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

TRichEditStrikeout

2.1.18.1 TRichEditItalic Members

The following tables list the members exposed by TRichEditItalic.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 127)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 127)	Enables or disables the action, depending on the currently active control in the form.

2.1.18.2 TRichEditItalic Methods

The methods of the TRichEditItalic class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 127)	Introduces an interface for invoking an action on a target client component or control.
	UpdateTarget (see page 127)	Enables or disables the action, depending on the currently active control in the form.

2.1.18.2.1 TRichEditItalic.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute

HandlesTarget

2.1.18.2.2 TRichEditItalic.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TEditAction, UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control and, if the action's function requires it, that some of its text is selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

2.1.19 TRichEditStrikeOut Class

TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditStrikeOut = class(TRichEditAction);
```

C++

```
class TRichEditStrikeOut : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditStrikeOut to an action list to let users control whether the selected text in a rich edit control has a line through it. Controls such as menu items and tool buttons linked to this action cause the application to toggle the strikeout attribute of selected text (or of all text if there is no text selected).

TRichEditStrikeOut is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control and adds or removes a check mark to indicate whether the target control's text has the strike out attribute.

See Also

TCustomRichEdit

TRichEditItalic (see page 125)

TRichEditUnderline (see page 130)

TRichEditBullets (see page 123)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

2.1.19.1 TRichEditStrikeOut Members

The following tables list the members exposed by TRichEditStrikeOut.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 129)	Toggles the fsStrikeOut style of the target rich edit control's text attributes.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 129)	Sets the Checked property to reflect whether the target control's text includes a strikeout line.

2.1.19.2 TRichEditStrikeOut Methods

The methods of the TRichEditStrikeOut class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 129)	Toggles the fsStrikeOut style of the target rich edit control's text attributes.

	UpdateTarget (see page 129)	Sets the Checked property to reflect whether the target control's text includes a strikeout line.

2.1.19.2.1 TRichEditStrikeOut.ExecuteTarget Method

Toggles the fsStrikeOut style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsStrikeOut style of the selected text attributes (SelAttributes property), adding the fsStrikeOut property if the selected text is not all struck out and removing it if the selected text is entirely struck out. Note that if the selected text contains both strikeout and non-strikeout text, this results in the text becoming entirely struck out.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsStrikeOut style of the control. This changes the strikeout attribute of all text that has not had a specific text attribute applied.

See Also

Style

SelAttributes

DefAttributes

2.1.19.2.2 TRichEditStrikeOut.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text includes a strikeout line.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget checks whether Target is a rich edit control with the strikeout font attribute on its selected text. If the selected text is struck out, UpdateTarget sets the Checked property to true, otherwise it sets Checked to false.

See Also

Style

SelAttributes

Checked

2.1.20 TRichEditUnderline Class

TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditUnderline = class(TRichEditAction);
```

C++

```
class TRichEditUnderline : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditUnderline to an action list to let users control whether the selected text in a rich edit control is underlined. Controls such as menu items and tool buttons linked to this action cause the application to toggle the underline attribute of selected text (or of all text if there is no text selected).

TRichEditUnderline is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check to indicate whether the target control's text is underlined.

See Also

TCustomRichEdit

TRichEditBold (see page 121)

TRichEditUnderline

TRichEditStrikeOut (see page 127)

TRichEditBullets (see page 123)

TRichEditAlignRight (see page 119)

TRichEditAlignLeft (see page 117)

TRichEditAlignCenter (see page 115)

TRichEditStrikeOut (see page 127)

2.1.20.1 TRichEditUnderline Members

The following tables list the members exposed by TRichEditUnderline.

Protected Methods

	Name	Description
	CurrText (see page 114)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 114)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	Create (see page 113)	Creates an instance of TRichEditAction (see page 112).
	ExecuteTarget (see page 131)	Toggles the fsUnderline style of the target rich edit control's text attributes.
	HandlesTarget (see page 114)	Indicates whether the action can execute its function.
	UpdateTarget (see page 131)	Sets the Checked property to reflect whether the target control's text is underlined.

2.1.20.2 TRichEditUnderline Methods

The methods of the TRichEditUnderline class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 131)	Toggles the fsUnderline style of the target rich edit control's text attributes.
	UpdateTarget (see page 131)	Sets the Checked property to reflect whether the target control's text is underlined.

2.1.20.2.1 TRichEditUnderline.ExecuteTarget Method

Toggles the fsUnderline style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsUnderline style of the selected text attributes (SelAttributes property), adding the fsUnderline property if the selected text is not all underlined and removing it if the selected text is entirely underlined. Note that if the selected text contains both underlined and non-underlined text, this results in the text becoming entirely underlined.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsUnderline style of the control. This changes the underlined attribute of all text that has not had a specific text attribute applied.

See Also

Style

SelAttributes

DefAttributes

2.1.20.2.2 TRichEditUnderline.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text is underlined.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget checks the selected text of the target rich edit control (which is specified by the Control property or the Target parameter) is underlined. If the selected text is underlined, UpdateTarget sets the Checked property to true, otherwise it sets Checked to false.

See Also

Style

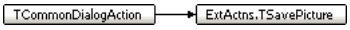
SelAttributes

Checked

2.1.21 TSavePicture Class

TSavePicture is the standard action for displaying the save picture dialog.

Class Hierarchy



Pascal

```
TSavePicture = class(TCommonDialogAction);
```

C++

```
class TSavePicture : public TCommonDialogAction;
```

File

ExtActns

Description

Add TSavePicture to an action list to add a graphics "save as" dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the save picture dialog when invoked. Use the OnAccept (see page 139) event to respond after the user selects a file name in the dialog or the OnCancel (see page 139) event to respond when the user cancels from the dialog. In the event handler, you can read the file name from the dialog specified by the Dialog (see page 134) property.

See Also

TFileSaveAs

TSavePictureDialog (see page 1577)

TSavePicture

2.1.21.1 TSavePicture Members

The following tables list the members exposed by TSavePicture.

Protected Methods

	Name	Description
	GetDialogClass (see page 133)	This is GetDialogClass, a member of class TSavePicture.

Published Events

	Name	Description
	OnAccept (see page 139)	This is OnAccept, a member of class TSavePicture.
	OnCancel (see page 139)	This is OnCancel, a member of class TSavePicture.
	OnHint (see page 139)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 139)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 134)	Represents the caption of client controls and menu items.
	Dialog (see page 134)	Provides access the save picture dialog that the action executes.
	Enabled (see page 135)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 135)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 136)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 136)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 137)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 137)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 137)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 138)	Specifies the ShortCut property for client menu items.
	Visible (see page 138)	Specifies the Visible property for client controls and menu items.

2.1.21.2 TSavePicture Methods

The methods of the TSavePicture class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 133)	This is GetDialogClass, a member of class TSavePicture.

2.1.21.2.1 TSavePicture.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSavePicture.

2.1.21.3 TSavePicture Properties

The properties of the TSavePicture class are listed here.

Published Properties

	Name	Description
	Caption (see page 134)	Represents the caption of client controls and menu items.
R	Dialog (see page 134)	Provides access the save picture dialog that the action executes.
	Enabled (see page 135)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 135)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 136)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 136)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 137)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 137)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 137)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 138)	Specifies the ShortCut property for client menu items.
	Visible (see page 138)	Specifies the Visible property for client controls and menu items.

2.1.21.3.1 TSavePicture.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.21.3.2 TSavePicture.Dialog Property

Provides access the save picture dialog that the action executes.

Pascal

```
property Dialog: TSavePictureDialog;
```

C++

```
__property TSavePictureDialog Dialog;
```

Description

Use Dialog to configure the graphics file "save as" dialog before the action fires or to read the specified file name after the action executes the dialog. For example, you can set the dialog's default extension or initial file name at design time using the Object Inspector or read the dialog's FileName property from an OnAccept (see page 139) event handler to use the file name that the user selects.

When using a save picture action, you do not place a TSavePicture (see page 132) Dialog component on the form or data module at design time. Instead, you add the TSavePicture (see page 132) object to your action list and then use the Dialog property. The dialog component is created by the TSavePicture (see page 132) constructor.

See Also

onaccept

oncancel

FileName

2.1.21.3.3 TSavePicture.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.21.3.4 TSavePicture.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.21.3.5 TSavePicture.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.21.3.6 TSavePicture.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.21.3.7 TSavePicture.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.21.3.8 TSavePicture.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.21.3.9 TSavePicture.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.21.3.10 TSavePicture.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.21.3.11 TSavePicture.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.21.4 TSavePicture Events

The events of the TSavePicture class are listed here.

Published Events

	Name	Description
	OnAccept (see page 139)	This is OnAccept, a member of class TSavePicture.
	OnCancel (see page 139)	This is OnCancel, a member of class TSavePicture.
	OnHint (see page 139)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 139)	Occurs when the application is idle or when the action list updates.

2.1.21.4.1 TSavePicture.OnAccept Event

Pascal

```
property OnAccept;
```

C++

```
__property OnAccept;
```

Description

This is OnAccept, a member of class TSavePicture.

2.1.21.4.2 TSavePicture.OnCancel Event

Pascal

```
property OnCancel;
```

C++

```
__property OnCancel;
```

Description

This is OnCancel, a member of class TSavePicture.

2.1.21.4.3 TSavePicture.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.21.4.4 TSavePicture.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

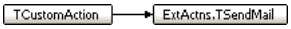
See Also

UpdateAction

2.1.22 TSendMail Class

TSendMail is the standard action for sending an email message.

Class Hierarchy



Pascal

```
TSendMail = class(TCustomAction);
```

C++

```
class TSendMail : public TCustomAction;
```

File

ExtActns

Description

Add TSendMail to an action list to let your application send email messages in response to user actions. Controls linked to this action cause the application to send a MAPI mail message containing the message specified in the Text (see page 147) property.

See Also

TFileRun (see page 27)

TSendMail

2.1.22.1 TSendMail Members

The following tables list the members exposed by TSendMail.

Public Methods

	Name	Description
	Destroy (see page 142)	Frees the memory associated with the TSendMail (see page 141) object.
	ExecuteTarget (see page 142)	Displays a dialog where the user can compose an email message and then sends the message.
	HandlesTarget (see page 143)	Indicates whether the action can execute.
	UpdateTarget (see page 143)	Enables the action.

Published Events

	Name	Description
	OnHint (see page 148)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 148)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 144)	Represents the caption of client controls and menu items.
	Enabled (see page 144)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 144)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 145)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 145)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 146)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 146)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 146)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 147)	Specifies the ShortCut property for client menu items.
	Text (see page 147)	Specifies the content of the mail message.
	Visible (see page 148)	Specifies the Visible property for client controls and menu items.

2.1.22.2 TSendMail.Destroy Destructor

Frees the memory associated with the TSendMail (see page 141) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSendMail();
```

Description

Do not explicitly destroy actions. Action objects are created and destroyed by the action manager or action list that uses them.

See Also

System::TObject::Free

2.1.22.3 TSendMail Methods

The methods of the TSendMail class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 142)	Displays a dialog where the user can compose an email message and then sends the message.
	HandlesTarget (see page 143)	Indicates whether the action can execute.
	UpdateTarget (see page 143)	Enables the action.

2.1.22.3.1 TSendMail.ExecuteTarget Method

Displays a dialog where the user can compose an email message and then sends the message.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter, and displays a dialog where the user can compose an email message. The body of the message is initialized to the value of the Text (see page 147) property. When the user exits the dialog, the message is sent.

ExecuteTarget always spawns a temporary MAPI session for every mail message, which lasts only for the duration of the

message. If necessary, the user is prompted with a login dialog.

See Also

Text (see page 147)

2.1.22.3.2 TSendMail.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget ignores the Target parameter and always returns true, because the ability to send mail is not dependant on the state of the application.

2.1.22.3.3 TSendMail.UpdateTarget Method

Enables the action.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically the application is idle so that actions can update themselves according to current conditions.

In TSendMail (see page 141), UpdateTarget ignores the Target parameter and always sets the Enabled (see page 144) property to true.

See Also

Enabled

2.1.22.4 TSendMail Properties

The properties of the TSendMail class are listed here.

Published Properties

	Name	Description
	Caption (see page 144)	Represents the caption of client controls and menu items.
	Enabled (see page 144)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 144)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 145)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 145)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 146)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 146)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 146)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 147)	Specifies the ShortCut property for client menu items.
	Text (see page 147)	Specifies the content of the mail message.
	Visible (see page 148)	Specifies the Visible property for client controls and menu items.

2.1.22.4.1 TSendMail.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

2.1.22.4.2 TSendMail.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

2.1.22.4.3 TSendMail.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

2.1.22.4.4 TSendMail.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

2.1.22.4.5 TSendMail.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

2.1.22.4.6 TSendMail.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

2.1.22.4.7 TSendMail.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex (see page 4916)

Images

2.1.22.4.8 TSendMail.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

2.1.22.4.9 TSendMail.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

2.1.22.4.10 TSendMail.Text Property

Specifies the content of the mail message.

Pascal

```
property Text: TStrings;
```

C++

```
__property TStrings Text;
```

Description

Set Text to the content of the mail message. Because Text is a TStrings object, you can use the string list editor at design time to compose the message. Use the TStrings object's LoadFromFile method to load the value of Text from a file.

When the action executes, TSendMail (see page 141) displays a dialog for the user to compose the mail message. In that dialog, the body of the message is initialized to Text.

See Also

ExecuteTarget (see page 142)

LoadFromFile

2.1.22.4.11 TSendMail.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

2.1.22.5 TSendMail Events

The events of the TSendMail class are listed here.

Published Events

	Name	Description
	OnHint (see page 148)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 148)	Occurs when the application is idle or when the action list updates.

2.1.22.5.1 TSendMail.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

2.1.22.5.2 TSendMail.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

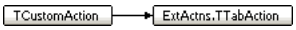
See Also

UpdateAction

2.1.23 TTabAction Class

TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.

Class Hierarchy



Pascal

```
TTabAction = class(TCustomAction);
```

C++

```
class TTabAction : public TCustomAction;
```

File

ExtActns

Description

TTabAction implements the common behavior for actions that operate on tabbed controls (TCustomTabControl descendants). Each descendant class is designed to perform a specific function in a tabbed dialog such as a wizard. Do not use TTabAction directly in an application. Instead, use a TTabAction descendant such as TNextTab (see page 80) or TPreviousTab (see page 101).

TTabAction introduces the TabControl (see page 154) property, which specifies the page control or tab control on which this action operates. It ensures that the action is only enabled when the TabControl (see page 154) property is assigned to an enabled TPageControl or TTabControl object.

See Also

TPreviousTab (see page 101)

TNextTab (see page 80)

TPageControl

TCustomTabControl

TTabControl

2.1.23.1 TTabAction Members

The following tables list the members exposed by TTabAction.

Protected Methods

	Name	Description
	Notification (see page 152)	Responds when components are created or destroyed.
	SelectNextTab (see page 152)	This is SelectNextTab, a member of class TTabAction.

Public Events

	Name	Description
	AfterTabChange (see page 155)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 155)	Occurs immediately before the action changes the page of the associated tab control.
	OnValidateTab (see page 156)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	Create (see page 151)	Creates an instance of TTabAction (see page 150).
	HandlesTarget (see page 152)	Indicates whether the action can execute its function.
	UpdateTarget (see page 153)	Enables or disables the action, depending on the target tab control.

Public Properties

	Name	Description
	SkipHiddenTabs (see page 153)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 154)	Specifies the target of the action.
	Wrap (see page 154)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.23.2 TTabAction.Create Constructor

Creates an instance of TTabAction (see page 150).

Pascal

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

C++

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

Description

Do not call the TTabAction (see page 150) constructor. Actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TTabAction (see page 150) instance.

See Also

Owner

2.1.23.3 TTabAction Methods

The methods of the TTabAction class are listed here.

Protected Methods

	Name	Description
	Notification (see page 152)	Responds when components are created or destroyed.
	SelectNextTab (see page 152)	This is SelectNextTab, a member of class TTabAction.

Public Methods

	Name	Description
	HandlesTarget (see page 152)	Indicates whether the action can execute its function.

	UpdateTarget (see page 153)	Enables or disables the action, depending on the target tab control.

2.1.23.3.1 TTabAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides an action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

In TTabAction (see page 150), HandlesTarget ignores the Target parameter, and checks, instead, whether the TabControl (see page 154) property has been set. If a target tab control has been assigned to the TabControl (see page 154) property, HandlesTarget returns true. Otherwise, it returns false.

See Also

ExecuteTarget

TabControl (see page 154)

2.1.23.3.2 TTabAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

2.1.23.3.3 TTabAction.SelectNextTab Method

Pascal

```
procedure SelectNextTab(GoForward: Boolean = True);
```

C++

```
__fastcall SelectNextTab(Boolean GoForward = True);
```

Description

This is SelectNextTab, a member of class TTabAction.

2.1.23.3.4 TTabAction.UpdateTarget Method

Enables or disables the action, depending on the target tab control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to the current conditions.

Target is the potential target object identified by the application.

In TTabAction (see page 150), UpdateTarget ignores the Target parameter and checks, instead, whether the TabControl (see page 154) property is set. If TabControl (see page 154) identifies an enabled page control or tab control that has more than one page, UpdateTarget sets the Enabled property to true. If not, it sets Enabled to false.

After setting the Enabled property, UpdateTarget generates an OnValidateTab (see page 156) event, where application-specific code can further determine whether the action should be enabled.

See Also

Enabled

TabControl (see page 154)

Enabled

OnValidateTab (see page 156)

2.1.23.4 TTabAction Properties

The properties of the TTabAction class are listed here.

Public Properties

	Name	Description
	SkipHiddenTabs (see page 153)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 154)	Specifies the target of the action.
	Wrap (see page 154)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

2.1.23.4.1 TTabAction.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl (see page 154) property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabVisible

TPageControl

2.1.23.4.2 TTabAction.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction (see page 150) descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

2.1.23.4.3 TTabAction.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange (see page 155) and an AfterTabChange (see page 155) event or in an OnFinish event.

See Also

ExecuteTarget (see page 81)

ExecuteTarget (see page 102)

TabControl (see page 154)

BeforeTabChange (see page 155)

AfterTabChange (see page 155)

OnFinish (see page 90)

2.1.23.5 TTabAction Events

The events of the TTabAction class are listed here.

Public Events

	Name	Description
	AfterTabChange (see page 155)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 155)	Occurs immediately before the action changes the page of the associated tab control.
	OnValidateTab (see page 156)	Occurs when the application is idle or when the action list updates.

2.1.23.5.1 TTabAction.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.23.5.2 TTabAction.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 155)

TabControl (see page 154)

Wrap (see page 154)

2.1.23.5.3 TTabAction.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget (see page 153) method, OnValidateTab occurs after UpdateTarget (see page 153) has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl (see page 154) is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate

UpdateTarget (see page 153)

TabControl (see page 154)

Enabled

2.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TShowCmd (see page 157)	This is record ExtActns.TShowCmd.

2.2.1 ExtActns.TShowCmd Enumeration

Pascal

```
TShowCmd = (
  scHide,
  scMaximize,
  scMinimize,
  scRestore,
  scShow,
  scShowDefault,
  scShowMaximized,
  scShowMinimized,
  scShowMinNoActive,
  scShowNA,
  scShowNoActivate,
  scShowNormal
);
```

C++

```
enum TShowCmd {
  scHide,
  scMaximize,
  scMinimize,
  scRestore,
  scShow,
  scShowDefault,
  scShowMaximized,
  scShowMinimized,
  scShowMinNoActive,
  scShowNA,
  scShowNoActivate,
  scShowNormal
};
```

File

ExtActns

Description

This is record ExtActns.TShowCmd.

2.3 Types

The following table lists types in this documentation.

Types

Name	Description
TValidateTabEvent (see page 158)	This is type ExtActns.TValidateTabEvent.

2.3.1 ExtActns.TValidateTabEvent Type

Pascal

```
TValidateTabEvent = procedure (Sender: TTabAction; TabControl: TCustomTabControl; var Enabled:
Boolean) of object;
```

C++

```
(Sender: TTabAction; TabControl: TCustomTabControl; var Enabled: Boolean) (
TValidateTabEvent)();
```

File

ExtActns

Description

This is type ExtActns.TValidateTabEvent.

3 ExtCtrls Namespace

This is namespace ExtCtrls.

Topics

Name	Description
Classes (see page 162)	The following table lists classes in this documentation.
Functions (see page 1536)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1538)	The following table lists structs, records, enums in this documentation.
Types (see page 1547)	The following table lists types in this documentation.
Constants (see page 1556)	The following table lists constants in this documentation.

Classes

	Name	Description
	EGridPanelException (see page 163)	The base class for runtime exceptions that occur in grid panels.
	TBevel (see page 163)	TBevel represents a beveled outline.
	TBoundLabel (see page 174)	TBoundLabel is the type of a labeled edit control's label.
	TButtonedEdit (see page 220)	TButtonedEdit is an edit control that has two embedded buttons.
	TCategoryPanel (see page 309)	TCategoryPanel is used to hold the set of display control legends for a particular display category.
	TCategoryPanelGroup (see page 379)	TCategoryPanelGroup is a container for collapsible panels that are TCategoryPanel (see page 309) objects.
	TCategoryPanelSurface (see page 456)	This is class ExtCtrls.TCategoryPanelSurface.
	TCellCollection (see page 458)	A collection of TCellItem (see page 462) objects.
	TCellItem (see page 462)	Represents an item in a TCellCollection (see page 458).
	TColorBox (see page 466)	TColorBox represents a combo box that lets users select a color.
	TColorListBox (see page 525)	TColorListBox represents a list box that lets users select a color.
	TColumnCollection (see page 587)	Controls the **Collection Editor** for columns within a grid panel.
	TColumnItem (see page 590)	This is class ExtCtrls.TColumnItem.
	TControlBar (see page 590)	TControlBar manages the layout of toolbar components.
	TControlCollection (see page 663)	This is class ExtCtrls.TControlCollection.
	TControlItem (see page 668)	This is class ExtCtrls.TControlItem.
	TCustomButtonedEdit (see page 673)	TCustomButtonedEditis the base class for edit controls that contain embedded buttons.
	TCustomCategoryPanel (see page 698)	TCustomCategoryPanel is a base class for the TCategoryPanel (see page 309) class.
	TCustomCategoryPanelGroup (see page 718)	This is class ExtCtrls.TCustomCategoryPanelGroup.
	TCustomColorBox (see page 735)	TCustomColorBox represents a combo box that lets users select a color.

	TCustomColorListBox (see page 745)	This is class ExtCtrls.TCustomColorListBox.
	TCustomControlBar (see page 754)	TCustomControlBar is the base class for control bar components.
	TCustomFlowPanel (see page 774)	The base class for all flow panel components.
	TCustomGridPanel (see page 781)	The base class for all grid panel components.
	TCustomLabeledEdit (see page 790)	TCustomLabeledEdit is the base class for edit controls that have an integrated label.
	TCustomLinkLabel (see page 797)	Base class for label containing HTML link.
	TCustomPanel (see page 802)	TCustomPanel is the base class for all panel components.
	TCustomRadioGroup (see page 815)	TCustomRadioGroup is the base class for radio-group components.
	TCustomTrayIcon (see page 822)	The base class for TTrayIcon (see page 1528).
	TEditButton (see page 835)	Button in a TButtonedEdit (see page 220).
	TFlowPanel (see page 857)	TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.
	TGridPanel (see page 941)	TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.
	THeader (see page 1021)	THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.
	TImage (see page 1048)	TImage displays a graphical image.
	TLabeledEdit (see page 1100)	TLabeledEdit is an edit control that has an associated label.
	TLinkLabel (see page 1167)	Label containing an HTML link.
	TNotebook (see page 1211)	TNotebook displays multiple pages, each with its own set of controls.
	TPage (see page 1267)	This is class ExtCtrls.TPage.
	TPaintBox (see page 1279)	TPaintBox provides a canvas that applications can use for rendering an image.
	TPanel (see page 1326)	TPanel implements a generic panel control.
	TRadioGroup (see page 1406)	TRadioGroup represents a group of radio buttons that function together.
	TRowCollection (see page 1455)	Controls the **Collection Editor** for rows within a grid panel.
	TRowItem (see page 1457)	This is class ExtCtrls.TRowItem.
	TShape (see page 1458)	TShape represents a geometric shape that can be drawn on a form.
	TSplitter (see page 1492)	TSplitter divides the client area of a form into resizable panes.
	TTimer (see page 1520)	TTimer encapsulates the Windows API timer functions.
	TTrayIcon (see page 1528)	Creates an icon in the system tray next to the clock.

Constants

Name	Description
DefaultCategoryPanelHeight (see page 1556)	This is constant ExtCtrls.DefaultCategoryPanelHeight.
NoColorSelected (see page 1556)	This is constant ExtCtrls.NoColorSelected.
WM_SYSTEM_TRAY_MESSAGE (see page 1556)	This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

Functions

	Name	Description
	Frame3D (see page 1536)	Draws a three dimensional frame on a canvas.
	NotebookHandlesNeeded (see page 1537)	This is function ExtCtrls.NotebookHandlesNeeded.

Structs, Records, Enums

	Name	Description
	TBalloonFlags (see page 1538)	A type specified when showing a balloon hint that puts one of the default icons on the balloon message.
	TBandDrawingStyle (see page 1539)	This is record ExtCtrls.TBandDrawingStyle.
	TBandPaintOption (see page 1539)	TBandPaintOption and TBandPaintOptions (see page 1549) specify which parts of a band are painted.
	TBevelShape (see page 1540)	TBevelShape determines the overall shape of a bevel control.
	TBevelStyle (see page 1541)	TBevelStyle indicates whether a bevel appears raised or lowered.
	TColorBoxStyles (see page 1541)	This is record ExtCtrls.TColorBoxStyles.
	TCornerEdge (see page 1542)	The size of the cornering for the edge of the bands in a TControlBar (see page 590).
	TExpandStyle (see page 1542)	Specifies how a grid panel expands.
	TFlowStyle (see page 1543)	This is record ExtCtrls.TFlowStyle.

	TLabelPosition (see page 1543)	TLabelPosition specifies the position of the label in a labeled edit control.
	TResizeStyle (see page 1544)	This is record ExtCtrls.TResizeStyle.
	TShapeType (see page 1544)	TShapeType determines the shape of a TShape (see page 1458) control.
	TSizeStyle (see page 1545)	This is record ExtCtrls.TSizeStyle.
	TSysLinkType (see page 1546)	TSysLinkType describes an HTML link type.

Types

Name	Description
NaturalNumber (see page 1547)	This is type ExtCtrls.NaturalNumber.
TBandDragEvent (see page 1548)	This is type ExtCtrls.TBandDragEvent.
TBandInfoEvent (see page 1548)	This is type ExtCtrls.TBandInfoEvent.
TBandMoveEvent (see page 1548)	This is type ExtCtrls.TBandMoveEvent.
TBandPaintEvent (see page 1549)	This is type ExtCtrls.TBandPaintEvent.
TBandPaintOptions (see page 1549)	This is type ExtCtrls.TBandPaintOptions.
TBeginBandMoveEvent (see page 1549)	This is type ExtCtrls.TBeginBandMoveEvent.
TCanResizeEvent (see page 1550)	This is type ExtCtrls.TCanResizeEvent.
TCategoryPanelClass (see page 1550)	This is type ExtCtrls.TCategoryPanelClass.
TCategoryPanelSurfaceClass (see page 1550)	This is type ExtCtrls.TCategoryPanelSurfaceClass.
TCellSpan (see page 1551)	This is type ExtCtrls.TCellSpan.
TColorBoxStyle (see page 1551)	TColorBoxStyle and TColorBoxStyles (see page 1541) specify how a color box control displays its selections.
TEditButtonClass (see page 1552)	This is type ExtCtrls.TEditButtonClass.
TEndBandMoveEvent (see page 1552)	This is type ExtCtrls.TEndBandMoveEvent.
TGetColorsEvent (see page 1552)	This is type ExtCtrls.TGetColorsEvent.
TLBGetColorsEvent (see page 1553)	This is type ExtCtrls.TLBGetColorsEvent.
TPanelBevel (see page 1553)	TPanelBevel specifies a bevel's cut.
TRowSize (see page 1554)	This is type ExtCtrls.TRowSize.
TSectionEvent (see page 1554)	TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 1021) control is resized.
TSplitterCanResizeEvent (see page 1554)	This is type ExtCtrls.TSplitterCanResizeEvent.
TSysLinkEvent (see page 1555)	Event routine for click on TCustomLinkLabel (see page 797) HTML link.

3.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EGridPanelException (see page 163)	The base class for runtime exceptions that occur in grid panels.
	TBevel (see page 163)	TBevel represents a beveled outline.
	TBoundLabel (see page 174)	TBoundLabel is the type of a labeled edit control's label.
	TButtonedEdit (see page 220)	TButtonedEdit is an edit control that has two embedded buttons.
	TCategoryPanel (see page 309)	TCategoryPanel is used to hold the set of display control legends for a particular display category.
	TCategoryPanelGroup (see page 379)	TCategoryPanelGroup is a container for collapsible panels that are TCategoryPanel (see page 309) objects.
	TCategoryPanelSurface (see page 456)	This is class ExtCtrls.TCategoryPanelSurface.
	TCellCollection (see page 458)	A collection of TCellItem (see page 462) objects.
	TCellItem (see page 462)	Represents an item in a TCellCollection (see page 458).
	TColorBox (see page 466)	TColorBox represents a combo box that lets users select a color.
	TColorListBox (see page 525)	TColorListBox represents a list box that lets users select a color.
	TColumnCollection (see page 587)	Controls the **Collection Editor** for columns within a grid panel.
	TColumnItem (see page 590)	This is class ExtCtrls.TColumnItem.
	TControlBar (see page 590)	TControlBar manages the layout of toolbar components.
	TControlCollection (see page 663)	This is class ExtCtrls.TControlCollection.
	TControlItem (see page 668)	This is class ExtCtrls.TControlItem.
	TCustomButtonedEdit (see page 673)	TCustomButtonedEditis the base class for edit controls that contain embedded buttons.
	TCustomCategoryPanel (see page 698)	TCustomCategoryPanel is a base class for the TCategoryPanel (see page 309) class.
	TCustomCategoryPanelGroup (see page 718)	This is class ExtCtrls.TCustomCategoryPanelGroup.
	TCustomColorBox (see page 735)	TCustomColorBox represents a combo box that lets users select a color.
	TCustomColorListBox (see page 745)	This is class ExtCtrls.TCustomColorListBox.
	TCustomControlBar (see page 754)	TCustomControlBar is the base class for control bar components.
	TCustomFlowPanel (see page 774)	The base class for all flow panel components.
	TCustomGridPanel (see page 781)	The base class for all grid panel components.
	TCustomLabeledEdit (see page 790)	TCustomLabeledEdit is the base class for edit controls that have an integrated label.
	TCustomLinkLabel (see page 797)	Base class for label containing HTML link.
	TCustomPanel (see page 802)	TCustomPanel is the base class for all panel components.
	TCustomRadioGroup (see page 815)	TCustomRadioGroup is the base class for radio-group components.
	TCustomTrayIcon (see page 822)	The base class for TTrayIcon (see page 1528).
	TEditButton (see page 835)	Button in a TButtonedEdit (see page 220).
	TFlowPanel (see page 857)	TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.
	TGridPanel (see page 941)	TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.
	THeader (see page 1021)	THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.
	TImage (see page 1048)	TImage displays a graphical image.
	TLabeledEdit (see page 1100)	TLabeledEdit is an edit control that has an associated label.
	TLinkLabel (see page 1167)	Label containing an HTML link.
	TNotebook (see page 1211)	TNotebook displays multiple pages, each with its own set of controls.
	TPage (see page 1267)	This is class ExtCtrls.TPage.
	TPaintBox (see page 1279)	TPaintBox provides a canvas that applications can use for rendering an image.

	TPanel (see page 1326)	TPanel implements a generic panel control.
	TRadioGroup (see page 1406)	TRadioGroup represents a group of radio buttons that function together.
	TRowCollection (see page 1455)	Controls the **Collection Editor** for rows within a grid panel.
	TRowItem (see page 1457)	This is class ExtCtrls.TRowItem.
	TShape (see page 1458)	TShape represents a geometric shape that can be drawn on a form.
	TSplitter (see page 1492)	TSplitter divides the client area of a form into resizable panes.
	TTimer (see page 1520)	TTimer encapsulates the Windows API timer functions.
	TTrayIcon (see page 1528)	Creates an icon in the system tray next to the clock.

3.1.1 EGridPanelException Class

The base class for runtime exceptions that occur in grid panels.

Class Hierarchy

Pascal

```
EGridPanelException = class(Exception);
```

C++

```
class EGridPanelException : public Exception;
```

File

ExtCtrls

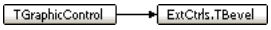
Description

Use EGridPanelException as the base class for creating custom exceptions that occur in grid panels.

3.1.2 TBevel Class

TBevel represents a beveled outline.

Class Hierarchy



Pascal

```
TBevel = class(TGraphicControl);
```

C++

```
class TBevel : public TGraphicControl;
```

File

ExtCtrls

Description

Use TBevel to create beveled boxes, frames, or lines. The bevel can appear raised or lowered.

See Also

TShape (see page 1458)

Frame3D (see page 1536)

Bevels

Graphic Controls

3.1.2.1 TBevel Members

The following tables list the members exposed by TBevel.

Protected Methods

	Name	Description
	Paint (see page 165)	Render control surface.

Public Methods

	Name	Description
	Create (see page 164)	Creates and initializes an instance of TBevel (see page 163).

Published Properties

	Name	Description
	Align (see page 165)	Determines how the control aligns within its container (parent control).
	Anchors (see page 169)	Specifies how the control is anchored to its parent.
	Constraints (see page 169)	Specifies the size constraints for the control.
	ParentShowHint (see page 170)	Determines where a control looks to find out if its Help Hint should be shown.
	Shape (see page 170)	Determines the shape of the bevel.
	ShowHint (see page 171)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 172)	Determines whether the bevel appears raised or lowered.
	Visible (see page 173)	Determines whether the component appears onscreen.

3.1.2.2 TBevel.Create Constructor

Creates and initializes an instance of TBevel (see page 163).

Pascal

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

C++

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

Description

Call Create to instantiate a bevel at runtime. Bevels placed on forms at design time are created automatically.

See Also

Shape (see page 170)

Style (see page 172)

3.1.2.3 TBevel Methods

The methods of the TBevel class are listed here.

Protected Methods

	Name	Description
	Paint (see page 165)	Render control surface.

3.1.2.3.1 TBevel.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

3.1.2.4 TBevel Properties

The properties of the TBevel class are listed here.

Published Properties

	Name	Description
	Align (see page 165)	Determines how the control aligns within its container (parent control).
	Anchors (see page 169)	Specifies how the control is anchored to its parent.
	Constraints (see page 169)	Specifies the size constraints for the control.
	ParentShowHint (see page 170)	Determines where a control looks to find out if its Help Hint should be shown.
	Shape (see page 170)	Determines the shape of the bevel.
	ShowHint (see page 171)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 172)	Determines whether the bevel appears raised or lowered.
	Visible (see page 173)	Determines whether the component appears onscreen.

3.1.2.4.1 TBevel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it

continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
```

```
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
```

```
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.2.4.2 TBevel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.2.4.3 TBevel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.2.4.4 TBevel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.2.4.5 TBevel.Shape Property

Determines the shape of the bevel.

Pascal

```
property Shape: TBevelShape;
```

C++

```
__property TBevelShape Shape;
```

Description

Set Shape to specify whether the bevel appears as a line, box, frame, or space. For shapes that can appear either raised or lowered, the Style (see page 172) property indicates which effect is used.

The default value for Shape is bsBox.

See Also

ClientRect

Style (see page 172)

3.1.2.4.6 TBevel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
```

```
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.2.4.7 TBevel.Style Property

Determines whether the bevel appears raised or lowered.

Pascal

```
property Style: TBevelStyle;
```

C++

```
__property TBevelStyle Style;
```

Description

Set Style to indicate whether the bevel should create a raised or a lowered effect. When the Shape (see page 170) property is

bsBox, the entire client area appears raised or lowered. For all other values of Shape (see page 170), the bevel displays a raised or lowered line along the edge or edges of the client area. The default value of Style is bsLowered.

See Also

Shape (see page 170)

3.1.2.4.8 TBevel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.3 TBoundLabel Class

TBoundLabel is the type of a labeled edit control's label.

Class Hierarchy

Pascal

```
  TBoundLabel = class(TCustomLabel);
```

C++

```
  class TBoundLabel : public TCustomLabel;
```

File

ExtCtrls

Description

TBoundLabel implements the label of a labeled edit control. Unlike other label controls such as TLabel, TBoundLabel "belongs" to the labeled edit control (it is a subcomponent of the edit control). This means that the bound label's Owner is the edit control

rather than the form, and the edit control is responsible for loading and saving the bound label.

Use the properties of TBoundLabel to specify how you want the labeled edit control's label to appear. For example, you can use the Caption (see page 178) property to specify the text that appears in the label. You can use the Alignment and Layout (see page 186) properties to specify the placement of that caption within the label.

See Also

TLabeledEdit (see page 1100)

TLabel

TComponent

3.1.3.1 TBoundLabel Members

The following tables list the members exposed by TBoundLabel.

Protected Methods

	Name	Description
	AdjustBounds (see page 176)	Resizes the label when the text changes.

Public Methods

	Name	Description
	Create (see page 176)	Creates a new TBoundLabel (see page 174) object.

Published Events

	Name	Description
	OnClick (see page 201)	Occurs when the user clicks the control.
	OnContextPopup (see page 203)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 203)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 204)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 207)	Occurs when the user drags an object over a control.
	OnEndDock (see page 210)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 210)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 211)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 212)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 214)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 217)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 219)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 220)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	BiDiMode (see page 178)	Specifies the bi-directional mode for the control.
	Caption (see page 178)	Specifies a text string that identifies the control to the user.
	Color (see page 181)	Specifies the background color of the control.
	DragCursor (see page 181)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 182)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 182)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 184)	Controls the attributes of text written on or in the control.
	Height (see page 185)	Specifies the vertical size of the control in pixels.
	Layout (see page 186)	Specifies the vertical placement of the text within the label.
R	Left (see page 186)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ParentBiDiMode (see page 187)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 188)	Determines where a control looks for its color information.
	ParentFont (see page 188)	Determines where a control looks for its font information.
	ParentShowHint (see page 189)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 189)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 194)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 195)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
R	Top (see page 197)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 198)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Width (see page 198)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 200)	Specifies whether the label text wraps when it is too long for the width of the label.

3.1.3.2 TBoundLabel.Create Constructor

Creates a new TBoundLabel (see page 174) object.

Pascal

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

C++

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

Description

Do not create instances of TBoundLabel (see page 174). Bound labels are instantiated by the labeled edit controls to which they belong.

AOwner specifies the labeled edit control that acts as the bound label's Owner.

See Also

TLabeledEdit (see page 1100)

Owner

3.1.3.3 TBoundLabel Methods

The methods of the TBoundLabel class are listed here.

Protected Methods

	Name	Description
V	AdjustBounds (see page 176)	Resizes the label when the text changes.

3.1.3.3.1 TBoundLabel.AdjustBounds Method

Resizes the label when the text changes.

Pascal

```
procedure AdjustBounds; override;
```

C++

```
virtual __fastcall AdjustBounds();
```

Description

When the text in the label or the font changes, AdjustBounds is implicitly called. It redraws the text in the client area, and then resizes the label if the AutoSize property is true.

When WordWrap is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap is false, the height of the label is determined by the font, and changes to the text cause the label to change in width.

Override AdjustBounds to make additional adjustments when the text changes.

See Also

DoDrawText

AutoSize

3.1.3.4 TBoundLabel Properties

The properties of the TBoundLabel class are listed here.

Published Properties

	Name	Description
	BiDiMode (see page 178)	Specifies the bi-directional mode for the control.
	Caption (see page 178)	Specifies a text string that identifies the control to the user.
	Color (see page 181)	Specifies the background color of the control.
	DragCursor (see page 181)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 182)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 182)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 184)	Controls the attributes of text written on or in the control.
	Height (see page 185)	Specifies the vertical size of the control in pixels.
	Layout (see page 186)	Specifies the vertical placement of the text within the label.
R	Left (see page 186)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ParentBiDiMode (see page 187)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 188)	Determines where a control looks for its color information.
	ParentFont (see page 188)	Determines where a control looks for its font information.
	ParentShowHint (see page 189)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 189)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 194)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 195)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
R	Top (see page 197)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 198)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Width (see page 198)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 200)	Specifies whether the label text wraps when it is too long for the width of the label.

3.1.3.4.1 TBoundLabel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.3.4.2 TBoundLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
```

```
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
```

```
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long to be the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
  This example creates a ToolBar and adds ToolButtons to it.
  Include ComCtlrs in the uses clause.
  }
  procedure AddButtons(ToolBar: TToolBar;
    Const ButtonCaptions: array of String);
  var
    I: Integer;
  begin
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
```

```
end;
```

3.1.3.4.3 TBoundLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.3.4.4 TBoundLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.3.4.5 TBoundLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.3.4.6 TBoundLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
```

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.3.4.7 TBoundLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
```

```
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.3.4.8 TBoundLabel.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

3.1.3.4.9 TBoundLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout;
```

C++

```
__property Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment property.

See Also

Alignment

ClientRect

3.1.3.4.10 TBoundLabel.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top

3.1.3.4.11 TBoundLabel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.3.4.12 TBoundLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.3.4.13 TBoundLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.3.4.14 TBoundLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.3.4.15 TBoundLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
```

```
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.3.4.16 TBoundLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar;
```

C++

```
__property ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
```

```
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl

GetLabelText

3.1.3.4.17 TBoundLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.3.4.18 TBoundLabel.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    dynamic_cast<TWinControl *>(GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
```

```
    begin
      TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
    end;
  end;

  procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
  var
    I: Integer;
    ChildControl: TControl;
  begin
    for I:= 0 to GroupBox1.ControlCount -1 do
    begin
      ChildControl := GroupBox1.Controls[I];
      if Button = btNext then
        ChildControl.Top := ChildControl.Top + 15
      else
        ChildControl.Top := ChildControl.Top - 15;
    end;
  end;
```

See Also

Left

3.1.3.4.19 TBoundLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent;
```

C++

```
__property Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color

ParentColor

3.1.3.4.20 TBoundLabel.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
```

```
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

3.1.3.4.21 TBoundLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap;
```

C++

```
__property WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth

GetLabelText

3.1.3.5 TBoundLabel Events

The events of the TBoundLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 201)	Occurs when the user clicks the control.
	OnContextPopup (see page 203)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 203)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 204)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 207)	Occurs when the user drags an object over a control.

	OnEndDock (see page 210)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 210)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 211)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 212)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 214)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 217)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 219)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 220)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.3.5.1 TBoundLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.3.5.2 TBoundLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.3.5.3 TBoundLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.3.5.4 TBoundLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.3.5.5 TBoundLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.3.5.6 TBoundLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.3.5.7 TBoundLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
```

```
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.3.5.8 TBoundLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants

to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.3.5.9 TBoundLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
```

```
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.3.5.10 TBoundLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
  property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.3.5.11 TBoundLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.3.5.12 TBoundLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.3.5.13 TBoundLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.4 TButtonedEdit Class

TButtonedEdit is an edit control that has two embedded buttons.

Class Hierarchy

Pascal

```
TButtonedEdit = class(TCustomButtonedEdit);
```

C++

```
class TButtonedEdit : public TCustomButtonedEdit;
```

File

ExtCtrls

Description

Use a TButtonedEdit to add an edit control with two (optional) embedded buttons, one on either side of the text, to a form. TButtonedEdit uses the Images (see page 244) property to supply images from a TImageList to the embedded buttons, referenced by its LeftButton (see page 248) and RightButton (see page 261) properties. Use each button's ImageIndex, HotImageIndex, PressedImageIndex and DisabledImageIndex properties to specify images for each possible state.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
              mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
              mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
```

```
    btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

Images (see page 244)

LeftButton (see page 248)

RightButton (see page 261)

Alignment (see page 230)

NumbersOnly (see page 252)

TextHint (see page 268)

TEditButton (see page 835)

3.1.4.1 TButtonedEdit Members

The following tables list the members exposed by TButtonedEdit.

Protected Methods

	Name	Description
	DoSetTextHint (see page 676)	This is DoSetTextHint, a member of class TCustomButtonedEdit.
	GetEditButtonClass (see page 676)	This is GetEditButtonClass, a member of class TCustomButtonedEdit.
	Notification (see page 676)	Responds to notifications that components are being created or destroyed.
	UpdateEditMargins (see page 677)	Update the margins of the edit control to provide adequate space for the buttons.
	WndProc (see page 677)	Provides specific message responses for the control.

Public Methods

	Name	Description
	Create (see page 674)	Creates an instance of TCustomEdit.
	DefaultHandler (see page 675)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 675)	Destroys an instance of TWinControl.

Published Events

	Name	Description
	OnChange (see page 270)	Occurs when the text for the edit control may have changed.
	OnClick (see page 271)	Occurs when the user clicks the control.
	OnContextPopup (see page 273)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 273)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 274)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 277)	Occurs when the user drags an object over a control.
	OnEndDock (see page 280)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 280)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 281)	Occurs when a control receives the input focus.
	OnExit (see page 284)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 287)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 290)	Occurs when a key is pressed.
	OnKeyUp (see page 291)	Occurs when the user releases a key that has been pressed.
	OnLeftButtonClick (see page 293)	Occurs when the user clicks the LeftButton.
	OnMouseActivate (see page 296)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 297)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 299)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 300)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 300)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 302)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnRightButtonClick (see page 304)	Occurs when the user clicks the RightButton.
	OnStartDock (see page 307)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnStartDrag (see page 308)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 227)	Determines how the control aligns within its container (parent control).
	Alignment (see page 230)	Determines how the text is aligned within the text edit control.
	Anchors (see page 231)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 231)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 232)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 233)	Specifies which edges of the control are beveled.
	BevelInner (see page 233)	Specifies the cut of the inner bevel.
	BevelKind (see page 234)	Specifies the control's bevel style.
	BevelOuter (see page 234)	Specifies the cut of the outer bevel.
	BevelWidth (see page 234)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 235)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 235)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 236)	Determines the case of the text within the edit control.
	Color (see page 236)	Specifies the background color of the control.
	Constraints (see page 237)	Specifies the size constraints for the control.
	Ctl3D (see page 237)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 238)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 240)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 240)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 240)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 242)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 242)	Controls the attributes of text written on or in the control.
	HideSelection (see page 244)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Images (see page 244)	Lists the images that can appear on the control's embedded buttons.
	ImeMode (see page 247)	Determines the behavior of the input method editor (IME).
	ImeName (see page 248)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LeftButton (see page 248)	Specifies the control that implements the left embedded button.
	MaxLength (see page 252)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 252)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 252)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 253)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 253)	Determines where a control looks for its color information.
	ParentCtl3D (see page 254)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 254)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 254)	Determines where a control looks for its font information.
	ParentShowHint (see page 255)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 255)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 256)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 261)	Determines whether the user can change the text of the edit control.
	RightButton (see page 261)	Specifies the control that implements the right embedded button.

	ShowHint (see page 264)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 266)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 266)	Determines if the user can tab to a control.
	Text (see page 267)	Contains a text string associated with the control.
	TextHint (see page 268)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 268)	Determines whether the component appears onscreen.

3.1.4.2 TButtonedEdit Properties

The properties of the TButtonedEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 227)	Determines how the control aligns within its container (parent control).
	Alignment (see page 230)	Determines how the text is aligned within the text edit control.
	Anchors (see page 231)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 231)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 232)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 233)	Specifies which edges of the control are beveled.
	BevelInner (see page 233)	Specifies the cut of the inner bevel.
	BevelKind (see page 234)	Specifies the control's bevel style.
	BevelOuter (see page 234)	Specifies the cut of the outer bevel.
	BevelWidth (see page 234)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 235)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 235)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 236)	Determines the case of the text within the edit control.
	Color (see page 236)	Specifies the background color of the control.
	Constraints (see page 237)	Specifies the size constraints for the control.
	Ctl3D (see page 237)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 238)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 240)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 240)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 240)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 242)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 242)	Controls the attributes of text written on or in the control.
	HideSelection (see page 244)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Images (see page 244)	Lists the images that can appear on the control's embedded buttons.
	ImeMode (see page 247)	Determines the behavior of the input method editor (IME).
	ImeName (see page 248)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LeftButton (see page 248)	Specifies the control that implements the left embedded button.
	MaxLength (see page 252)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 252)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 252)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 253)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 253)	Determines where a control looks for its color information.

	ParentCtl3D (see page 254)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 254)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 254)	Determines where a control looks for its font information.
	ParentShowHint (see page 255)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 255)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 256)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 261)	Determines whether the user can change the text of the edit control.
	RightButton (see page 261)	Specifies the control that implements the right embedded button.
	ShowHint (see page 264)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 266)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 266)	Determines if the user can tab to a control.
	Text (see page 267)	Contains a text string associated with the control.
	TextHint (see page 268)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 268)	Determines whether the component appears onscreen.

3.1.4.2.1 TButtonedEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
```

```
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
```

```
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
```

```
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.4.2.2 TButtonedEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment;
```

C++

```
__property Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment

Text

Displaying and Editing Fields in an Edit Box

3.1.4.2.3 TButtonedEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.4.2.4 TButtonedEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect;
```

C++

```
__property AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll

SelLength

SelStart

SelText

SetSelTextBuf

3.1.4.2.5 TButtonedEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle

ClientHeight

ControlStyle

Height

3.1.4.2.6 TButtonedEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.4.2.7 TButtonedEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.4.2.8 TButtonedEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.4.2.9 TButtonedEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.4.2.10 TButtonedEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

3.1.4.2.11 TButtonedEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.4.2.12 TButtonedEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If

BorderStyle is bsNone, there will be no border.

3.1.4.2.13 TButtonedEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase;
```

C++

```
__property CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

3.1.4.2.14 TButtonedEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.4.2.15 TButtonedEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.4.2.16 TButtonedEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.4.2.17 TButtonedEdit.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
```

```
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.4.2.18 TButtonedEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.4.2.19 TButtonedEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.4.2.20 TButtonedEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.4.2.21 TButtonedEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.4.2.22 TButtonedEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.4.2.23 TButtonedEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection;
```

C++

```
__property HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength

SelStart

SelText

SetSelTextBuf

3.1.4.2.24 TButtonedEdit.Images Property

Lists the images that can appear on the control's embedded buttons.

Pascal

```
property Images: TImageList;
```

C++

```
__property TImageList Images;
```

Description

Images contains a list of images that can appear on the control's embedded buttons. If Images is unassigned, no images are displayed on the buttons.

Images in the list should be 16x16.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;
```

```
    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
              mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
```

```
  Entering the control will enable the right button, left button will
  display a MessageDlg and the right button will clear the input text.
  }
  type
    TForm4 = class(TForm)
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
      btnEdit : TButtonedEdit;
      procedure btnEditEnter(Sender : Tobject);
      procedure btnEditLeftClick(Sender : Tobject);
      procedure btnEditRightClick(Sender : Tobject);
    public
      { Public declarations }
    end;

  var
    Form4: TForm4;

  implementation

  {$R *.dfm}

  procedure TForm4.btnEditEnter(Sender: Tobject);
  begin
    btnEdit.Text := '';
    btnEdit.RightButton.Visible := true;
  end;

  procedure TForm4.btnEditLeftClick(Sender: Tobject);
  begin
    MessageDlg('Left button pressed with: ' + btnEdit.Text,
               mtInformation, [mbYes], 0);

  end;

  procedure TForm4.btnEditRightClick(Sender: Tobject);
  begin
    //resetting to default state
     btnEdit.Text := 'Default';
     btnEdit.RightButton.Visible := false;
  end;

  procedure TForm4.FormCreate(Sender: TObject);
  var
    imList : TImageList;
    bmap   : TBitmap;
  begin
    //declaring an instance
      btnEdit := TButtonedEdit.Create(Self);

      //setting the parent so it will autodetele
      btnEdit.Parent := Self;

      //positioning
      btnEdit.Left := 10;
      btnEdit.Top := 10;

      //creating a TImageList and using it for the button
      imList := TImageList.Create(Self);
      bmap := TBitmap.Create();
      bmap.LoadFromFile('littleB_16.bmp');
      imList.Add(bmap,nil);
      bmap.LoadFromFile('little_logo_16.bmp');
      imList.Add(bmap,nil);
      btnEdit.Images := imList;
```

```
    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

TImageList

LeftButton (see page 685)

RightButton (see page 688)

3.1.4.2.25 TButtonedEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

3.1.4.2.26 TButtonedEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

3.1.4.2.27 TButtonedEdit.LeftButton Property

Specifies the control that implements the left embedded button.

Pascal

```
property LeftButton: TEditButton;
```

C++

```
__property TEditButton LeftButton;
```

Description

Use LeftButton to work with the left button embedded in this edit control. Use this button's properties to specify images to correspond to the various states of the button, to disable the button, to toggle it's visiblity and to add a dropdown menu. At design time, expand the LeftButton property in the Object Inspector to set its properties.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
```

```
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

Images (see page 682)

RightButton (see page 688)

OnLeftButtonClick (see page 692)

TImageList

TEditButton (see page 835)

3.1.4.2.28 TButtonedEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength;
```

C++

```
__property MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

3.1.4.2.29 TButtonedEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly;
```

C++

```
__property NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

3.1.4.2.30 TButtonedEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert;
```

C++

```
__property OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

3.1.4.2.31 TButtonedEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.4.2.32 TButtonedEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.4.2.33 TButtonedEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.4.2.34 TButtonedEdit.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.4.2.35 TButtonedEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own

Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.4.2.36 TButtonedEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.4.2.37 TButtonedEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar;
```

C++

```
__property PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

3.1.4.2.38 TButtonedEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
```

```
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
```

```
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.4.2.39 TButtonedEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard

GetSelTextBuf

3.1.4.2.40 TButtonedEdit.RightButton Property

Specifies the control that implements the right embedded button.

Pascal

```
property RightButton: TEditButton;
```

C++

```
__property TEditButton RightButton;
```

Description

Use RightButton to work with the right button embedded in this edit control. Use this button's properties to specify images to correspond to the various states of the button, to disable the button, to toggle it's visiblity and to add a dropdown menu. At design time, expand the RightButton property in the Object Inspector to set its properties.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
```

```
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
```

```
            mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;
```

```
    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

Images (see page 682)

LeftButton (see page 685)

OnRightButtonClick (see page 695)

TImageList

TEditButton (see page 835)

3.1.4.2.41 TButtonedEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine

whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
```

```
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.4.2.42 TButtonedEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.4.2.43 TButtonedEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.4.2.44 TButtonedEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines

3.1.4.2.45 TButtonedEdit.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint;
```

C++

```
__property TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

3.1.4.2.46 TButtonedEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.4.3 TButtonedEdit Events

The events of the TButtonedEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 270)	Occurs when the text for the edit control may have changed.
	OnClick (see page 271)	Occurs when the user clicks the control.
	OnContextPopup (see page 273)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 273)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 274)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 277)	Occurs when the user drags an object over a control.
	OnEndDock (see page 280)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 280)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 281)	Occurs when a control receives the input focus.
	OnExit (see page 284)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 287)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 290)	Occurs when a key is pressed.
	OnKeyUp (see page 291)	Occurs when the user releases a key that has been pressed.
	OnLeftButtonClick (see page 293)	Occurs when the user clicks the LeftButton.
	OnMouseActivate (see page 296)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 297)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 299)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 300)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 300)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 302)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnRightButtonClick (see page 304)	Occurs when the user clicks the RightButton.
	OnStartDock (see page 307)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 308)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.4.3.1 TButtonedEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change

Text

3.1.4.3.2 TButtonedEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.4.3.3 TButtonedEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.4.3.4 TButtonedEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.4.3.5 TButtonedEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.4.3.6 TButtonedEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.4.3.7 TButtonedEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.4.3.8 TButtonedEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.4.3.9 TButtonedEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.4.3.10 TButtonedEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));
```

```
    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.4.3.11 TButtonedEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

3.1.4.3.12 TButtonedEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

3.1.4.3.13 TButtonedEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

3.1.4.3.14 TButtonedEdit.OnLeftButtonClick Event

Occurs when the user clicks the LeftButton.

Pascal

```
property OnLeftButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLeftButtonClick;
```

Description

Use the OnLeftButtonClick event handler to respond when the user clicks the LeftButton of the control.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
```

```
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
```

```
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
```

```
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

LeftButton (see page 685)

OnRightButtonClick (see page 695)

TEditButton (see page 835)

3.1.4.3.15 TButtonedEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of

whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.4.3.16 TButtonedEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.4.3.17 TButtonedEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.4.3.18 TButtonedEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.4.3.19 TButtonedEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.4.3.20 TButtonedEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.4.3.21 TButtonedEdit.OnRightButtonClick Event

Occurs when the user clicks the RightButton.

Pascal

```
property OnRightButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRightButtonClick;
```

Description

Use the OnRightButtonClick event handler to respond when the user clicks the RightButton of the control.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}
```

```
void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
```

```
    //declaring an instance
      btnEdit := TButtonedEdit.Create(Self);

      //setting the parent so it will autodetele
      btnEdit.Parent := Self;

      //positioning
      btnEdit.Left := 10;
      btnEdit.Top := 10;

      //creating a TImageList and using it for the button
      imList := TImageList.Create(Self);
      bmap := TBitmap.Create();
      bmap.LoadFromFile('littleB_16.bmp');
      imList.Add(bmap,nil);
      bmap.LoadFromFile('little_logo_16.bmp');
      imList.Add(bmap,nil);
      btnEdit.Images := imList;

      //selecting the image for the left button and enabling it
      btnEdit.LeftButton.Visible := true;
      btnEdit.LeftButton.Enabled := true;
      btnEdit.LeftButton.ImageIndex := 0;
      btnEdit.LeftButton.DisabledImageIndex := 0;
      btnEdit.LeftButton.HotImageIndex := 0;

      //selecting the image for the right button
      //this will be enabled when the content changes
      btnEdit.RightButton.Enabled := true;
      btnEdit.RightButton.ImageIndex := 1;
      btnEdit.RightButton.DisabledImageIndex := 1;
      btnEdit.RightButton.HotImageIndex := 1;

      //selecting action for the buttoned edit to take
      btnEdit.OnEnter := btnEditEnter;
      btnEdit.OnRightButtonClick := btnEditRightClick;
      btnEdit.OnLeftButtonClick := btnEditLeftClick;

      //default text in the edit
      btnEdit.Text := 'Default';

      btnEdit.Constraints.MinHeight := 16;
      btnEdit.Height := 16;

      //finally displaying the button
      btnEdit.Show();

  end;
```

See Also

RightButton (see page 688)

OnLeftButtonClick (see page 692)

TEditButton (see page 835)

3.1.4.3.22 TButtonedEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.4.3.23 TButtonedEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.5 TCategoryPanel Class

TCategoryPanel is used to hold the set of display control legends for a particular display category.

Class Hierarchy

Pascal

```
TCategoryPanel = class(TCustomCategoryPanel);
```

C++

```
class TCategoryPanel : public TCustomCategoryPanel;
```

File

ExtCtrls

Description

TCategoryPanel is used to hold the set of display control legends for a particular display category.

See Also

TCategoryPanelGroup (see page 379)

3.1.5.1 TCategoryPanel Members

The following tables list the members exposed by TCategoryPanel.

Enumerations

	Name	Description
	THeaderState (see page 701)	THeaderState is the type of the HeaderState (see page 717) property for a category panel.

Protected Fields

	Name	Description
	FPanelSurface (see page 702)	This is FPanelSurface, a member of class TCustomCategoryPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	CalcRectBounds (see page 703)	Sets the rectangles used to display the text and glyphs of the category panel.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.

	DefineProperties (see page 703)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DrawCollapsedPanel (see page 704)	Displays the category panel in the collapsed state.
	DrawHeader (see page 704)	This is the overview for the DrawHeader method overload.
	DrawHeaderBackground (see page 705)	This is DrawHeaderBackground, a member of class TCustomCategoryPanel.
	DrawHeaderCaption (see page 705)	This is DrawHeaderCaption, a member of class TCustomCategoryPanel.
	DrawHeaderChevron (see page 705)	This is DrawHeaderChevron, a member of class TCustomCategoryPanel.
	GetCollapsedHeight (see page 706)	This is GetCollapsedHeight, a member of class TCustomCategoryPanel.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	Paint (see page 806)	Renders the image of the panel.
	ReadState (see page 707)	Prepares the control for having its properties assigned values from a stream.
	ResizeHeader (see page 708)	This is ResizeHeader, a member of class TCustomCategoryPanel.
	SetParent (see page 708)	Sets the parent of the control.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.
	UpdateControlOriginalParentSize (see page 709)	This is UpdateControlOriginalParentSize, a member of class TCustomCategoryPanel.
	UpdateHeader (see page 709)	This is UpdateHeader, a member of class TCustomCategoryPanel.
	WndProc (see page 709)	Provides specific message responses for the control.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	HeaderHeight (see page 717)	Retrieves the HeaderHeight property associated with the parent TCustomCategoryPanelGroup (see page 718) HeaderHeight.
	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Methods

	Name	Description
	Collapse (see page 703)	Collapses the category panel.
	Create (see page 700)	Creates and initializes an instance of TCustomPanel.
	Create (see page 803)	Creates and initializes an instance of TCustomPanel (see page 802).
	Destroy (see page 700)	Destroys an instance of TCustomControl.
	Expand (see page 705)	Expands the category panel.
	GetChildren (see page 706)	Calls a specified procedure for every control in the category panel.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	GetTabOrderList (see page 706)	Builds a list of controls in tab order.
	SetBounds (see page 708)	Changes the position and size of the category panel in a single operation.

Public Properties

	Name	Description
	HeaderState (see page 717)	Shows the state of the header of category panel.
	PanelGroup (see page 717)	Indentifies the parent category panel group of the category panel.

Published Events

	Name	Description
	OnClick (see page 344)	Occurs when the user clicks the control.
	OnCollapse (see page 346)	Occurs immediately after a category panel is collapsed.
	OnContextPopup (see page 346)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 347)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 347)	Occurs when another control is docked to the control.
	OnDockOver (see page 349)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 352)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 355)	Occurs when the user drags an object over a control.
	OnEndDock (see page 358)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 358)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 359)	Occurs when a control receives the input focus.
	OnExit (see page 362)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 364)	Occurs immediately after a category panel is expanded.
	OnGetSiteInfo (see page 365)	Returns the control's docking information.
	OnMouseActivate (see page 367)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 368)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 370)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 370)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 371)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 373)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 375)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 376)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 376)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	BiDiMode (see page 313)	Specifies the bi-directional mode for the control.
	Caption (see page 314)	Specifies a text string that identifies the control to the user.
	Collapsed (see page 316)	Specifies whether the category panel is collapsed.
	CollapsedHotImageIndex (see page 316)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.
	CollapsedImageIndex (see page 317)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..
	CollapsedPressedImageIndex (see page 317)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .
	Color (see page 318)	Specifies the background color of the control.
	Ctl3D (see page 318)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 319)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 320)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 322)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 322)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 322)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 324)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandedHotImageIndex (see page 324)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.
	ExpandedImageIndex (see page 325)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.
	ExpandedPressedImageIndex (see page 325)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.

	Font (see page 326)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 327)	Determines how the panel repaints itself when it is resized.
	Height (see page 327)	Specifies the vertical size of the control in pixels.
	Left (see page 328)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 329)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 329)	Specifies the padding of a control.
	ParentBackground (see page 330)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 330)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 330)	Determines where a control looks for its color information.
	ParentCtl3D (see page 331)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 331)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 332)	Determines where a control looks for its font information.
	ParentShowHint (see page 332)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 333)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 338)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 339)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 340)	Determines if the user can tab to a control.
	UseDockManager (see page 340)	Specifies whether the docking manager is used in drag-and-dock operations.
	Visible (see page 340)	Determines whether the component appears onscreen.
	Width (see page 342)	Specifies the horizontal size of the control or form in pixels.

3.1.5.2 TCategoryPanel Properties

The properties of the TCategoryPanel class are listed here.

Published Properties

	Name	Description
	BiDiMode (see page 313)	Specifies the bi-directional mode for the control.
	Caption (see page 314)	Specifies a text string that identifies the control to the user.
	Collapsed (see page 316)	Specifies whether the category panel is collapsed.
	CollapsedHotImageIndex (see page 316)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.
	CollapsedImageIndex (see page 317)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..
	CollapsedPressedImageIndex (see page 317)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .
	Color (see page 318)	Specifies the background color of the control.
	Ctl3D (see page 318)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 319)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 320)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 322)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 322)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 322)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 324)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandedHotImageIndex (see page 324)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.
	ExpandedImageIndex (see page 325)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.

	ExpandedPressedImageIndex (see page 325)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.
	Font (see page 326)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 327)	Determines how the panel repaints itself when it is resized.
	Height (see page 327)	Specifies the vertical size of the control in pixels.
	Left (see page 328)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 329)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 329)	Specifies the padding of a control.
	ParentBackground (see page 330)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 330)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 330)	Determines where a control looks for its color information.
	ParentCtl3D (see page 331)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 331)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 332)	Determines where a control looks for its font information.
	ParentShowHint (see page 332)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 333)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 338)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 339)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 340)	Determines if the user can tab to a control.
	UseDockManager (see page 340)	Specifies whether the docking manager is used in drag-and-dock operations.
	Visible (see page 340)	Determines whether the component appears onscreen.
	Width (see page 342)	Specifies the horizontal size of the control or form in pixels.

3.1.5.2.1 TCategoryPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.5.2.2 TCategoryPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
```

```
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
```

```
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

3.1.5.2.3 TCategoryPanel.Collapsed Property

Specifies whether the category panel is collapsed.

Pascal

```
property Collapsed: Boolean;
```

C++

```
__property Boolean Collapsed;
```

Description

Use Collapsed to determine if the category panel is in a collapsed state. It can also be used to cause the category panel to collapse or expand.

See Also

Collapse (see page 703)

Expand (see page 705)

3.1.5.2.4 TCategoryPanel.CollapsedHotImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.

Pascal

```
property CollapsedHotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedHotImageIndex;
```

Description

Set CollapsedHotImageIndex to associate the cheveron with one of the images in the Images property of the parent CategoryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.5 TCategoryPanel.CollapsedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..

Pascal

```
property CollapsedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedImageIndex;
```

Description

Set CollapsedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the mouse is outside of the area of the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.6 TCategoryPanel.CollapsedPressedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .

Pascal

```
property CollapsedPressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedPressedImageIndex;
```

Description

Set CollapsedPressedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the left mouse button is pressed while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.7 TCategoryPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.5.2.8 TCategoryPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.5.2.9 TCategoryPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.5.2.10 TCategoryPanel.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.5.2.11 TCategoryPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.5.2.12 TCategoryPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.5.2.13 TCategoryPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.5.2.14 TCategoryPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.5.2.15 TCategoryPanel.ExpandedHotImageIndex Property

Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.

Pascal

```
property ExpandedHotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedHotImageIndex;
```

Description

Set ExpandedHotImageIndex to associate the cheveron with one of the images in the Images property of the parent CategoryPanelGroup. The selected image will be displayed when the category panel is in the expanded state while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.16 TCategoryPanel.ExpandedImageIndex Property

Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.

Pascal

```
property ExpandedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedImageIndex;
```

Description

Set ExpandedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the expanded state while the mouse is outside of the area of the header.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.17 TCategoryPanel.ExpandedPressedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.

Pascal

```
property ExpandedPressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedPressedImageIndex;
```

Description

Set CollapsedPressedImageIndex to associate the chevron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state and the left mouse button is pressed while the mouse hovers over the header.

This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.5.2.18 TCategoryPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
```

```
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.5.2.19 TCategoryPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate

3.1.5.2.20 TCategoryPanel.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
```

```
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

3.1.5.2.21 TCategoryPanel.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left;
```

C++

```
__property Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
```

```
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top

3.1.5.2.22 TCategoryPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align

TOleContainer (see page 5969)

3.1.5.2.23 TCategoryPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

3.1.5.2.24 TCategoryPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.5.2.25 TCategoryPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.5.2.26 TCategoryPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.5.2.27 TCategoryPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.5.2.28 TCategoryPanel.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.5.2.29 TCategoryPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.5.2.30 TCategoryPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.5.2.31 TCategoryPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.5.2.32 TCategoryPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.5.2.33 TCategoryPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.5.2.34 TCategoryPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.5.2.35 TCategoryPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager;
```

C++

```
__property UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager

3.1.5.2.36 TCategoryPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.ActivePage.ClientWidth - Width);
        Left := Random(PageControl1.ActivePage.ClientHeight - Height);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.5.2.37 TCategoryPanel.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

3.1.5.3 TCategoryPanel Events

The events of the TCategoryPanel class are listed here.

Published Events

	Name	Description
	OnClick (see page 344)	Occurs when the user clicks the control.
	OnCollapse (see page 346)	Occurs immediately after a category panel is collapsed.
	OnContextPopup (see page 346)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 347)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 347)	Occurs when another control is docked to the control.
	OnDockOver (see page 349)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 352)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 355)	Occurs when the user drags an object over a control.
	OnEndDock (see page 358)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 358)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 359)	Occurs when a control receives the input focus.
	OnExit (see page 362)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 364)	Occurs immediately after a category panel is expanded.
	OnGetSiteInfo (see page 365)	Returns the control's docking information.
	OnMouseActivate (see page 367)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 368)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 370)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 370)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 371)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 373)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 375)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 376)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 376)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.5.3.1 TCategoryPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This

event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.5.3.2 TCategoryPanel.OnCollapse Event

Occurs immediately after a category panel is collapsed.

Pascal

```
property OnCollapse: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a category panel is collapsed.

See Also

OnExpand (see page 718)

3.1.5.3.3 TCategoryPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.5.3.4 TCategoryPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.5.3.5 TCategoryPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

3.1.5.3.6 TCategoryPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.5.3.7 TCategoryPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.5.3.8 TCategoryPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.5.3.9 TCategoryPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.5.3.10 TCategoryPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.5.3.11 TCategoryPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.5.3.12 TCategoryPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.5.3.13 TCategoryPanel.OnExpand Event

Occurs immediately after a category panel is expanded.

Pascal

```
property OnExpand: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a category panel is expanded.

See Also

OnCollapse (see page 718)

3.1.5.3.14 TCategoryPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
```

```
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
```

```
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.5.3.15 TCategoryPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events.

This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.5.3.16 TCategoryPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
```

```
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.5.3.17 TCategoryPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.5.3.18 TCategoryPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.5.3.19 TCategoryPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.5.3.20 TCategoryPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.5.3.21 TCategoryPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.5.3.22 TCategoryPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.5.3.23 TCategoryPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}
```

```
void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
```

```
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

3.1.6 TCategoryPanelGroup Class

TCategoryPanelGroup is a container for collapsible panels that are TCategoryPanel (see page 309) objects.

Class Hierarchy

Pascal

```
TCategoryPanelGroup = class(TCustomCategoryPanelGroup);
```

C++

```
class TCategoryPanelGroup : public TCustomCategoryPanelGroup;
```

File

ExtCtrls

Description

TCategoryPanelGroup is a container for collapsible panels that are TCategoryPanel (see page 309) objects. To add a new panel to TCategoryPanelGroup at design time, right-click the TCategoryPanelGroup object and choose New Panel. You can also collapse or expand all the panel in the same context menu by selecting Collapse All or Expand All. To collapse or expand and individual panel, you can toggle the Collapsed property in the object inspector for the desired TCateoryPanel object.

See Also

TCategoryPanel (see page 309)

3.1.6.1 TCategoryPanelGroup Members

The following tables list the members exposed by TCategoryPanelGroup.

Enumerations

	Name	Description
	THeaderStyle (see page 721)	Specifies the appearance of the headers of the the category panels.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 723)	Creates a Windows control corresponding to the TScrollingWinControl object.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	GetCategoryPanelClass (see page 723)	This is GetCategoryPanelClass, a member of class TCustomCategoryPanelGroup.
	Loaded (see page 724)	Initializes the control after it is loaded from a stream.
	Notification (see page 724)	Responds to notifications that components are being created or destroyed.
	ReorderPanels (see page 725)	This is ReorderPanels, a member of class TCustomCategoryPanelGroup.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	WndProc (see page 725)	Provides specific message responses for the control.

Protected Properties

	Name	Description
	AutoScroll (see page 2561)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Public Methods

	Name	Description
	CollapseAll (see page 722)	Collapses all the category panels within the category panel group.
	Create (see page 720)	Creates and initializes a new TScrollingWinControl object.
	Create (see page 2555)	Creates and initializes a new TScrollingWinControl (see page 2554) object.
	CreatePanel (see page 722)	Creates a new category panel for the category panel group.
	Destroy (see page 721)	Removes the TScrollingWinControl object from memory.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	ExpandAll (see page 723)	Expands all the category panels within the category panel group.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Public Properties

	Name	Description
	Panels (see page 735)	Lists the panels (TCategoryPanel (see page 309) objects) in the category panel group.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 418)	Occurs when an attempt is made to resize the control.
	OnClick (see page 419)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 421)	Adjust resize constraints.

	OnContextPopup (see page 421)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 422)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 422)	Occurs when another control is docked to the control.
	OnDockOver (see page 424)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 427)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 430)	Occurs when the user drags an object over a control.
	OnEndDock (see page 433)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 433)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 434)	Occurs when a control receives the input focus.
	OnExit (see page 437)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 439)	Returns the control's docking information.
	OnMouseActivate (see page 442)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 443)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 445)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 445)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 445)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 448)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 450)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 450)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 451)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 451)	Occurs immediately after the control is resized.
	OnStartDock (see page 452)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 452)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 453)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 383)	Determines how the control aligns within its container (parent control).
	Anchors (see page 387)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 387)	Specifies which edges of the control are beveled.
	BevelInner (see page 388)	Specifies the cut of the inner bevel.
	BevelKind (see page 388)	Specifies the control's bevel style.
	BevelOuter (see page 389)	Specifies the cut of the outer bevel.
	BevelWidth (see page 389)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 389)	Specifies the bi-directional mode for the control.
	ChevronAlignment (see page 390)	Determines how the chevron is aligned within the header of the category panels in this category panel group.
	ChevronColor (see page 390)	Specifies the color of the default chevron in the normal state.
	ChevronHotColor (see page 391)	Specifies the color of the default chevron in the hot state.
	Color (see page 391)	Specifies the background color of the control.
	Constraints (see page 392)	Specifies the size constraints for the control.
	Ctl3D (see page 393)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 393)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 394)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 396)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 396)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 397)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 398)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 399)	Controls the attributes of text written on or in the control.
	GradientBaseColor (see page 400)	Specifies the left or top color of the panel header in gradient style.
	GradientColor (see page 400)	Specifies the right or bottom color of the panel header in gradient style.
	GradientDirection (see page 400)	Specifies the direction of the gradient in the panel header.
	HeaderAlignment (see page 401)	Determines how the caption is aligned within the headers of the category panels in this category panel group.
	HeaderFont (see page 401)	Specifies the font of the caption of the headers of the panels.
	HeaderHeight (see page 401)	Specifies the height of the headers of the panels.
	HeaderImage (see page 402)	Specifies the image used for the background of the headers of the panels.
	HeaderStyle (see page 402)	Specifies the appearance of the headers of the the category panels.
	Height (see page 402)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	Images (see page 403)	Specifies an imagelist that can be used by the panels to replace the chevron. images.
	Padding (see page 404)	Specifies the padding of a control.
	ParentBackground (see page 404)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 404)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 405)	Determines where a control looks for its color information.
	ParentCtl3D (see page 405)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 406)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 406)	Determines where a control looks for its font information.
	ParentShowHint (see page 406)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 407)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 412)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 414)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 414)	Determines if the user can tab to a control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 415)	Determines whether the component appears onscreen.
	Width (see page 416)	Specifies the horizontal size of the control or form in pixels.

3.1.6.2 TCategoryPanelGroup Properties

The properties of the TCategoryPanelGroup class are listed here.

Published Properties

	Name	Description
	Align (see page 383)	Determines how the control aligns within its container (parent control).
	Anchors (see page 387)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 387)	Specifies which edges of the control are beveled.
	BevelInner (see page 388)	Specifies the cut of the inner bevel.
	BevelKind (see page 388)	Specifies the control's bevel style.
	BevelOuter (see page 389)	Specifies the cut of the outer bevel.
	BevelWidth (see page 389)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 389)	Specifies the bi-directional mode for the control.
	ChevronAlignment (see page 390)	Determines how the chevron is aligned within the header of the category panels in this category panel group.
	ChevronColor (see page 390)	Specifies the color of the default chevron in the normal state.
	ChevronHotColor (see page 391)	Specifies the color of the default chevron in the hot state.
	Color (see page 391)	Specifies the background color of the control.
	Constraints (see page 392)	Specifies the size constraints for the control.

	Ctl3D (see page 393)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 393)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 394)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 396)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 396)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 397)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 398)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 399)	Controls the attributes of text written on or in the control.
	GradientBaseColor (see page 400)	Specifies the left or top color of the panel header in gradient style.
	GradientColor (see page 400)	Specifies the right or bottom color of the panel header in gradient style.
	GradientDirection (see page 400)	Specifies the direction of the gradient in the panel header.
	HeaderAlignment (see page 401)	Determines how the caption is aligned within the headers of the category panels in this category panel group.
	HeaderFont (see page 401)	Specifies the font of the caption of the headers of the panels.
	HeaderHeight (see page 401)	Specifies the height of the headers of the panels.
	HeaderImage (see page 402)	Specifies the image used for the background of the headers of the panels.
	HeaderStyle (see page 402)	Specifies the appearance of the headers of the the category panels.
	Height (see page 402)	Specifies the vertical size of the control in pixels.
	Images (see page 403)	Specifies an imagelist that can be used by the panels to replace the chevron. images.
	Padding (see page 404)	Specifies the padding of a control.
	ParentBackground (see page 404)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 404)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 405)	Determines where a control looks for its color information.
	ParentCtl3D (see page 405)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 406)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 406)	Determines where a control looks for its font information.
	ParentShowHint (see page 406)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 407)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 412)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 414)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 414)	Determines if the user can tab to a control.
	Visible (see page 415)	Determines whether the component appears onscreen.
	Width (see page 416)	Specifies the horizontal size of the control or form in pixels.

3.1.6.2.1 TCategoryPanelGroup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The

value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
```

```
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
```

```
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.6.2.2 TCategoryPanelGroup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.6.2.3 TCategoryPanelGroup.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.6.2.4 TCategoryPanelGroup.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.6.2.5 TCategoryPanelGroup.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.6.2.6 TCategoryPanelGroup.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.6.2.7 TCategoryPanelGroup.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

3.1.6.2.8 TCategoryPanelGroup.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.6.2.9 TCategoryPanelGroup.ChevronAlignment Property

Determines how the chevron is aligned within the header of the category panels in this category panel group.

Pascal

```
property ChevronAlignment: TAlignment;
```

C++

```
__property TAlignment ChevronAlignment;
```

Description

Use the ChevronAlignment property to control the position of the chevron of the category panels within the category panel group. These are the possible values:

Value	Meaning
taLeftJustify	Align Chevron on the left side of the panel.
taCenter	Center the chevron on the panel.
taRightJustify	Align chevron on the right side of the panel.

3.1.6.2.10 TCategoryPanelGroup.ChevronColor Property

Specifies the color of the default chevron in the normal state.

Pascal

```
property ChevronColor: TColor;
```

C++

```
__property TColor ChevronColor;
```

Description

Use the ChevronColor property to select the color of the default normal chevron of the category panels in the category panel group. This property has no effect if the category panel overrides the default normal chevron with the CollapsedImageIndex property or the ExpandedImageIndex property.

See Also

THeaderState (see page 701)

CollapsedImageIndex (see page 715)

ExpandedImageIndex (see page 716)

3.1.6.2.11 TCategoryPanelGroup.ChevronHotColor Property

Specifies the color of the default chevron in the hot state.

Pascal

```
property ChevronHotColor: TColor;
```

C++

```
__property TColor ChevronHotColor;
```

Description

Use the ChevronHotColor property to select the color of the default hot chevron of the category panels in the category panel group. This property has no effect if the category panel overrides the default hot chevron color with the CollpasedHotImageIndex property or the ExpandedHotImageIndex property.

See Also

THeaderState (see page 701)

CollapsedImageIndex (see page 715)

ExpandedImageIndex (see page 716)

3.1.6.2.12 TCategoryPanelGroup.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.6.2.13 TCategoryPanelGroup.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.6.2.14 TCategoryPanelGroup.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.6.2.15 TCategoryPanelGroup.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
```

```
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.6.2.16 TCategoryPanelGroup.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.6.2.17 TCategoryPanelGroup.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.6.2.18 TCategoryPanelGroup.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.6.2.19 TCategoryPanelGroup.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.6.2.20 TCategoryPanelGroup.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.6.2.21 TCategoryPanelGroup.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.6.2.22 TCategoryPanelGroup.GradientBaseColor Property

Specifies the left or top color of the panel header in gradient style.

Pascal

```
property GradientBaseColor: TColor;
```

C++

```
__property TColor GradientBaseColor;
```

Description

Use the GradientBaseColor property to select the base gradient color for the panel header when HeaderStyle is set to hsGradient. If GradientDirection is set to gdHorizontal, GradientBaseColor represents the left color of the header. If GradientDirection is set to gdVertical, GradientBaseColor represents the top color of the header. This property has no effect if HeaderStyle is set to something other than hsGradient

See Also

HeaderStyle (see page 734)

3.1.6.2.23 TCategoryPanelGroup.GradientColor Property

Specifies the right or bottom color of the panel header in gradient style.

Pascal

```
property GradientColor: TColor;
```

C++

```
__property TColor GradientColor;
```

Description

Use the GradientColor property to select the gradient color for the panel header when HeaderStyle is set to hsGradient. If GradientDirection is set to gdHorizontal, GradientColor represents the right color of the header. If GradientDirection is set to gdVertical, GradientColor represents the bottom color of the header. This property has no effect if HeaderStyle is set to something other than hsGradient.

See Also

HeaderStyle (see page 734)

3.1.6.2.24 TCategoryPanelGroup.GradientDirection Property

Specifies the direction of the gradient in the panel header.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

Use the GradientDirection property to select the direction of the gradient header of the category panels in the category panel

group. The direction can either be horizontal or vertical. This property has no effect if HeaderStyle is set to something other than hsGradient.

See Also

HeaderStyle (see page 734)

3.1.6.2.25 TCategoryPanelGroup.HeaderAlignment Property

Determines how the caption is aligned within the headers of the category panels in this category panel group.

Pascal

```
property HeaderAlignment: TAlignment;
```

C++

```
__property TAlignment HeaderAlignment;
```

Description

Use the HeaderAlignment property to control the position of the caption of the category panels within the category panel group. These are the possible values:

Value	Meaning
taLeftJustify	Align caption on the left side of the header.
taCenter	Center the caption in the header.
taRightJustify	Align caption on the right side of the header.

3.1.6.2.26 TCategoryPanelGroup.HeaderFont Property

Specifies the font of the caption of the headers of the panels.

Pascal

```
property HeaderFont: TFont;
```

C++

```
__property TFont HeaderFont;
```

Description

Use the HeaderFont property to specify the font and font properties used to draw the caption of the headers of the category panels in the category panel group. The HeaderFont property has no effect when HeaderStyle is hsThemed.

3.1.6.2.27 TCategoryPanelGroup.HeaderHeight Property

Specifies the height of the headers of the panels.

Pascal

```
property HeaderHeight: Integer;
```

C++

```
__property int HeaderHeight;
```

Description

Use the HeaderHeight property to specify the height of the headers of the category panels in the category panel group.

3.1.6.2.28 TCategoryPanelGroup.HeaderImage Property

Specifies the image used for the background of the headers of the panels.

Pascal

```
property HeaderImage: TPicture;
```

C++

```
__property TPicture HeaderImage;
```

Description

Use the HeaderImage property to specify an image as the background for the headers of the category panels within the category panel group. This property has no effect if HeaderStyle is set to something other than hsImage.

See Also

HeaderStyle (see page 734)

3.1.6.2.29 TCategoryPanelGroup.HeaderStyle Property

Specifies the appearance of the headers of the the category panels.

Pascal

```
property HeaderStyle: THeaderStyle;
```

C++

```
__property THeaderStyle HeaderStyle;
```

Description

Use HeaderStyle to determine the appearance of headers of the category panels in the category panel group.

Value	Meaning
hsGradient	Header is a gradient determined by the GradientBaseColor, GradientColor, and GradientDirection properties.
hsImage	Header is the stretched image defined by the HeaderImage property. If HeaderImage is undefined, the hsGradient style will be used instead.
hsThemed	Header is defined by the system's theme. If runtime themes aren't enabled or supported, the hsGradient style will be used instead.

3.1.6.2.30 TCategoryPanelGroup.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
```

```
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

3.1.6.2.31 TCategoryPanelGroup.Images Property

Specifies an imagelist that can be used by the panels to replace the chevron. images.

Pascal

```
property Images: TImageList;
```

C++

```
__property TImageList Images;
```

Description

Use the images property to define a set of pictures that can be use by the category panels within the category panel group to replace the chevron.

See Also

CollapsedHotImageIndex (see page 714)

CollapsedImageIndex (see page 715)

CollapsedPressedImageIndex (see page 715)

ExpandedHotImageIndex (see page 716)

ExpandedImageIndex (see page 716)

ExpandedPressedImageIndex (see page 716)

3.1.6.2.32 TCategoryPanelGroup.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

3.1.6.2.33 TCategoryPanelGroup.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.6.2.34 TCategoryPanelGroup.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.6.2.35 TCategoryPanelGroup.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.6.2.36 TCategoryPanelGroup.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.6.2.37 TCategoryPanelGroup.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.6.2.38 TCategoryPanelGroup.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.6.2.39 TCategoryPanelGroup.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.6.2.40 TCategoryPanelGroup.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
```

```
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.6.2.41 TCategoryPanelGroup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
```

```
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.6.2.42 TCategoryPanelGroup.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.6.2.43 TCategoryPanelGroup.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.6.2.44 TCategoryPanelGroup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.6.2.45 TCategoryPanelGroup.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

3.1.6.3 TCategoryPanelGroup Events

The events of the TCategoryPanelGroup class are listed here.

Published Events

	Name	Description
	OnCanResize (see page 418)	Occurs when an attempt is made to resize the control.
	OnClick (see page 419)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 421)	Adjust resize constraints.
	OnContextPopup (see page 421)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 422)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 422)	Occurs when another control is docked to the control.
	OnDockOver (see page 424)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 427)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 430)	Occurs when the user drags an object over a control.
	OnEndDock (see page 433)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 433)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 434)	Occurs when a control receives the input focus.
	OnExit (see page 437)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 439)	Returns the control's docking information.
	OnMouseActivate (see page 442)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 443)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 445)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 445)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 445)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 448)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 450)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 450)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 451)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 451)	Occurs immediately after the control is resized.
	OnStartDock (see page 452)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 452)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 453)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.6.3.1 TCategoryPanelGroup.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

3.1.6.3.2 TCategoryPanelGroup.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
```

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.6.3.3 TCategoryPanelGroup.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

3.1.6.3.4 TCategoryPanelGroup.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.6.3.5 TCategoryPanelGroup.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.6.3.6 TCategoryPanelGroup.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

3.1.6.3.7 TCategoryPanelGroup.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.6.3.8 TCategoryPanelGroup.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.6.3.9 TCategoryPanelGroup.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.6.3.10 TCategoryPanelGroup.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.6.3.11 TCategoryPanelGroup.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.6.3.12 TCategoryPanelGroup.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.6.3.13 TCategoryPanelGroup.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.6.3.14 TCategoryPanelGroup.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.6.3.15 TCategoryPanelGroup.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.6.3.16 TCategoryPanelGroup.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.6.3.17 TCategoryPanelGroup.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.6.3.18 TCategoryPanelGroup.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.6.3.19 TCategoryPanelGroup.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.6.3.20 TCategoryPanelGroup.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.6.3.21 TCategoryPanelGroup.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel;
```

C++

```
__property OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown

OnMouseWheelUp

3.1.6.3.22 TCategoryPanelGroup.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

3.1.6.3.23 TCategoryPanelGroup.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

3.1.6.3.24 TCategoryPanelGroup.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

3.1.6.3.25 TCategoryPanelGroup.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.6.3.26 TCategoryPanelGroup.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.6.3.27 TCategoryPanelGroup.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
```

```
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

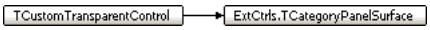
DockSite

OnDockDrop

OnDockOver

3.1.7 TCategoryPanelSurface Class

Class Hierarchy



Pascal

```
TCategoryPanelSurface = class(TCustomTransparentControl);
```

C++

```
class TCategoryPanelSurface : public TCustomTransparentControl;
```

File

ExtCtrls

Description

This is class ExtCtrls.TCategoryPanelSurface.

3.1.7.1 TCategoryPanelSurface Members

The following tables list the members exposed by TCategoryPanelSurface.

Protected Methods

	Name	Description
	CreateParams (see page 457)	Initializes a window-creation parameter data structure.

Public Methods

	Name	Description
	Create (see page 456)	Creates an instance of TCustomControl.

3.1.7.2 TCategoryPanelSurface.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

3.1.7.3 TCategoryPanelSurface Methods

The methods of the TCategoryPanelSurface class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 457)	Initializes a window-creation parameter data structure.

3.1.7.3.1 TCategoryPanelSurface.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

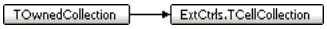
CreateWnd

AddBiDiModeExStyle

3.1.8 TCellCollection Class

A collection of TCellItem (see page 462) objects.

Class Hierarchy



Pascal

```
TCellCollection = class(TOwnedCollection);
```

C++

```
class TCellCollection : public TOwnedCollection;
```

File

ExtCtrls

Description

Use TCellCollection to represent all the cell items in a grid panel.

3.1.8.1 TCellCollection Members

The following tables list the members exposed by TCellCollection.

Protected Methods

	Name	Description
	GetAttr (see page 459)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 459)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 460)	Returns a specified item in the collection.
	GetItemAttr (see page 460)	Returns the value of a custom attribute assigned to one of the collection's items.
	SetItem (see page 461)	This is SetItem, a member of class TCellCollection.
	Update (see page 461)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Owner (see page 460)	Returns the Owner of the cell collection.

Public Properties

	Name	Description
	Items (see page 461)	Lists the items of the cell collection.

3.1.8.2 TCellCollection Methods

The methods of the TCellCollection class are listed here.

Protected Methods

	Name	Description
	GetAttr (see page 459)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 459)	Returns the number of custom attributes associated with items in the collection.

	GetItem (see page 460)	Returns a specified item in the collection.
V	GetItemAttr (see page 460)	Returns the value of a custom attribute assigned to one of the collection's items.
	SetItem (see page 461)	This is SetItem, a member of class TCellCollection.
V	Update (see page 461)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Owner (see page 460)	Returns the Owner of the cell collection.

3.1.8.2.1 TCellCollection.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount

GetItemAttr

3.1.8.2.2 TCellCollection.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr

GetItemAttr

3.1.8.2.3 TCellCollection.GetItem Method

Returns a specified item in the collection.

Pascal

```
function GetItem(Index: Integer): TCellItem;
```

C++

```
__fastcall TCellItem GetItem(int Index);
```

Description

GetItem is the protected read implementation of the Items property.

See Also

Items

3.1.8.2.4 TCellCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

3.1.8.2.5 TCellCollection.Owner Method

Returns the Owner of the cell collection.

Pascal

```
function Owner: TCustomGridPanel;
```

C++

```
__fastcall TCustomGridPanel Owner();
```

Description

Use the Owner method to obtain a reference to the list view that owns this cell collection.

3.1.8.2.6 TCellCollection.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TCellItem);
```

C++

```
__fastcall SetItem(int Index, TCellItem Value);
```

Description

This is SetItem, a member of class TCellCollection.

3.1.8.2.7 TCellCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed

Changed

3.1.8.3 TCellCollection Properties

The properties of the TCellCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 461)	Lists the items of the cell collection.

3.1.8.3.1 TCellCollection.Items Property

Lists the items of the cell collection.

Pascal

```
property Items [Index: Integer]: TCellItem;
```

C++

```
__property TCellItem Items[int Index];
```

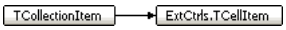
Description

Use the Items property to list the items of TCellCollection (see page 458).

3.1.9 TCellItem Class

Represents an item in a TCellCollection (see page 458).

Class Hierarchy



Pascal

```
TCellItem = class(TCollectionItem);
```

C++

```
class TCellItem : public TCollectionItem;
```

File

ExtCtrls

Description

The TCellItem class represents a cell on a grid panel.

3.1.9.1 TCellItem Members

The following tables list the members exposed by TCellItem.

Protected Methods

	Name	Description
	AssignTo (see page 463)	Copies the properties of an object to a destination object.
	SetSizeStyle (see page 464)	This is SetSizeStyle, a member of class TCellItem.
	SetValue (see page 464)	This is SetValue, a member of class TCellItem.

Protected Properties

	Name	Description
	AutoAdded (see page 464)	This is AutoAdded, a member of class TCellItem.
	Size (see page 465)	This is Size, a member of class TCellItem.

Public Methods

	Name	Description
	Create (see page 462)	Creates and initializes a TCollectionItem instance.

Published Properties

	Name	Description
	SizeStyle (see page 465)	Specifies the manner in which column size or row size is specified.
	Value (see page 465)	Sets the size of the cell item.

3.1.9.2 TCellItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCellItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

3.1.9.3 TCellItem Methods

The methods of the TCellItem class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 463)	Copies the properties of an object to a destination object.
	SetSizeStyle (see page 464)	This is SetSizeStyle, a member of class TCellItem.
	SetValue (see page 464)	This is SetValue, a member of class TCellItem.

3.1.9.3.1 TCellItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

3.1.9.3.2 TCellItem.SetSizeStyle Method

Pascal

```
procedure SetSizeStyle(Value: TSizeStyle);
```

C++

```
__fastcall SetSizeStyle(TSizeStyle Value);
```

Description

This is SetSizeStyle, a member of class TCellItem.

3.1.9.3.3 TCellItem.SetValue Method

Pascal

```
procedure SetValue(Value: Double);
```

C++

```
__fastcall SetValue(Double Value);
```

Description

This is SetValue, a member of class TCellItem.

3.1.9.4 TCellItem Properties

The properties of the TCellItem class are listed here.

Protected Properties

	Name	Description
	AutoAdded (see page 464)	This is AutoAdded, a member of class TCellItem.
	Size (see page 465)	This is Size, a member of class TCellItem.

Published Properties

	Name	Description
	SizeStyle (see page 465)	Specifies the manner in which column size or row size is specified.
	Value (see page 465)	Sets the size of the cell item.

3.1.9.4.1 TCellItem.AutoAdded Property

Pascal

```
property AutoAdded: Boolean;
```

C++

```
__property Boolean AutoAdded;
```

Description

This is AutoAdded, a member of class TCellItem.

3.1.9.4.2 TCellItem.Size Property

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

This is Size, a member of class TCellItem.

3.1.9.4.3 TCellItem.SizeStyle Property

Specifies the manner in which column size or row size is specified.

Pascal

```
property SizeStyle: TSizeStyle;
```

C++

```
__property TSizeStyle SizeStyle;
```

Description

Use the SizeStyle property to specify whether the size of the row or column in a grid is an absolute value, a percentage of the grid size, or generated automatically.

Style	Description
ssAbsolute	The number specified for the Value (see page 465) property is interpreted as the number of pixels for the row or column.
ssAuto	The number specified for the Value (see page 465) property is ignored, and the row or column is sized automatically.
ssPercent	The number specified for the Value (see page 465) property is interpreted as a percentage of the entire grid panel.

3.1.9.4.4 TCellItem.Value Property

Sets the size of the cell item.

Pascal

```
property Value: Double;
```

C++

```
__property Double Value;
```

Description

Use the Value property to specify the size of the column or row in the grid. This value can be set automatically or expressed as a percentage of the grid size or as an absolute value, depending on the SizeStyle (see page 465). For example, if SizeStyle (see page 465) is set to ssPercent, and you specify a value of 60, the column occupies approximately 60% of the grid panes width or a row occupies approximately 60% of the grid panel height.

3.1.10 TColorBox Class

TColorBox represents a combo box that lets users select a color.

Class Hierarchy

Pascal

```
TColorBox = class(TCustomColorBox);
```

C++

```
class TColorBox : public TCustomColorBox;
```

File

ExtCtrls

Description

Use TColorBox to provide the user with a drop-down combo box from which to select a color. Use the Style (see page 493) property to specify which colors the color box should list. Each color is displayed next to a small square that is drawn in the corresponding color.

Use the Selected (see page 491) property to access the color that the user selects. The Colors (see page 742) property provides access to the entire list of colors, while the ColorNames (see page 741) property lists their names.

See Also

TComboBox

TColorDialog

3.1.10.1 TColorBox Members

The following tables list the members exposed by TColorBox.

Protected Methods

	Name	Description
	CloseUp (see page 737)	Generates an OnCloseUp event.
	CreateWnd (see page 738)	Creates the window used to implement the color box.
	DrawItem (see page 738)	Renders an individual item in the color box.
	KeyDown (see page 738)	Generates an OnKeyDown event.
	KeyPress (see page 739)	Generates an OnKeyPress event.
	Loaded (see page 739)	Initializes the combo box after its form is loaded into memory.
	PickCustomColor (see page 740)	Displays the color selection dialog so that the user can select a custom color.
	PopulateList (see page 740)	Fills the color box with items that reflect the colors specified by the Style (see page 744) property.
	Select (see page 740)	Generates an OnSelect event.
	SetStyle (see page 741)	Sets the Style (see page 744) property.

Public Methods

	Name	Description
	Create (see page 737)	Creates a new TCustomColorBox (see page 735) object.

Public Properties

	Name	Description
R	ColorNames (see page 741)	Lists the names of the colors in the color box.
R	Colors (see page 742)	Lists the colors in the color box.

Published Events

	Name	Description
	OnChange (see page 497)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 499)	Occurs when the user clicks the control.
	OnCloseUp (see page 501)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 501)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 502)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 502)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 505)	Occurs when the user drags an object over a control.
	OnDropDown (see page 508)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 510)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 510)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 511)	Occurs when a control receives the input focus.
	OnExit (see page 514)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 516)	Occurs when the control is populated with colors.
	OnKeyDown (see page 517)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 520)	Occurs when a key is pressed.
	OnKeyUp (see page 521)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 523)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 523)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 524)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 524)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 525)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 469)	Determines how the control aligns within its container (parent control).
	Anchors (see page 472)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 473)	Positions to matching list items as you type.
	AutoDropDown (see page 474)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 474)	Specifies which edges of the control are beveled.
	BevelInner (see page 475)	Specifies the cut of the inner bevel.
	BevelKind (see page 475)	Specifies the control's bevel style.
	BevelOuter (see page 476)	Specifies the cut of the outer bevel.
	BiDiMode (see page 476)	Specifies the bi-directional mode for the control.
	Color (see page 477)	Specifies the background color of the control.
	Constraints (see page 477)	Specifies the size constraints for the control.
	Ctl3D (see page 478)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 479)	Specifies the color displayed in the color box for clDefault.
	DoubleBuffered (see page 479)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropDownCount (see page 481)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 481)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 482)	Controls the attributes of text written on or in the control.

	ItemHeight (see page 483)	Specifies the height, in pixels, of the items in the drop-down list.
	NoneColorColor (see page 483)	Specifies the color displayed in the color box for clNone.
	ParentBiDiMode (see page 484)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 484)	Determines where a control looks for its color information.
	ParentCtl3D (see page 485)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 485)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 485)	Determines where a control looks for its font information.
	ParentShowHint (see page 486)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 486)	Identifies the pop-up menu associated with the control.
	Selected (see page 491)	Specifies the currently selected color.
	ShowHint (see page 492)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 493)	Controls what colors are listed and how their names are formatted.
	TabOrder (see page 493)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 494)	Determines if the user can tab to a control.
	Visible (see page 494)	Determines whether the component appears onscreen.

3.1.10.2 TColorBox Properties

The properties of the TColorBox class are listed here.

Published Properties

	Name	Description
	Align (see page 469)	Determines how the control aligns within its container (parent control).
	Anchors (see page 472)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 473)	Positions to matching list items as you type.
	AutoDropDown (see page 474)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 474)	Specifies which edges of the control are beveled.
	BevelInner (see page 475)	Specifies the cut of the inner bevel.
	BevelKind (see page 475)	Specifies the control's bevel style.
	BevelOuter (see page 476)	Specifies the cut of the outer bevel.
	BiDiMode (see page 476)	Specifies the bi-directional mode for the control.
	Color (see page 477)	Specifies the background color of the control.
	Constraints (see page 477)	Specifies the size constraints for the control.
	Ctl3D (see page 478)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 479)	Specifies the color displayed in the color box for clDefault.
	DoubleBuffered (see page 479)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropDownCount (see page 481)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 481)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 482)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 483)	Specifies the height, in pixels, of the items in the drop-down list.
	NoneColorColor (see page 483)	Specifies the color displayed in the color box for clNone.
	ParentBiDiMode (see page 484)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 484)	Determines where a control looks for its color information.
	ParentCtl3D (see page 485)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 485)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 485)	Determines where a control looks for its font information.
	ParentShowHint (see page 486)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 486)	Identifies the pop-up menu associated with the control.

	Selected (see page 491)	Specifies the currently selected color.
	ShowHint (see page 492)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 493)	Controls what colors are listed and how their names are formatted.
	TabOrder (see page 493)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 494)	Determines if the user can tab to a control.
	Visible (see page 494)	Determines whether the component appears onscreen.

3.1.10.2.1 TColorBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
```

```
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.10.2.2 TColorBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.10.2.3 TColorBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
```

```
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
             mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

AutoCompleteDelay

3.1.10.2.4 TColorBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown;
```

C++

```
__property AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete

OnDropDown

DroppedDown

3.1.10.2.5 TColorBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.10.2.6 TColorBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.10.2.7 TColorBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.10.2.8 TColorBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.10.2.9 TColorBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.10.2.10 TColorBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.10.2.11 TColorBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.10.2.12 TColorBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.10.2.13 TColorBox.DefaultColorColor Property

Specifies the color displayed in the color box for clDefault.

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

Set DefaultColorColor to indicate what color the color box should use when drawing the icon for clDefault. This property only affects the color box if the Style property includes cbSystemColors and cbIncludeDefault.

Note: DefaultColorColor only affects the way clDefault appears in the color box. When you assign clDefault to a control, the control determines its own default color.

See Also

Colors (see page 742)

NoneColorColor (see page 743)

Style (see page 744)

3.1.10.2.14 TColorBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.10.2.15 TColorBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount;
```

C++

```
__property DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items

ItemCount

3.1.10.2.16 TColorBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.10.2.17 TColorBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.10.2.18 TColorBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight;
```

C++

```
__property ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items

OnMeasureItem

Style

3.1.10.2.19 TColorBox.NoneColorColor Property

Specifies the color displayed in the color box for clNone.

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

Set NoneColorColor to indicate what color the color box should use when drawing the icon for clNone. This property only affects the color box if the Style property includes cbSystemColors and cbIncludeNone.

Note: NoneColorColor only affects the way clNone appears in the color box. When you assign clNone to a control, the control determines how to interpret clNone.

See Also

Colors (see page 742)

DefaultColorColor (see page 742)

Style (see page 744)

3.1.10.2.20 TColorBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.10.2.21 TColorBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.10.2.22 TColorBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.10.2.23 TColorBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.10.2.24 TColorBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.10.2.25 TColorBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.10.2.26 TColorBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.10.2.27 TColorBox.Selected Property

Specifies the currently selected color.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to get or set the selected color as a TColor value.

See Also

Colors (see page 742)

ItemIndex

3.1.10.2.28 TColorBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.10.2.29 TColorBox.Style Property

Controls what colors are listed and how their names are formatted.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to configure what colors appear in the list and how the names appear. Style is a set that includes 0 or more of the TColorBoxStyles flags.

See Also

DefaultColorColor (see page 742)

NoneColorColor (see page 743)

ColorNames (see page 741)

Colors (see page 742)

TColor (see page 2878)

TColorDialog

3.1.10.2.30 TColorBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.10.2.31 TColorBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.10.2.32 TColorBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.10.3 TColorBox Events

The events of the TColorBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 497)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 499)	Occurs when the user clicks the control.
	OnCloseUp (see page 501)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 501)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 502)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 502)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 505)	Occurs when the user drags an object over a control.
	OnDropDown (see page 508)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 510)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 510)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 511)	Occurs when a control receives the input focus.
	OnExit (see page 514)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 516)	Occurs when the control is populated with colors.
	OnKeyDown (see page 517)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 520)	Occurs when a key is pressed.
	OnKeyUp (see page 521)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 523)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 523)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 524)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 524)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnStartDrag (see page 525)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.10.3.1 TColorBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
```

```
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
    end;
  ComboBox1.ItemIndex := 0;
  PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
```

```
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text

OnCloseUp

3.1.10.3.2 TColorBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.10.3.3 TColorBox.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp;
```

C++

```
__property OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown

DroppedDown

3.1.10.3.4 TColorBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.10.3.5 TColorBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.10.3.6 TColorBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
```

```
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
```

```
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.10.3.7 TColorBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.10.3.8 TColorBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown;
```

C++

```
__property OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown

OnCloseUp

3.1.10.3.9 TColorBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.10.3.10 TColorBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
```

```
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.10.3.11 TColorBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on

the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
```

```
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.10.3.12 TColorBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');
```

```
    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.10.3.13 TColorBox.OnGetColors Event

Occurs when the control is populated with colors.

Pascal

```
property OnGetColors: TGetColorsEvent;
```

C++

```
__property TGetColorsEvent OnGetColors;
```

3.1.10.3.14 TColorBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
```

```
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
```

```
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

3.1.10.3.15 TColorBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an

OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

3.1.10.3.16 TColorBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;
```

```
procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

3.1.10.3.17 TColorBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.10.3.18 TColorBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse

outside of a control when the parent top-level form is not active.

3.1.10.3.19 TColorBox.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect;
```

C++

```
__property OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete

ItemIndex

Text

3.1.10.3.20 TColorBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.10.3.21 TColorBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.11 TColorListBox Class

TColorListBox represents a list box that lets users select a color.

Class Hierarchy

Pascal

```
TColorListBox = class(TCustomColorListBox);
```

C++

```
class TColorListBox : public TCustomColorListBox;
```

File

ExtCtrls

Description

TCustomColorBox (see page 735) is the base class for TColorListBox and for combo boxes that display a list of colors. The list box displays a list of all the colors for which the Graphics (see page 2619) unit defines a constant. For a list of these colors, see TColor (see page 2878). Each color is displayed next to a small square that is drawn in the corresponding color.

Do not use TCustomColorBox (see page 735) in applications. It is intended to act as a base class for color selection controls. Rather, to add a list box control with selectable colors, use the TColorListBox component. To use a drop-down combo box that lets users select a color, use the TColorBox (see page 466) component, which appears on the Additional page of the component palette.

3.1.11.1 TColorListBox Members

The following tables list the members exposed by TColorListBox.

Protected Methods

	Name	Description
	CreateWnd (see page 746)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 747)	Generates an OnDrawItem event.
	KeyDown (see page 747)	Respond to key press events.
	KeyPress (see page 748)	Responds when the user presses a key.
	Loaded (see page 748)	Initializes the control after it is loaded from a stream.
	PickCustomColor (see page 749)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 749)	This is PopulateList, a member of class TCustomColorListBox.
	SetStyle (see page 749)	This is SetStyle, a member of class TCustomColorListBox.

Public Methods

	Name	Description
	Create (see page 746)	Instantiates a list box control.

Public Properties

	Name	Description
	ColorNames (see page 750)	This is ColorNames, a member of class TCustomColorListBox.
	Colors (see page 750)	This is Colors, a member of class TCustomColorListBox.

Published Events

	Name	Description
	OnClick (see page 555)	Occurs when the user clicks the control.
	OnContextPopup (see page 557)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 558)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 558)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 561)	Occurs when the user drags an object over a control.
	OnEndDock (see page 564)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 565)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 566)	Occurs when a control receives the input focus.
	OnExit (see page 568)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 571)	This is OnGetColors, a member of class TColorListBox.
	OnKeyDown (see page 571)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 575)	Occurs when a key is pressed.
	OnKeyUp (see page 576)	Occurs when the user releases a key that has been pressed.

	OnMouseActivate (see page 577)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 578)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 581)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 581)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 581)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 584)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 586)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 586)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 528)	Determines how the control aligns within its container (parent control).
	Anchors (see page 532)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 532)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 533)	Specifies which edges of the control are beveled.
	BevelInner (see page 533)	Specifies the cut of the inner bevel.
	BevelKind (see page 533)	Specifies the control's bevel style.
	BevelOuter (see page 534)	Specifies the cut of the outer bevel.
	BiDiMode (see page 534)	Specifies the bi-directional mode for the control.
	Color (see page 535)	Sets the background color of the TColorListBox (see page 525) control.
	Constraints (see page 535)	Specifies the size constraints for the control.
	Ctl3D (see page 536)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 536)	This is DefaultColorColor, a member of class TColorListBox.
	DoubleBuffered (see page 536)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 538)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 539)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 540)	Specifies the height in pixels of an item in an owner-draw list box.
	NoneColorColor (see page 540)	This is NoneColorColor, a member of class TColorListBox.
	ParentBiDiMode (see page 540)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 541)	Determines where a control looks for its color information.
	ParentCtl3D (see page 541)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 542)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 542)	Determines where a control looks for its font information.
	ParentShowHint (see page 542)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 543)	Identifies the pop-up menu associated with the control.
	Selected (see page 548)	Indicates whether a particular item is selected.
	ShowHint (see page 550)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 551)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 552)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 553)	Determines if the user can tab to a control.
	Visible (see page 553)	Determines whether the component appears onscreen.

3.1.11.2 TColorListBox Properties

The properties of the TColorListBox class are listed here.

Published Properties

	Name	Description
	Align (see page 528)	Determines how the control aligns within its container (parent control).
	Anchors (see page 532)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 532)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 533)	Specifies which edges of the control are beveled.
	BevelInner (see page 533)	Specifies the cut of the inner bevel.
	BevelKind (see page 533)	Specifies the control's bevel style.
	BevelOuter (see page 534)	Specifies the cut of the outer bevel.
	BiDiMode (see page 534)	Specifies the bi-directional mode for the control.
	Color (see page 535)	Sets the background color of the TColorListBox (see page 525) control.
	Constraints (see page 535)	Specifies the size constraints for the control.
	Ctl3D (see page 536)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 536)	This is DefaultColorColor, a member of class TColorListBox.
	DoubleBuffered (see page 536)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 538)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 539)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 540)	Specifies the height in pixels of an item in an owner-draw list box.
	NoneColorColor (see page 540)	This is NoneColorColor, a member of class TColorListBox.
	ParentBiDiMode (see page 540)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 541)	Determines where a control looks for its color information.
	ParentCtl3D (see page 541)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 542)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 542)	Determines where a control looks for its font information.
	ParentShowHint (see page 542)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 543)	Identifies the pop-up menu associated with the control.
	Selected (see page 548)	Indicates whether a particular item is selected.
	ShowHint (see page 550)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 551)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 552)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 553)	Determines if the user can tab to a control.
	Visible (see page 553)	Determines whether the component appears onscreen.

3.1.11.2.1 TColorListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always

equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
```

```
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
```

```
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
```

```
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.11.2.2 TColorListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.11.2.3 TColorListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

3.1.11.2.4 TColorListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.11.2.5 TColorListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.11.2.6 TColorListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.11.2.7 TColorListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.11.2.8 TColorListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.11.2.9 TColorListBox.Color Property

Sets the background color of the TColorListBox (see page 525) control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

The color property is set to white, by default.

3.1.11.2.10 TColorListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.11.2.11 TColorListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.11.2.12 TColorListBox.DefaultColorColor Property

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

This is DefaultColorColor, a member of class TColorListBox.

3.1.11.2.13 TColorListBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
```

```
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.11.2.14 TColorListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.11.2.15 TColorListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.11.2.16 TColorListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight;
```

C++

```
__property ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem

IntegralHeight

Items

MeasureItem

OnDrawItem

OnMeasureItem

Style

Using TDBListBox and TDBComboBox

3.1.11.2.17 TColorListBox.NoneColorColor Property

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

This is NoneColorColor, a member of class TColorListBox.

3.1.11.2.18 TColorListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.11.2.19 TColorListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.11.2.20 TColorListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.11.2.21 TColorListBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.11.2.22 TColorListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.11.2.23 TColorListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.11.2.24 TColorListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
```

```
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}
```

```
void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
```

```
  This example uses two edit boxes and one pop-up menu on a form.
  The pop-up menu contains Cut, Copy, and Paste commands. This code
  makes the pop-up menu available to both edit boxes:
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
```

```
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.11.2.25 TColorListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").t_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).t_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
```

```
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

extendedselect

Items

MultiSelect

SelCount

3.1.11.2.26 TColorListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
```

```
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.11.2.27 TColorListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display

graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem

ItemHeight

Items

MeasureItem

OnDrawItem

OnMeasureItem

OnData

OnDataFind

OnDataObject

3.1.11.2.28 TColorListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.11.2.29 TColorListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.11.2.30 TColorListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
```

```
    {
      TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this); // The owner will clean this up.
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.11.3 TColorListBox Events

The events of the TColorListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 555)	Occurs when the user clicks the control.
	OnContextPopup (see page 557)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 558)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 558)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 561)	Occurs when the user drags an object over a control.
	OnEndDock (see page 564)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 565)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 566)	Occurs when a control receives the input focus.
	OnExit (see page 568)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 571)	This is OnGetColors, a member of class TColorListBox.
	OnKeyDown (see page 571)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 575)	Occurs when a key is pressed.
	OnKeyUp (see page 576)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 577)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 578)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 581)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 581)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 581)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 584)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 586)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 586)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.11.3.1 TColorListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check

box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.11.3.2 TColorListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.11.3.3 TColorListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.11.3.4 TColorListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
```

```
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
```

```
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.11.3.5 TColorListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
```

```
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.11.3.6 TColorListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.11.3.7 TColorListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.11.3.8 TColorListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.11.3.9 TColorListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.11.3.10 TColorListBox.OnGetColors Event

Pascal

```
property OnGetColors: TLBGetColorsEvent;
```

C++

```
__property TLBGetColorsEvent OnGetColors;
```

Description

This is OnGetColors, a member of class TColorListBox.

3.1.11.3.11 TColorListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR

message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
```

```
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
```

```
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

3.1.11.3.12 TColorListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
```

```
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

3.1.11.3.13 TColorListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

3.1.11.3.14 TColorListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.11.3.15 TColorListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
```

```
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.11.3.16 TColorListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.11.3.17 TColorListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.11.3.18 TColorListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.11.3.19 TColorListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.11.3.20 TColorListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.11.3.21 TColorListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.12 TColumnCollection Class

Controls the **Collection Editor** for columns within a grid panel.

Class Hierarchy

Pascal

```
TColumnCollection = class(TCellCollection);
```

C++

```
class TColumnCollection : public TCellCollection;
```

File

ExtCtrls

Description

TColumnCollection controls the **Collection Editor** for columns within a grid panel.

See Also

Collection Editor

3.1.12.1 TColumnCollection Members

The following tables list the members exposed by TColumnCollection.

Protected Methods

	Name	Description
	GetAttr (see page 459)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 459)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 460)	Returns a specified item in the collection.
	GetItemAttr (see page 589)	Returns the value of a custom attribute assigned to one of the collection's items.
	Notify (see page 589)	Responds when items are added to or removed from the collection.
	SetItem (see page 461)	This is SetItem, a member of class TCellCollection.
	Update (see page 461)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 589)	Adds a column to the grid panel.
	Create (see page 588)	This is Create, a member of class TColumnCollection.
	Owner (see page 460)	Returns the Owner of the cell collection.

Public Properties

	Name	Description
	Items (see page 461)	Lists the items of the cell collection.

3.1.12.2 TColumnCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TColumnCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TColumnCollection.

3.1.12.3 TColumnCollection Methods

The methods of the TColumnCollection class are listed here.

Protected Methods

	Name	Description
	GetItemAttr (see page 589)	Returns the value of a custom attribute assigned to one of the collection's items.
	Notify (see page 589)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 589)	Adds a column to the grid panel.

3.1.12.3.1 TColumnCollection.Add Method

Adds a column to the grid panel.

Pascal

```
function Add: TColumnItem;
```

C++

```
__fastcall TColumnItem Add();
```

Description

Use the Add method to add a new column to the grid panel.

3.1.12.3.2 TColumnCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

3.1.12.3.3 TColumnCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added

Deleting

3.1.13 TColumnItem Class

Class Hierarchy

Pascal

```
TColumnItem = class(TCellItem);
```

C++

```
class TColumnItem : public TCellItem;
```

File

ExtCtrls

Description

This is class ExtCtrls.TColumnItem.

3.1.14 TControlBar Class

TControlBar manages the layout of toolbar components.

Class Hierarchy

Pascal

```
TControlBar = class(TCustomControlBar);
```

C++

```
class TControlBar : public TCustomControlBar;
```

File

ExtCtrls

Description

Use TControlBar as a docking site for toolbar components. Control bars contain child controls (usually TToolBar objects) that can be moved and resized independently. As with a cool bar, each child control resides on its own individual band that the user positions by dragging its sizing grip.

Tip: When docking TToolBar objects to a control bar, you may find it useful to use TToolDockObject.

Note: If you want to use TActionToolBar on a TControlBar component, you should set the AutoSize (see page 599) property to **False**. When the AutoSize (see page 599) property is set to True (default), the TActionToolBar control wraps its controls

and expands itself in size, accordingly. This does not allow the TActionToolBar component to display properly on a TControlBar component. When the AutoSize (see page 599) property is set to **False**, the TActionToolBar hides items on the end of the bar and displays a dropdown button that provides access to those items. The control displays in the proper size.

See Also

TCoolBar

TToolBar

3.1.14.1 TControlBar Members

The following tables list the members exposed by TControlBar.

Protected Methods

	Name	Description
	AlignControls (see page 757)	Aligns controls in the control bar.
	CanAutoSize (see page 757)	Indicates how the control can be resized.
	CreateParams (see page 758)	Sets the control bar's window flags.
	DoBandMove (see page 758)	Generates an OnBandMove (see page 772) event.
	DoBandPaint (see page 758)	Generates an OnBandPaint (see page 772) event.
	DoBeginBandMove (see page 759)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 759)	Generates an OnDockOver event and ensures proper docking of floating controls.
	DoEndBandMove (see page 759)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoPaletteChange (see page 760)	Prepares the control bar to realize a new palette.
	DragControl (see page 760)	Starts a drag operation on a control band.
	GetControlInfo (see page 761)	Generates an OnBandInfo (see page 771) event.
	GetPalette (see page 762)	Returns the handle to a palette for use by the control.
	GetSiteInfo (see page 762)	Generates an OnGetSiteInfo event
	HitTest (see page 763)	Returns the control at the specified coordinates.
	MouseDown (see page 763)	Handles mouse button clicks.
	MouseMove (see page 763)	Handles mouse movements.
	MouseUp (see page 764)	Handles releasing of a mouse button.
	Paint (see page 764)	Paints the control bar.
	PaintControlFrame (see page 765)	Paints a control band's frame.

Public Methods

	Name	Description
	Create (see page 755)	Creates and initializes an instance of TCustomControlBar (see page 754).
	Destroy (see page 756)	Destroys the TCustomControlBar (see page 754) instance.
	FlipChildren (see page 761)	Reverses the positions of child controls.
	StickControls (see page 765)	Places controls in their final positions after a control is moved in or dropped on the control bar.

Public Properties

	Name	Description
	Canvas (see page 602)	Specifies the TCanvas object that presents a drawing surface for the control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 623)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 624)	Occurs when an object with custom alignment is aligned.
	OnBandDrag (see page 624)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 625)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 625)	Occurs when a control band is moving.

	OnBandPaint (see page 626)	Occurs when a control band is painted.
	OnBeginBandMove (see page 626)	This is OnBeginBandMove, a member of class TControlBar.
	OnCanResize (see page 626)	Occurs when an attempt is made to resize the control.
	OnClick (see page 627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 628)	Adjust resize constraints.
	OnContextPopup (see page 629)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 630)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 630)	Occurs when another control is docked to the control.
	OnDockOver (see page 632)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 634)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 637)	Occurs when the user drags an object over a control.
	OnEndBandMove (see page 640)	This is OnEndBandMove, a member of class TControlBar.
	OnEndDock (see page 640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 641)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 642)	Occurs when a control receives the input focus.
	OnExit (see page 644)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 647)	Returns the control's docking information.
	OnMouseActivate (see page 649)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 650)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 652)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 653)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 653)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 655)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 657)	Occurs when the control bar is painted.
	OnResize (see page 658)	Occurs immediately after the control is resized.
	OnStartDock (see page 659)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 659)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 660)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 594)	Determines how the control aligns within its container (parent control).
	Anchors (see page 597)	Specifies how the control is anchored to its parent.
	AutoDock (see page 598)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 598)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 599)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 599)	Specifies which edges of the control are beveled.
	BevelInner (see page 600)	Specifies the cut of the inner bevel.
	BevelKind (see page 600)	Specifies the control's bevel style.
	BevelOuter (see page 601)	Specifies the cut of the outer bevel.
	BevelWidth (see page 601)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 601)	Specifies the width of the control's border.
	Color (see page 604)	Specifies the background color of the control.
	Constraints (see page 604)	Specifies the size constraints for the control.
	CornerEdge (see page 605)	The size of the cornering for the edge of the bands.
	DockSite (see page 605)	Specifies whether the control can be the target of drag-and-dock operations.

	DragCursor (see page 606)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 607)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 607)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 609)	This is DrawingStyle, a member of class TControlBar.
	Enabled (see page 609)	Controls whether the control responds to mouse, keyboard, and timer events.
	GradientDirection (see page 609)	Horizontal or Vertical.
	GradientEndColor (see page 609)	Stores the color that the gradient is painted with when it ends.
	GradientStartColor (see page 610)	Stores the color that the gradient is painted when it starts.
	ParentBackground (see page 610)	Determines whether control uses parent's theme background.
	ParentColor (see page 610)	Determines where a control looks for its color information.
	ParentCtl3D (see page 611)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 611)	Determines where a control looks for its font information.
	ParentShowHint (see page 612)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 612)	Specifies the control bar's picture.
	PopupMenu (see page 612)	Identifies the pop-up menu associated with the control.
	RowSize (see page 617)	Specifies the height of the control bar's rows.
	RowSnap (see page 618)	Specifies whether controls are snapped to the control bar's rows when docked.
	ShowHint (see page 618)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 620)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 620)	Determines if the user can tab to a control.
	Visible (see page 621)	Determines whether the component appears onscreen.

3.1.14.2 TControlBar Properties

The properties of the TControlBar class are listed here.

Public Properties

	Name	Description
	Canvas (see page 602)	Specifies the TCanvas object that presents a drawing surface for the control.

Published Properties

	Name	Description
	Align (see page 594)	Determines how the control aligns within its container (parent control).
	Anchors (see page 597)	Specifies how the control is anchored to its parent.
	AutoDock (see page 598)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 598)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 599)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 599)	Specifies which edges of the control are beveled.
	BevelInner (see page 600)	Specifies the cut of the inner bevel.
	BevelKind (see page 600)	Specifies the control's bevel style.
	BevelOuter (see page 601)	Specifies the cut of the outer bevel.
	BevelWidth (see page 601)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 601)	Specifies the width of the control's border.
	Color (see page 604)	Specifies the background color of the control.
	Constraints (see page 604)	Specifies the size constraints for the control.
	CornerEdge (see page 605)	The size of the cornering for the edge of the bands.
	DockSite (see page 605)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 606)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 607)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 607)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 609)	This is DrawingStyle, a member of class TControlBar.
	Enabled (see page 609)	Controls whether the control responds to mouse, keyboard, and timer events.
	GradientDirection (see page 609)	Horizontal or Vertical.
	GradientEndColor (see page 609)	Stores the color that the gradient is painted with when it ends.
	GradientStartColor (see page 610)	Stores the color that the gradient is painted when it starts.
	ParentBackground (see page 610)	Determines whether control uses parent's theme background.
	ParentColor (see page 610)	Determines where a control looks for its color information.
	ParentCtl3D (see page 611)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 611)	Determines where a control looks for its font information.
	ParentShowHint (see page 612)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 612)	Specifies the control bar's picture.
	PopupMenu (see page 612)	Identifies the pop-up menu associated with the control.
	RowSize (see page 617)	Specifies the height of the control bar's rows.
	RowSnap (see page 618)	Specifies whether controls are snapped to the control bar's rows when docked.
	ShowHint (see page 618)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 620)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 620)	Determines if the user can tab to a control.
	Visible (see page 621)	Determines whether the component appears onscreen.

3.1.14.2.1 TControlBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
```

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
```

```
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.14.2.2 TControlBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.14.2.3 TControlBar.AutoDock Property

Determines whether the control bar temporarily docks drag clients to provide visual feedback.

Pascal

```
property AutoDock: Boolean;
```

C++

```
__property Boolean AutoDock;
```

Description

If AutoDock is true (the default), when a dockable object is dragged over the control bar, the control bar temporarily docks the object so that the user can see the impact of the drag operation. That is, the user can see the dock position and how other docked controls will be repositioned to make room. When the user releases the mouse, the dragged object is docked at the current spot, and any temporary dockings are canceled. However, if the user cancels the drag-and-dock operation, the dragged object reverts to its last temporary dock site rather than to its location at the start of the drag operation.

If AutoDock is false, the control bar does not temporarily dock dragged objects. Instead, the user only sees the dragging rectangle as it passes over the control bar, with no feedback as to the adjustments the control bar would make if the dragged object were released.

See Also

DockClients

DockSite

OnDockOver

3.1.14.2.4 TControlBar.AutoDrag Property

Specifies whether control bands can be undocked by dragging.

Pascal

```
property AutoDrag: Boolean;
```

C++

```
__property Boolean AutoDrag;
```

Description

Use AutoDrag to specify whether control bands are automatically undocked when the user drags them. If AutoDrag is true, control bands (TToolBar objects) are undocked when dragged off the control bar. Otherwise, control bands are kept within the control bar.

See Also

DragMode

OnBandDrag (see page 771)

onbandmove (see page 772)

3.1.14.2.5 TControlBar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.14.2.6 TControlBar.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.14.2.7 TControlBar.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.14.2.8 TControlBar.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.14.2.9 TControlBar.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.14.2.10 TControlBar.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

3.1.14.2.11 TControlBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth;
```

C++

```
__property BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter

3.1.14.2.12 TControlBar.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
```

```
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = dynamic_cast<TStringGrid *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint

3.1.14.2.13 TControlBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.14.2.14 TControlBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.14.2.15 TControlBar.CornerEdge Property

The size of the cornering for the edge of the bands.

Pascal

```
property CornerEdge: TCornerEdge;
```

C++

```
__property TCornerEdge CornerEdge;
```

Description

This property stores the size of the cornering for the edge of the bands.

3.1.14.2.16 TControlBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.14.2.17 TControlBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.14.2.18 TControlBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.14.2.19 TControlBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.14.2.20 TControlBar.DrawingStyle Property

Pascal

```
property DrawingStyle: TBandDrawingStyle;
```

C++

```
__property TBandDrawingStyle DrawingStyle;
```

Description

This is DrawingStyle, a member of class TControlBar.

3.1.14.2.21 TControlBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.14.2.22 TControlBar.GradientDirection Property

Horizontal or Vertical.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

Stores the direction of the gradient as Horizontal or Vertical.

3.1.14.2.23 TControlBar.GradientEndColor Property

Stores the color that the gradient is painted with when it ends.

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

Use this property to store the color that you want to use to end your drawing. (This blends with the start color.)

3.1.14.2.24 TControlBar.GradientStartColor Property

Stores the color that the gradient is painted when it starts.

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

Use this property to store the color that you want to use to initialize the drawing.

3.1.14.2.25 TControlBar.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.14.2.26 TControlBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.14.2.27 TControlBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.14.2.28 TControlBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.14.2.29 TControlBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.14.2.30 TControlBar.Picture Property

Specifies the control bar's picture.

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

Use Picture to specify the control bar's picture. The picture is tiled across the control bar.

See Also

OnBandPaint (see page 772)

OnPaint (see page 773)

3.1.14.2.31 TControlBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}
```

```
/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
```

```
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.14.2.32 TControlBar.RowSize Property

Specifies the height of the control bar's rows.

Pascal

```
property RowSize: TRowSize;
```

C++

```
__property TRowSize RowSize;
```

Description

Use RowSize to specify the height, in pixels, of the control bar's rows.

See Also

CanAutoSize (see page 757)

AutoSize

OnBandInfo (see page 771)

ClientHeight

3.1.14.2.33 TControlBar.RowSnap Property

Specifies whether controls are snapped to the control bar's rows when docked.

Pascal

```
property RowSnap: Boolean;
```

C++

```
__property Boolean RowSnap;
```

Description

Use RowSnap to specify whether controls are snapped to the control bar's rows when docked. Controls that are snapped have their Top and Height properties set automatically to correspond to the control bar's rows.

See Also

StickControls (see page 765)

Height

Top

3.1.14.2.34 TControlBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.14.2.35 TControlBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.14.2.36 TControlBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.14.2.37 TControlBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.14.3 TControlBar Events

The events of the TControlBar class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 623)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 624)	Occurs when an object with custom alignment is aligned.
	OnBandDrag (see page 624)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 625)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 625)	Occurs when a control band is moving.
	OnBandPaint (see page 626)	Occurs when a control band is painted.
	OnBeginBandMove (see page 626)	This is OnBeginBandMove, a member of class TControlBar.
	OnCanResize (see page 626)	Occurs when an attempt is made to resize the control.
	OnClick (see page 627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 628)	Adjust resize constraints.
	OnContextPopup (see page 629)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 630)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 630)	Occurs when another control is docked to the control.
	OnDockOver (see page 632)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 634)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 637)	Occurs when the user drags an object over a control.
	OnEndBandMove (see page 640)	This is OnEndBandMove, a member of class TControlBar.
	OnEndDock (see page 640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 641)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 642)	Occurs when a control receives the input focus.
	OnExit (see page 644)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 647)	Returns the control's docking information.
	OnMouseActivate (see page 649)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 650)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 652)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 653)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 653)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 655)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 657)	Occurs when the control bar is painted.
	OnResize (see page 658)	Occurs immediately after the control is resized.
	OnStartDock (see page 659)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 659)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 660)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.14.3.1 TControlBar.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

3.1.14.3.2 TControlBar.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

3.1.14.3.3 TControlBar.OnBandDrag Event

Occurs when the user starts a drag operation on a specified control.

Pascal

```
property OnBandDrag: TBandDragEvent;
```

C++

```
__property TBandDragEvent OnBandDrag;
```

Description

Use OnBandDrag to conditionally allow the user to start dragging a control in the control bar.

The event handler indicates the control that the user is trying to drag. Change its Drag parameter to false to prevent the drag operation. If Drag remains true on exit from the event handler, the drag operation proceeds.

See Also

OnStartDock

onbandmove (see page 772)

3.1.14.3.4 TControlBar.OnBandInfo Event

Occurs when the control bar adjusts the docking position of a band.

Pascal

```
property OnBandInfo: TBandInfoEvent;
```

C++

```
__property TBandInfoEvent OnBandInfo;
```

Description

Write code in the OnBandInfo event handler to provide information to the control bar that it can use when arranging its bands. The OnBandInfo event handler has several parameters that can be used to return information about how the band should be positioned, including its size, inset, and number of rows.

See Also

TBDockHeight

onbandmove (see page 772)

LRDockWidth

Height

Width

3.1.14.3.5 TControlBar.OnBandMove Event

Occurs when a control band is moving.

Pascal

```
property OnBandMove: TBandMoveEvent;
```

C++

```
__property TBandMoveEvent OnBandMove;
```

Description

Write an OnBandMove event handler to perform actions when a band is moving.

Use OnBandMove to specify the destination rectangle for the band. For example, adjust the destination rectangle so that its right edge matches the right edge of the control bar. This means the control extends to the control bar's right edge no matter where it is moved.

See Also

AutoDrag (see page 766)

RowSnap (see page 770)

3.1.14.3.6 TControlBar.OnBandPaint Event

Occurs when a control band is painted.

Pascal

```
property OnBandPaint: TBandPaintEvent;
```

C++

```
__property TBandPaintEvent OnBandPaint;
```

Description

Write an OnBandPaint event handler to customize the way the control bar draws its bands. From the event handler, you can draw to the control bar's canvas, using the dimensions of the band that are passed in as the ARect parameter. You can change the dimensions so that the automatically drawn frame and grabber appear in a different portion of the control bar (for example, indented slightly). You can suppress the automatic painting of the frame or grabber by removing options from the Options parameter.

See Also

OnPaint (see page 773)

3.1.14.3.7 TControlBar.OnBeginBandMove Event

Pascal

```
property OnBeginBandMove: TBeginBandMoveEvent;
```

C++

```
__property TBeginBandMoveEvent OnBeginBandMove;
```

Description

This is OnBeginBandMove, a member of class TControlBar.

3.1.14.3.8 TControlBar.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

3.1.14.3.9 TControlBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.14.3.10 TControlBar.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

3.1.14.3.11 TControlBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.14.3.12 TControlBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.14.3.13 TControlBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
```

```
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
```

```
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockOver

3.1.14.3.14 TControlBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
```

```
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
```

```
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.14.3.15 TControlBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
```

```
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
```

```
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.14.3.16 TControlBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
```

```
  label control to be dropped on a panel control and stops the
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  */
  void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

  {
    reinterpret_cast<TForm1 *>(Source)->Left = X;
    reinterpret_cast<TForm1 *>(Source)->Top = Y;
  }

  void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
            TDragState State, bool &Accept)
  {
    Accept = Source->ClassNameIs("TLabel");
  }

  void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
            TDragState State, bool &Accept)
  {
    Accept = false;
    if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
      dynamic_cast<TLabel *>(Source)->EndDrag(false);
  }
  /*
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  */

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
  void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
  TDragState State, bool &Accept)
  {
     Accept = Source->ClassNameIs("TLabel");
  }

  /*
  This OnDragDrop event handler implements the drop behavior.
  */
  void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
  {
    if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
    {
      TListBox *DestList = dynamic_cast<TListBox *>(Sender);
      DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
      DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
      DestList->DoubleBuffered = true;
      DestList->Color = clWindow;
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    ListBox1->Items->Add("Not");
    ListBox1->Items->Add("In");
    ListBox1->Items->Add("Alphabetical");
    ListBox1->Items->Add("Order");

  }
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.14.3.17 TControlBar.OnEndBandMove Event

Pascal

```
property OnEndBandMove: TEndBandMoveEvent;
```

C++

```
__property TEndBandMoveEvent OnEndBandMove;
```

Description

This is OnEndBandMove, a member of class TControlBar.

3.1.14.3.18 TControlBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.14.3.19 TControlBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.14.3.20 TControlBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
```

```
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.14.3.21 TControlBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;
```

```
    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.14.3.22 TControlBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
```

```
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
```

```
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.14.3.23 TControlBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is

over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.14.3.24 TControlBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
```

```
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.14.3.25 TControlBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
  property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.14.3.26 TControlBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.14.3.27 TControlBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
```

```
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.14.3.28 TControlBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
```

```
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.14.3.29 TControlBar.OnPaint Event

Occurs when the control bar is painted.

Pascal

```
  property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the control bar is rendered. When a control bar receives a Windows paint message, it first draws the background picture, if any, and then generates an OnPaint event. After the OnPaint event exits, the control bar generates an OnBandPaint event for each band, finally rendering the grabbers and frames for each band.

Use the Canvas property to draw the control bar's image. Use the ClientRect property to obtain the dimensions on which to draw.

If the custom drawing is intended to change the appearance of individual bands, use the OnBandPaint event instead.

See Also

onbandpaint (see page 772)

Canvas

ClientRect

picture (see page 770)

3.1.14.3.30 TControlBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

3.1.14.3.31 TControlBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.14.3.32 TControlBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.14.3.33 TControlBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
```

```
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

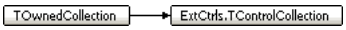
DockSite

OnDockDrop

OnDockOver

3.1.15 TControlCollection Class

Class Hierarchy



Pascal

```
TControlCollection = class(TOwnedCollection);
```

C++

```
class TControlCollection : public TOwnedCollection;
```

File

ExtCtrls

Description

This is class ExtCtrls.TControlCollection.

3.1.15.1 TControlCollection Members

The following tables list the members exposed by TControlCollection.

Protected Methods

	Name	Description
	GetControl (see page 665)	This is GetControl, a member of class TControlCollection.
	GetControlItem (see page 665)	This is GetControlItem, a member of class TControlCollection.
	GetItem (see page 665)	Returns a specified item in the collection.
	SetControl (see page 666)	This is SetControl, a member of class TControlCollection.
	SetItem (see page 666)	This is SetItem, a member of class TControlCollection.
V	Update (see page 667)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 664)	Creates a new TCollectionItem instance and adds it to the Items array.
	AddControl (see page 665)	This is AddControl, a member of class TControlCollection.
	Create (see page 664)	This is Create, a member of class TControlCollection.
	IndexOf (see page 666)	This is IndexOf, a member of class TControlCollection.
	Owner (see page 666)	Returns the Owner of the collection.
	RemoveControl (see page 666)	This is RemoveControl, a member of class TControlCollection.

Public Properties

	Name	Description
R	ControlItems (see page 667)	This is ControlItems, a member of class TControlCollection.
	Controls (see page 668)	This is Controls, a member of class TControlCollection.
	Items (see page 668)	Lists the items in the collection.

3.1.15.2 TControlCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TControlCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TControlCollection.

3.1.15.3 TControlCollection Methods

The methods of the TControlCollection class are listed here.

Protected Methods

	Name	Description
	GetControl (see page 665)	This is GetControl, a member of class TControlCollection.
	GetControlItem (see page 665)	This is GetControlItem, a member of class TControlCollection.
	GetItem (see page 665)	Returns a specified item in the collection.
	SetControl (see page 666)	This is SetControl, a member of class TControlCollection.
	SetItem (see page 666)	This is SetItem, a member of class TControlCollection.
V	Update (see page 667)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 664)	Creates a new TCollectionItem instance and adds it to the Items array.
	AddControl (see page 665)	This is AddControl, a member of class TControlCollection.
	IndexOf (see page 666)	This is IndexOf, a member of class TControlCollection.
	Owner (see page 666)	Returns the Owner of the collection.
	RemoveControl (see page 666)	This is RemoveControl, a member of class TControlCollection.

3.1.15.3.1 TControlCollection.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TControlItem;
```

C++

```
__fastcall TControlItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items

Insert

Create

Delete

3.1.15.3.2 TControlCollection.AddControl Method

Pascal

```
procedure AddControl(AControl: TControl; AColumn: Integer = -1; ARow: Integer = -1);
```

C++

```
__fastcall AddControl(TControl * AControl, int AColumn = -1, int ARow = -1);
```

Description

This is AddControl, a member of class TControlCollection.

3.1.15.3.3 TControlCollection.GetControl Method

Pascal

```
function GetControl(AColumn: Integer; ARow: Integer): TControl;
```

C++

```
__fastcall TControl * GetControl(int AColumn, int ARow);
```

Description

This is GetControl, a member of class TControlCollection.

3.1.15.3.4 TControlCollection.GetControlItem Method

Pascal

```
function GetControlItem(AColumn: Integer; ARow: Integer): TControlItem;
```

C++

```
__fastcall TControlItem GetControlItem(int AColumn, int ARow);
```

Description

This is GetControlItem, a member of class TControlCollection.

3.1.15.3.5 TControlCollection.GetItem Method

Returns a specified item in the collection.

Pascal

```
function GetItem(Index: Integer): TControlItem;
```

C++

```
__fastcall TControlItem GetItem(int Index);
```

Description

GetItem is the protected read implementation of the Items property.

See Also

Items

3.1.15.3.6 TControlCollection.IndexOf Method

Pascal

```
function IndexOf(AControl: TControl): Integer;
```

C++

```
__fastcall int IndexOf(TControl * AControl);
```

Description

This is IndexOf, a member of class TControlCollection.

3.1.15.3.7 TControlCollection.Owner Method

Returns the Owner of the collection.

Pascal

```
function Owner: TCustomGridPanel;
```

C++

```
__fastcall TCustomGridPanel Owner();
```

Description

Call Owner to obtain a reference to the object that owns this collection. Typically, the owner uses the collection to implement one of its properties.

3.1.15.3.8 TControlCollection.RemoveControl Method

Pascal

```
procedure RemoveControl(AControl: TControl);
```

C++

```
__fastcall RemoveControl(TControl * AControl);
```

Description

This is RemoveControl, a member of class TControlCollection.

3.1.15.3.9 TControlCollection.SetControl Method

Pascal

```
procedure SetControl(AColumn: Integer; ARow: Integer; Value: TControl);
```

C++

```
__fastcall SetControl(int AColumn, int ARow, TControl * Value);
```

Description

This is SetControl, a member of class TControlCollection.

3.1.15.3.10 TControlCollection.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TControlItem);
```

C++

```
__fastcall SetItem(int Index, TControlItem Value);
```

Description

This is SetItem, a member of class TControlCollection.

3.1.15.3.11 TControlCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed

Changed

3.1.15.4 TControlCollection Properties

The properties of the TControlCollection class are listed here.

Public Properties

	Name	Description
	ControlItems (see page 667)	This is ControlItems, a member of class TControlCollection.
	Controls (see page 668)	This is Controls, a member of class TControlCollection.
	Items (see page 668)	Lists the items in the collection.

3.1.15.4.1 TControlCollection.ControlItems Property

Pascal

```
property ControlItems [AColumn, ARow: Integer]: TControlItem;
```

C++

```
__property TControlItem ControlItems[int AColumn, ARow];
```

Description

This is ControlItems, a member of class TControlCollection.

3.1.15.4.2 TControlCollection.Controls Property

Pascal

```
property Controls [AColumn, ARow: Integer]: TControl;
```

C++

```
__property TControl * Controls[int AColumn, ARow];
```

Description

This is Controls, a member of class TControlCollection.

3.1.15.4.3 TControlCollection.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TControlItem;
```

C++

```
__property TControlItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count

ItemClass

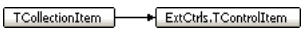
Index

Add

Insert

3.1.16 TControlItem Class

Class Hierarchy



Pascal

```
TControlItem = class(TCollectionItem);
```

C++

```
class TControlItem : public TCollectionItem;
```

File

ExtCtrls

Description

This is class ExtCtrls.TControlItem.

3.1.16.1 TControlItem Members

The following tables list the members exposed by TControlItem.

Protected Methods

	Name	Description
	AssignTo (see page 670)	Copies the properties of an object to a destination object.
	InternalSetLocation (see page 671)	This is InternalSetLocation, a member of class TControlItem.

Protected Properties

	Name	Description
	GridPanel (see page 672)	This is GridPanel, a member of class TControlItem.
	Pushed (see page 672)	This is Pushed, a member of class TControlItem.

Public Methods

	Name	Description
	Create (see page 669)	Creates and initializes a TCollectionItem instance.
	Destroy (see page 669)	Destroys the TCollectionItem instance and frees its memory.
	SetLocation (see page 671)	This is SetLocation, a member of class TControlItem.

Published Properties

	Name	Description
	Column (see page 672)	This is Column, a member of class TControlItem.
	ColumnSpan (see page 672)	This is ColumnSpan, a member of class TControlItem.
	Control (see page 672)	This is Control, a member of class TControlItem.
	Row (see page 673)	This is Row, a member of class TControlItem.
	RowSpan (see page 673)	This is RowSpan, a member of class TControlItem.

3.1.16.2 TControlItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TControlItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

3.1.16.3 TControlItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free

3.1.16.4 TControlItem Methods

The methods of the TControlItem class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 670)	Copies the properties of an object to a destination object.
	InternalSetLocation (see page 671)	This is InternalSetLocation, a member of class TControlItem.

Public Methods

	Name	Description
	SetLocation (see page 671)	This is SetLocation, a member of class TControlItem.

3.1.16.4.1 TControlItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

3.1.16.4.2 TControlItem.InternalSetLocation Method

Pascal

```
procedure InternalSetLocation(AColumn: Integer; ARow: Integer; APushed: Boolean; MoveExisting:
Boolean);
```

C++

```
__fastcall InternalSetLocation(int AColumn, int ARow, Boolean APushed, Boolean MoveExisting);
```

Description

This is InternalSetLocation, a member of class TControlItem.

3.1.16.4.3 TControlItem.SetLocation Method

Pascal

```
procedure SetLocation(AColumn: Integer; ARow: Integer; APushed: Boolean = False);
```

C++

```
__fastcall SetLocation(int AColumn, int ARow, Boolean APushed = False);
```

Description

This is SetLocation, a member of class TControlItem.

3.1.16.5 TControlItem Properties

The properties of the TControlItem class are listed here.

Protected Properties

	Name	Description
R	GridPanel (see page 672)	This is GridPanel, a member of class TControlItem.
R	Pushed (see page 672)	This is Pushed, a member of class TControlItem.

Published Properties

	Name	Description
	Column (see page 672)	This is Column, a member of class TControlItem.
	ColumnSpan (see page 672)	This is ColumnSpan, a member of class TControlItem.
	Control (see page 672)	This is Control, a member of class TControlItem.
	Row (see page 673)	This is Row, a member of class TControlItem.
	RowSpan (see page 673)	This is RowSpan, a member of class TControlItem.

3.1.16.5.1 TControlItem.Column Property

Pascal

```
property Column: Integer;
```

C++

```
__property int Column;
```

Description

This is Column, a member of class TControlItem.

3.1.16.5.2 TControlItem.ColumnSpan Property

Pascal

```
property ColumnSpan: TCellSpan;
```

C++

```
__property TCellSpan ColumnSpan;
```

Description

This is ColumnSpan, a member of class TControlItem.

3.1.16.5.3 TControlItem.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TControlItem.

3.1.16.5.4 TControlItem.GridPanel Property

Pascal

```
property GridPanel: TCustomGridPanel;
```

C++

```
__property TCustomGridPanel GridPanel;
```

Description

This is GridPanel, a member of class TControlItem.

3.1.16.5.5 TControlItem.Pushed Property

Pascal

```
property Pushed: Boolean;
```

C++

```
__property Boolean Pushed;
```

Description

This is Pushed, a member of class TControlItem.

3.1.16.5.6 TControlItem.Row Property

Pascal

```
property Row: Integer;
```

C++

```
__property int Row;
```

Description

This is Row, a member of class TControlItem.

3.1.16.5.7 TControlItem.RowSpan Property

Pascal

```
property RowSpan: TCellSpan;
```

C++

```
__property TCellSpan RowSpan;
```

Description

This is RowSpan, a member of class TControlItem.

3.1.17 TCustomButtonedEdit Class

TCustomButtonedEditis the base class for edit controls that contain embedded buttons.

Class Hierarchy

Pascal

```
TCustomButtonedEdit = class(TCustomEdit);
```

C++

```
class TCustomButtonedEdit : public TCustomEdit;
```

File

ExtCtrls

Description

A TCustomButtonedEdit is the base class for edit controls that have associated buttons. This class introduces new properties for managing buttons that sit to the sides of the edit control.

This class contains an Images (see page 682) property that holds an TImagelist that displays on the LeftButton (see page 685) and RightButton (see page 688) to indicate state. LeftButton (see page 685) and RightButton (see page 688) properties contain indexes into the Images (see page 682) TImageList to specify which image to use for which state. OnLeftButtonClick (see page 692) and OnRightButtonClick (see page 695) are event handlers for the buttons.

Do not create instances of TCustomButtonedEdit. Use TCustomButtonedEdit as a base class when declaring edit controls that include buttons. Properties and methods of TCustomButtonedEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

Images (see page 682)

LeftButton (see page 685)

RightButton (see page 688)

NumbersOnly (see page 252)

TextHint (see page 268)

TEditButton (see page 835)

3.1.17.1 TCustomButtonedEdit Members

The following tables list the members exposed by TCustomButtonedEdit.

Protected Methods

	Name	Description
	DoSetTextHint (see page 676)	This is DoSetTextHint, a member of class TCustomButtonedEdit.
	GetEditButtonClass (see page 676)	This is GetEditButtonClass, a member of class TCustomButtonedEdit.
	Notification (see page 676)	Responds to notifications that components are being created or destroyed.
	UpdateEditMargins (see page 677)	Update the margins of the edit control to provide adequate space for the buttons.
	WndProc (see page 677)	Provides specific message responses for the control.

Public Events

	Name	Description
	OnLeftButtonClick (see page 692)	Occurs when the user clicks the LeftButton (see page 685).
	OnRightButtonClick (see page 695)	Occurs when the user clicks the RightButton (see page 688).

Public Methods

	Name	Description
	Create (see page 674)	Creates an instance of TCustomEdit.
	DefaultHandler (see page 675)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 675)	Destroys an instance of TWinControl.

Public Properties

	Name	Description
	Images (see page 682)	Lists the images that can appear on the control's embedded buttons.
	LeftButton (see page 685)	Specifies the control that implements the left embedded button.
	RightButton (see page 688)	Specifies the control that implements the right embedded button.

3.1.17.2 TCustomButtonedEdit.Create Constructor

Creates an instance of TCustomEdit.

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomEdit.

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner

3.1.17.3 TCustomButtonedEdit.Destroy Destructor

Destroys an instance of TWinControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomButtonedEdit();
```

Description

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

See Also

System::TObject::Free

Destroy

3.1.17.4 TCustomButtonedEdit Methods

The methods of the TCustomButtonedEdit class are listed here.

Protected Methods

	Name	Description
	DoSetTextHint (see page 676)	This is DoSetTextHint, a member of class TCustomButtonedEdit.
	GetEditButtonClass (see page 676)	This is GetEditButtonClass, a member of class TCustomButtonedEdit.
	Notification (see page 676)	Responds to notifications that components are being created or destroyed.
	UpdateEditMargins (see page 677)	Update the margins of the edit control to provide adequate space for the buttons.
	WndProc (see page 677)	Provides specific message responses for the control.

Public Methods

	Name	Description
	DefaultHandler (see page 675)	Provides message handling for all messages that the control does not fully process itself.

3.1.17.4.1 TCustomButtonedEdit.DefaultHandler Method

Provides message handling for all messages that the control does not fully process itself.

Pascal

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

C++

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

Description

Override DefaultHandler to change the default message handling for the edit control. The Message parameter can be cast to a TMessage type, to obtain the WParam, LParam, and Result of the message. If the Result of the message is non-zero, the message has already been handled. Set the Result field to a non-zero value to prevent further processing of the message by the inherited method.

Note: In Delphi, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method if that ancestor does not specify a handler for the message being handled.

See Also

WindowProc

DefWndProc

TMessage

3.1.17.4.2 TCustomButtonedEdit.DoSetTextHint Method

Pascal

```
procedure DoSetTextHint(const Value: string); override;
```

C++

```
virtual __fastcall DoSetTextHint(const UnicodeString Value);
```

Description

This is DoSetTextHint, a member of class TCustomButtonedEdit.

3.1.17.4.3 TCustomButtonedEdit.GetEditButtonClass Method

Pascal

```
function GetEditButtonClass: TEditButtonClass; dynamic;
```

C++

```
__fastcall TEditButtonClass GetEditButtonClass();
```

Description

This is GetEditButtonClass, a member of class TCustomButtonedEdit.

3.1.17.4.4 TCustomButtonedEdit.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

3.1.17.4.5 TCustomButtonedEdit.UpdateEditMargins Method

Update the margins of the edit control to provide adequate space for the buttons.

Pascal

```
procedure UpdateEditMargins; dynamic;
```

C++

```
__fastcall UpdateEditMargins();
```

Description

Use UpdateEditMargins to set the margins of the edit control to provide adequate space for the buttons. The LeftButton (see page 685) or the RightButton (see page 688) must be visible for this to occur.

See Also

TEditButton (see page 835)

3.1.17.4.6 TCustomButtonedEdit.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
```

```
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}
```

```
type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
```

```
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc

MainWndProc

WndProc

WindowProc

3.1.17.5 TCustomButtonedEdit Properties

The properties of the TCustomButtonedEdit class are listed here.

Public Properties

	Name	Description
	Images (see page 682)	Lists the images that can appear on the control's embedded buttons.
	LeftButton (see page 685)	Specifies the control that implements the left embedded button.
	RightButton (see page 688)	Specifies the control that implements the right embedded button.

3.1.17.5.1 TCustomButtonedEdit.Images Property

Lists the images that can appear on the control's embedded buttons.

Pascal

```
property Images: TImageList;
```

C++

```
__property TImageList Images;
```

Description

Images contains a list of images that can appear on the control's embedded buttons. If Images is unassigned, no images are displayed on the buttons.

Images in the list should be 16x16.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;
```

```
    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
              mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
```

```
    public
      { Public declarations }
    end;

  var
    Form4: TForm4;

  implementation

  {$R *.dfm}

  procedure TForm4.btnEditEnter(Sender: Tobject);
  begin
    btnEdit.Text := '';
    btnEdit.RightButton.Visible := true;
  end;

  procedure TForm4.btnEditLeftClick(Sender: Tobject);
  begin
    MessageDlg('Left button pressed with: ' + btnEdit.Text,
               mtInformation, [mbYes], 0);

  end;

  procedure TForm4.btnEditRightClick(Sender: Tobject);
  begin
    //resetting to default state
     btnEdit.Text := 'Default';
     btnEdit.RightButton.Visible := false;
  end;

  procedure TForm4.FormCreate(Sender: TObject);
  var
    imList : TImageList;
    bmap   : TBitmap;
  begin
    //declaring an instance
      btnEdit := TButtonedEdit.Create(Self);

      //setting the parent so it will autodetele
      btnEdit.Parent := Self;

      //positioning
      btnEdit.Left := 10;
      btnEdit.Top := 10;

      //creating a TImageList and using it for the button
      imList := TImageList.Create(Self);
      bmap := TBitmap.Create();
      bmap.LoadFromFile('littleB_16.bmp');
      imList.Add(bmap,nil);
      bmap.LoadFromFile('little_logo_16.bmp');
      imList.Add(bmap,nil);
      btnEdit.Images := imList;

      //selecting the image for the left button and enabling it
      btnEdit.LeftButton.Visible := true;
      btnEdit.LeftButton.Enabled := true;
      btnEdit.LeftButton.ImageIndex := 0;
      btnEdit.LeftButton.DisabledImageIndex := 0;
      btnEdit.LeftButton.HotImageIndex := 0;

      //selecting the image for the right button
      //this will be enabled when the content changes
      btnEdit.RightButton.Enabled := true;
      btnEdit.RightButton.ImageIndex := 1;
```

```
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

TImageList

LeftButton (see page 685)

RightButton (see page 688)

3.1.17.5.2 TCustomButtonedEdit.LeftButton Property

Specifies the control that implements the left embedded button.

Pascal

```
property LeftButton: TEditButton;
```

C++

```
__property TEditButton LeftButton;
```

Description

Use LeftButton to work with the left button embedded in this edit control. Use this button's properties to specify images to correspond to the various states of the button, to disable the button, to toggle it's visiblity and to add a dropdown menu. At design time, expand the LeftButton property in the Object Inspector to set its properties.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;
```

```
    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
```

```
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
```

```
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

Images (see page 682)

RightButton (see page 688)

OnLeftButtonClick (see page 692)

TImageList

TEditButton (see page 835)

3.1.17.5.3 TCustomButtonedEdit.RightButton Property

Specifies the control that implements the right embedded button.

Pascal

```
property RightButton: TEditButton;
```

C++

```
__property TEditButton RightButton;
```

Description

Use RightButton to work with the right button embedded in this edit control. Use this button's properties to specify images to correspond to the various states of the button, to disable the button, to toggle it's visiblity and to add a dropdown menu. At design time, expand the RightButton property in the Object Inspector to set its properties.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
```

```
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
              mtInformation, TMsgDlgButtons() << mbOK, 0);
```

```
}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;
```

```
    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

Images (see page 682)

LeftButton (see page 685)

OnRightButtonClick (see page 695)

TImageList

TEditButton (see page 835)

3.1.17.6 TCustomButtonedEdit Events

The events of the TCustomButtonedEdit class are listed here.

Public Events

	Name	Description
	OnLeftButtonClick (see page 692)	Occurs when the user clicks the LeftButton (see page 685).
	OnRightButtonClick (see page 695)	Occurs when the user clicks the RightButton (see page 688).

3.1.17.6.1 TCustomButtonedEdit.OnLeftButtonClick Event

Occurs when the user clicks the LeftButton (see page 685).

Pascal

```
property OnLeftButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLeftButtonClick;
```

Description

Use the OnLeftButtonClick event handler to respond when the user clicks the LeftButton (see page 685) of the control.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;
```

```
    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
```

```
              mtInformation, [mbYes], 0);

  end;

  procedure TForm4.btnEditRightClick(Sender: Tobject);
  begin
    //resetting to default state
     btnEdit.Text := 'Default';
     btnEdit.RightButton.Visible := false;
  end;

  procedure TForm4.FormCreate(Sender: TObject);
  var
    imList : TImageList;
    bmap   : TBitmap;
  begin
    //declaring an instance
      btnEdit := TButtonedEdit.Create(Self);

      //setting the parent so it will autodetele
      btnEdit.Parent := Self;

      //positioning
      btnEdit.Left := 10;
      btnEdit.Top := 10;

      //creating a TImageList and using it for the button
      imList := TImageList.Create(Self);
      bmap := TBitmap.Create();
      bmap.LoadFromFile('littleB_16.bmp');
      imList.Add(bmap,nil);
      bmap.LoadFromFile('little_logo_16.bmp');
      imList.Add(bmap,nil);
      btnEdit.Images := imList;

      //selecting the image for the left button and enabling it
      btnEdit.LeftButton.Visible := true;
      btnEdit.LeftButton.Enabled := true;
      btnEdit.LeftButton.ImageIndex := 0;
      btnEdit.LeftButton.DisabledImageIndex := 0;
      btnEdit.LeftButton.HotImageIndex := 0;

      //selecting the image for the right button
      //this will be enabled when the content changes
      btnEdit.RightButton.Enabled := true;
      btnEdit.RightButton.ImageIndex := 1;
      btnEdit.RightButton.DisabledImageIndex := 1;
      btnEdit.RightButton.HotImageIndex := 1;

      //selecting action for the buttoned edit to take
      btnEdit.OnEnter := btnEditEnter;
      btnEdit.OnRightButtonClick := btnEditRightClick;
      btnEdit.OnLeftButtonClick := btnEditLeftClick;

      //default text in the edit
      btnEdit.Text := 'Default';

      btnEdit.Constraints.MinHeight := 16;
      btnEdit.Height := 16;

      //finally displaying the button
      btnEdit.Show();

  end;
```

See Also

LeftButton (see page 685)

OnRightButtonClick (see page 695)

TEditButton (see page 835)

3.1.17.6.2 TCustomButtonedEdit.OnRightButtonClick Event

Occurs when the user clicks the RightButton (see page 688).

Pascal

```
property OnRightButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRightButtonClick;
```

Description

Use the OnRightButtonClick event handler to respond when the user clicks the RightButton (see page 688) of the control.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
```

```
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
```

```
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
```

```
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

RightButton (see page 688)

OnLeftButtonClick (see page 692)

TEditButton (see page 835)

3.1.18 TCustomCategoryPanel Class

TCustomCategoryPanel is a base class for the TCategoryPanel (see page 309) class.

Class Hierarchy

Pascal

```
TCustomCategoryPanel = class(TCustomPanel);
```

C++

```
class TCustomCategoryPanel : public TCustomPanel;
```

File

ExtCtrls

Description

TCustomCategoryPanel is a base class for the TCategoryPanel (see page 309) class. All of the class members for TCategoryPanel (see page 309) originate in TCustomCategoryPanel.

See Also

TCustomCategoryPanelGroup (see page 718)

3.1.18.1 TCustomCategoryPanel Members

The following tables list the members exposed by TCustomCategoryPanel.

Enumerations

	Name	Description
	THeaderState (see page 701)	THeaderState is the type of the HeaderState (see page 717) property for a category panel.

Protected Fields

	Name	Description
	FPanelSurface (see page 702)	This is FPanelSurface, a member of class TCustomCategoryPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.

	CalcRectBounds (see page 703)	Sets the rectangles used to display the text and glyphs of the category panel.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	DefineProperties (see page 703)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DrawCollapsedPanel (see page 704)	Displays the category panel in the collapsed state.
	DrawHeader (see page 704)	This is the overview for the DrawHeader method overload.
	DrawHeaderBackground (see page 705)	This is DrawHeaderBackground, a member of class TCustomCategoryPanel.
	DrawHeaderCaption (see page 705)	This is DrawHeaderCaption, a member of class TCustomCategoryPanel.
	DrawHeaderChevron (see page 705)	This is DrawHeaderChevron, a member of class TCustomCategoryPanel.
	GetCollapsedHeight (see page 706)	This is GetCollapsedHeight, a member of class TCustomCategoryPanel.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	Paint (see page 806)	Renders the image of the panel.
	ReadState (see page 707)	Prepares the control for having its properties assigned values from a stream.
	ResizeHeader (see page 708)	This is ResizeHeader, a member of class TCustomCategoryPanel.
	SetParent (see page 708)	Sets the parent of the control.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.
	UpdateControlOriginalParentSize (see page 709)	This is UpdateControlOriginalParentSize, a member of class TCustomCategoryPanel.
	UpdateHeader (see page 709)	This is UpdateHeader, a member of class TCustomCategoryPanel.
	WndProc (see page 709)	Provides specific message responses for the control.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 812)	Specifies the background color of the control.
	FullRepaint (see page 813)	Determines how the panel repaints itself when it is resized.
R	HeaderHeight (see page 717)	Retrieves the HeaderHeight property associated with the parent TCustomCategoryPanelGroup (see page 718) HeaderHeight.
	Locked (see page 813)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 814)	Determines where a control looks for its color information.
	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Events

	Name	Description
	OnCollapse (see page 718)	Occurs immediately after a category panel is collapsed.
	OnExpand (see page 718)	Occurs immediately after a category panel is expanded.

Public Methods

	Name	Description
	Collapse (see page 703)	Collapses the category panel.
	Create (see page 700)	Creates and initializes an instance of TCustomPanel.
	Destroy (see page 700)	Destroys an instance of TCustomControl.
	Expand (see page 705)	Expands the category panel.
	GetChildren (see page 706)	Calls a specified procedure for every control in the category panel.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	GetTabOrderList (see page 706)	Builds a list of controls in tab order.
	SetBounds (see page 708)	Changes the position and size of the category panel in a single operation.

Public Properties

	Name	Description
	Collapsed (see page 714)	Specifies whether the category panel is collapsed.
	CollapsedHotImageIndex (see page 714)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.
	CollapsedImageIndex (see page 715)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..
	CollapsedPressedImageIndex (see page 715)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .
	ExpandedHotImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.
	ExpandedImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.
	ExpandedPressedImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.
R	HeaderState (see page 717)	Shows the state of the header of category panel.
	PanelGroup (see page 717)	Indentifies the parent category panel group of the category panel.
	ParentBackground (see page 813)	Determines whether control uses parent's theme background.

3.1.18.2 TCustomCategoryPanel.Create Constructor

Creates and initializes an instance of TCustomPanel.

Pascal

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

C++

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

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner

3.1.18.3 TCustomCategoryPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomCategoryPanel();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

3.1.18.4 TCustomCategoryPanel Enumerations

The enumerations of the TCustomCategoryPanel class are listed here.

Enumerations

	Name	Description
	THeaderState (see page 701)	THeaderState is the type of the HeaderState (see page 717) property for a category panel.

3.1.18.4.1 ExtCtrls.TCustomCategoryPanel.THeaderState Enumeration

THeaderState is the type of the HeaderState (see page 717) property for a category panel.

Pascal

```
THeaderState = (
  hsNormal,
  hsHot,
  hsPressed
);
```

C++

```
enum THeaderState {
  hsNormal,
  hsHot,
  hsPressed
};
```

File

ExtCtrls

Description

These are the possible value of the THeaderState type:

Value	Meaning
hsNormal	Mouse cursor is not hovering over header
hsHot	Mouse cursor is hovering over header

hsPressed	Mouse cursor is hovering over header and left mouse button is depressed

3.1.18.5 TCustomCategoryPanel Fields

The fields of the TCustomCategoryPanel class are listed here.

Protected Fields

	Name	Description
	FPanelSurface (see page 702)	This is FPanelSurface, a member of class TCustomCategoryPanel.

3.1.18.5.1 TCustomCategoryPanel.FPanelSurface Field

Pascal

```
FPanelSurface: TCategoryPanelSurface;
```

C++

```
TCategoryPanelSurface FPanelSurface;
```

Description

This is FPanelSurface, a member of class TCustomCategoryPanel.

3.1.18.6 TCustomCategoryPanel Methods

The methods of the TCustomCategoryPanel class are listed here.

Protected Methods

	Name	Description
	CalcRectBounds (see page 703)	Sets the rectangles used to display the text and glyphs of the category panel.
	DefineProperties (see page 703)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DrawCollapsedPanel (see page 704)	Displays the category panel in the collapsed state.
	DrawHeader (see page 704)	This is the overview for the DrawHeader method overload.
	DrawHeaderBackground (see page 705)	This is DrawHeaderBackground, a member of class TCustomCategoryPanel.
	DrawHeaderCaption (see page 705)	This is DrawHeaderCaption, a member of class TCustomCategoryPanel.
	DrawHeaderChevron (see page 705)	This is DrawHeaderChevron, a member of class TCustomCategoryPanel.
	GetCollapsedHeight (see page 706)	This is GetCollapsedHeight, a member of class TCustomCategoryPanel.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	ReadState (see page 707)	Prepares the control for having its properties assigned values from a stream.
	ResizeHeader (see page 708)	This is ResizeHeader, a member of class TCustomCategoryPanel.
	SetParent (see page 708)	Sets the parent of the control.
	UpdateControlOriginalParentSize (see page 709)	This is UpdateControlOriginalParentSize, a member of class TCustomCategoryPanel.
	UpdateHeader (see page 709)	This is UpdateHeader, a member of class TCustomCategoryPanel.
	WndProc (see page 709)	Provides specific message responses for the control.

Public Methods

	Name	Description
	Collapse (see page 703)	Collapses the category panel.
	Expand (see page 705)	Expands the category panel.
	GetChildren (see page 706)	Calls a specified procedure for every control in the category panel.
	GetTabOrderList (see page 706)	Builds a list of controls in tab order.

	SetBounds (see page 708)	Changes the position and size of the category panel in a single operation.

3.1.18.6.1 TCustomCategoryPanel.CalcRectBounds Method

Sets the rectangles used to display the text and glyphs of the category panel.

Pascal

```
procedure CalcRectBounds; virtual;
```

C++

```
virtual __fastcall CalcRectBounds();
```

Description

Changes the position and size of the glyph rect and text used in the DrawHeaderChevron (see page 705) and UpdateButtonState methods by using the themes, header and group images information.

See Also

DrawHeaderChevron (see page 705)

3.1.18.6.2 TCustomCategoryPanel.Collapse Method

Collapses the category panel.

Pascal

```
procedure Collapse;
```

C++

```
__fastcall Collapse();
```

Description

Collapse is called automatically when the user clicks on the header of an expanded category panel. This function will set the Collapsed (see page 714) property to true.

See Also

Collapsed (see page 714)

3.1.18.6.3 TCustomCategoryPanel.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer

parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty

DefineProperty

DefineProperties

IsControl

3.1.18.6.4 TCustomCategoryPanel.DrawCollapsedPanel Method

Displays the category panel in the collapsed state.

Pascal

```
procedure DrawCollapsedPanel(ACanvas: TCanvas); virtual;
```

C++

```
virtual __fastcall DrawCollapsedPanel(TCanvas ACanvas);
```

Description

Displays the category panel in the collapsed state.

3.1.18.6.5 DrawHeader Method

This is the overview for the DrawHeader method overload.

Overload List

	Name	Description
	TCustomCategoryPanel.DrawHeader () (see page 704)	This is DrawHeader, a member of class TCustomCategoryPanel.
	TCustomCategoryPanel.DrawHeader (HDC) (see page 704)	This is DrawHeader, a member of class TCustomCategoryPanel.

3.1.18.6.5.1 TCustomCategoryPanel.DrawHeader Method ()

Pascal

```
procedure DrawHeader; virtual; overload;
```

C++

```
virtual __fastcall DrawHeader();
```

Description

This is DrawHeader, a member of class TCustomCategoryPanel.

3.1.18.6.5.2 TCustomCategoryPanel.DrawHeader Method (HDC)

Pascal

```
procedure DrawHeader(DC: HDC); virtual; overload;
```

C++

```
virtual __fastcall DrawHeader(HDC DC);
```

Description

This is DrawHeader, a member of class TCustomCategoryPanel.

3.1.18.6.6 TCustomCategoryPanel.DrawHeaderBackground Method

Pascal

```
procedure DrawHeaderBackground(ACanvas: TCanvas); virtual;
```

C++

```
virtual __fastcall DrawHeaderBackground(TCanvas ACanvas);
```

Description

This is DrawHeaderBackground, a member of class TCustomCategoryPanel.

3.1.18.6.7 TCustomCategoryPanel.DrawHeaderCaption Method

Pascal

```
procedure DrawHeaderCaption(ACanvas: TCanvas); virtual;
```

C++

```
virtual __fastcall DrawHeaderCaption(TCanvas ACanvas);
```

Description

This is DrawHeaderCaption, a member of class TCustomCategoryPanel.

3.1.18.6.8 TCustomCategoryPanel.DrawHeaderChevron Method

Pascal

```
procedure DrawHeaderChevron(ACanvas: TCanvas); virtual;
```

C++

```
virtual __fastcall DrawHeaderChevron(TCanvas ACanvas);
```

Description

This is DrawHeaderChevron, a member of class TCustomCategoryPanel.

3.1.18.6.9 TCustomCategoryPanel.Expand Method

Expands the category panel.

Pascal

```
procedure Expand;
```

C++

```
__fastcall Expand();
```

Description

Expand is called automatically when the user clicks on the header of a collapsed category panel. This function will set the Collapsed (see page 714) property to false.

See Also

Collapse (see page 703)

Collapsed (see page 714)

3.1.18.6.10 TCustomCategoryPanel.GetChildren Method

Calls a specified procedure for every control in the category panel.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved.

The **Proc** parameter defines the callback.

The **Root** parameter is ignored.

3.1.18.6.11 TCustomCategoryPanel.GetCollapsedHeight Method

Pascal

```
function GetCollapsedHeight: Integer; virtual;
```

C++

```
virtual __fastcall int GetCollapsedHeight();
```

Description

This is GetCollapsedHeight, a member of class TCustomCategoryPanel.

3.1.18.6.12 TCustomCategoryPanel.GetTabOrderList Method

Builds a list of controls in tab order.

Pascal

```
procedure GetTabOrderList(List: TList); override;
```

C++

```
virtual __fastcall GetTabOrderList(TList List);
```

Description

Call GetTabOrderList to construct a list of child controls in tab order.

GetTabOrderList iterates through the control's internal tab-order list, adding each of the child controls to List, including any controls contained in those controls. The result is a list of all the controls and their owned controls, in tab order.

The FindNextControl method calls GetTabOrderList to build a complete list of the controls that FindNextControl uses to locate

the next control in the tab order.

If the category panel is collapsed, GetTabOrderList returns an empty list.

3.1.18.6.13 TCustomCategoryPanel.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

3.1.18.6.14 TCustomCategoryPanel.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

3.1.18.6.15 TCustomCategoryPanel.ResizeHeader Method

Pascal

```
procedure ResizeHeader(OldHeight: Integer; NewHeight: Integer);
```

C++

```
__fastcall ResizeHeader(int OldHeight, int NewHeight);
```

Description

This is ResizeHeader, a member of class TCustomCategoryPanel.

3.1.18.6.16 TCustomCategoryPanel.SetBounds Method

Changes the position and size of the category panel in a single operation.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Changes the position and size of the category panel within the constraints of the category panel group. SetBounds will not allow the height to be less than the header height of the category panel. Other category panels within the same category panel group will be adjusted accordingly. By calling SetBounds instead of adjusting the **LeftTop**, **Width**, and **Height** individually, you ensure that the panel will not repaint between changes.

3.1.18.6.17 TCustomCategoryPanel.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

3.1.18.6.18 TCustomCategoryPanel.UpdateControlOriginalParentSize Method

Pascal

```
procedure UpdateControlOriginalParentSize(AControl: TControl; var AOriginalParentSize:
TPoint); override;
```

C++

```
virtual __fastcall UpdateControlOriginalParentSize(TControl * AControl, TPoint
AOriginalParentSize);
```

Description

This is UpdateControlOriginalParentSize, a member of class TCustomCategoryPanel.

3.1.18.6.19 TCustomCategoryPanel.UpdateHeader Method

Pascal

```
procedure UpdateHeader;
```

C++

```
__fastcall UpdateHeader();
```

Description

This is UpdateHeader, a member of class TCustomCategoryPanel.

3.1.18.6.20 TCustomCategoryPanel.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
```

```
  called WM_STYLEMESSAGE. The subclassed window procedure can
  be turned on or off by pressing a radio button.
  */
  class TMyListBoxDescendant : public TListBox
  {
  __published:    // IDE-managed Components
      void __fastcall SubClassWndProc(Messages::TMessage &Message);
      void __fastcall ToggleSubClass(bool On);
      void __fastcall OnDrawItemProc(
        TWinControl *Control, int Index, const TRect &Rect,
        TOwnerDrawState State);
  private:    // User declarations
  public:     // User declarations
      __fastcall TMyListBoxDescendant(TComponent* Owner);
  };

  TForm1 *Form1;
  TMyListBoxDescendant *MyListBoxDescendant1;
  Graphics::TBitmap *bitmap0;

  const WM_STYLEMESSAGE = WM_USER + 2000;

  __fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
  {
  }

  __fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
      : TListBox(Owner)
  {
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    PostMessage(
      MyListBoxDescendant1->Handle,
      WM_STYLEMESSAGE,
      Integer(lbOwnerDrawFixed),
      0);
  }

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    PostMessage(
      MyListBoxDescendant1->Handle,
      WM_STYLEMESSAGE,
      Integer(lbStandard),
      0);
  }

  void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
  {
    MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
  }

  void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
  {
    if (Message.Msg == WM_STYLEMESSAGE)
      Style = (TListBoxStyle)Message.WParam;
    else
      WndProc(Message);
  }

  void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
  {
    if (On)
```

```
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
```

```
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;
```

```
procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc

MainWndProc

WndProc

WindowProc

3.1.18.7 TCustomCategoryPanel Properties

The properties of the TCustomCategoryPanel class are listed here.

Protected Properties

	Name	Description
	HeaderHeight (see page 717)	Retrieves the HeaderHeight property associated with the parent TCustomCategoryPanelGroup (see page 718) HeaderHeight.

Public Properties

	Name	Description
	Collapsed (see page 714)	Specifies whether the category panel is collapsed.
	CollapsedHotImageIndex (see page 714)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.
	CollapsedImageIndex (see page 715)	Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..
	CollapsedPressedImageIndex (see page 715)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .
	ExpandedHotImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.
	ExpandedImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.
	ExpandedPressedImageIndex (see page 716)	Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.
R	HeaderState (see page 717)	Shows the state of the header of category panel.
	PanelGroup (see page 717)	Indentifies the parent category panel group of the category panel.

3.1.18.7.1 TCustomCategoryPanel.Collapsed Property

Specifies whether the category panel is collapsed.

Pascal

```
property Collapsed: Boolean;
```

C++

```
__property Boolean Collapsed;
```

Description

Use Collapsed to determine if the category panel is in a collapsed state. It can also be used to cause the category panel to collapse or expand.

See Also

Collapse (see page 703)

Expand (see page 705)

3.1.18.7.2 TCustomCategoryPanel.CollapsedHotImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is hovering over the header.

Pascal

```
property CollapsedHotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedHotImageIndex;
```

Description

Set CollapsedHotImageIndex to associate the cheveron with one of the images in the Images property of the parent CategoryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.3 TCustomCategoryPanel.CollapsedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the mouse is not hovering over the header..

Pascal

```
property CollapsedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedImageIndex;
```

Description

Set CollapsedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the mouse is outside of the area of the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.4 TCustomCategoryPanel.CollapsedPressedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers the header. .

Pascal

```
property CollapsedPressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex CollapsedPressedImageIndex;
```

Description

Set CollapsedPressedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state while the left mouse button is pressed while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.5 TCustomCategoryPanel.ExpandedHotImageIndex Property

Determines which image is displayed as the chevron when the category panel is expanded and the mouse is hovering over the header.

Pascal

```
property ExpandedHotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedHotImageIndex;
```

Description

Set ExpandedHotImageIndex to associate the cheveron with one of the images in the Images property of the parent CategoryPanelGroup. The selected image will be displayed when the category panel is in the expanded state while the mouse hovers over the header. This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.6 TCustomCategoryPanel.ExpandedImageIndex Property

Determines which image is displayed as the chevron when the category panel is expanded and the mouse is not hovering over the header.

Pascal

```
property ExpandedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedImageIndex;
```

Description

Set ExpandedImageIndex to associate the cheveron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the expanded state while the mouse is outside of the area of the header.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.7 TCustomCategoryPanel.ExpandedPressedImageIndex Property

Determines which image is displayed as the chevron when the category panel is collapsed and the left mouse button is pressed while the mouse hovers over the header.

Pascal

```
property ExpandedPressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ExpandedPressedImageIndex;
```

Description

Set CollapsedPressedImageIndex (see page 715) to associate the chevron with one of the images in the Images property of the parent CateogryPanelGroup. The selected image will be displayed when the category panel is in the collapsed state and the left mouse button is pressed while the mouse hovers over the header.

This property has no effect if the HeaderStyle property of the parent panel group is set to hsThemed.

See Also

HeaderStyle (see page 734)

Images (see page 735)

3.1.18.7.8 TCustomCategoryPanel.HeaderHeight Property

Retrieves the HeaderHeight property associated with the parent TCustomCategoryPanelGroup (see page 718) HeaderHeight.

Pascal

```
property HeaderHeight: Integer;
```

C++

```
__property int HeaderHeight;
```

Description

HeaderHeight is a read-only property used to access the HeaderHeight property of the parent category panel group.

3.1.18.7.9 TCustomCategoryPanel.HeaderState Property

Shows the state of the header of category panel.

Pascal

```
property HeaderState: THeaderState;
```

C++

```
__property THeaderState HeaderState;
```

Description

Use HeaderState to determine the current state of the header of the category panel. This is a read-only property.

Value	Meaning
hsNormal	Mouse cursor is not hovering over header .
hsHot	Mouse cursor is hovering over header.
hsPressed	Mouse cursor is hovering over header and left mouse button is depressed.

3.1.18.7.10 TCustomCategoryPanel.PanelGroup Property

Indentifies the parent category panel group of the category panel.

Pascal

```
property PanelGroup: TCustomCategoryPanelGroup;
```

C++

```
__property TCustomCategoryPanelGroup PanelGroup;
```

Description

Use the PanelGroup property to get or set the parent panel group of this category panel. The category panel will adopt properties of the new category panel group (e.g. Width, HeaderHeight (see page 717), Images).

3.1.18.8 TCustomCategoryPanel Events

The events of the TCustomCategoryPanel class are listed here.

Public Events

	Name	Description
	OnCollapse (see page 718)	Occurs immediately after a category panel is collapsed.
	OnExpand (see page 718)	Occurs immediately after a category panel is expanded.

3.1.18.8.1 TCustomCategoryPanel.OnCollapse Event

Occurs immediately after a category panel is collapsed.

Pascal

```
property OnCollapse: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a category panel is collapsed.

See Also

OnExpand (see page 718)

3.1.18.8.2 TCustomCategoryPanel.OnExpand Event

Occurs immediately after a category panel is expanded.

Pascal

```
property OnExpand: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a category panel is expanded.

See Also

OnCollapse (see page 718)

3.1.19 TCustomCategoryPanelGroup Class

Class Hierarchy

Pascal

```
TCustomCategoryPanelGroup = class(TScrollingWinControl);
```

C++

```
class TCustomCategoryPanelGroup : public TScrollingWinControl;
```

File

ExtCtrls

Description

This is class ExtCtrls.TCustomCategoryPanelGroup.

3.1.19.1 TCustomCategoryPanelGroup Members

The following tables list the members exposed by TCustomCategoryPanelGroup.

Enumerations

	Name	Description
	THeaderStyle (see page 721)	Specifies the appearance of the headers of the the category panels.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 723)	Creates a Windows control corresponding to the TScrollingWinControl object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	GetCategoryPanelClass (see page 723)	This is GetCategoryPanelClass, a member of class TCustomCategoryPanelGroup.
	Loaded (see page 724)	Initializes the control after it is loaded from a stream.
	Notification (see page 724)	Responds to notifications that components are being created or destroyed.
	ReorderPanels (see page 725)	This is ReorderPanels, a member of class TCustomCategoryPanelGroup.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	WndProc (see page 725)	Provides specific message responses for the control.

Protected Properties

	Name	Description
	AutoScroll (see page 2561)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Public Methods

	Name	Description
	CollapseAll (see page 722)	Collapses all the category panels within the category panel group.
	Create (see page 720)	Creates and initializes a new TScrollingWinControl object.
	CreatePanel (see page 722)	Creates a new category panel for the category panel group.
	Destroy (see page 721)	Removes the TScrollingWinControl object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	ExpandAll (see page 723)	Expands all the category panels within the category panel group.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Public Properties

	Name	Description
	ChevronAlignment (see page 730)	Determines how the chevron is aligned within the header of the category panels in this category panel group.
	ChevronColor (see page 730)	Specifies the color of the default chevron in the normal state.
	ChevronHotColor (see page 731)	Specifies the color of the default chevron in the hot state.
	Color (see page 731)	Specifies the background color of the control.
	GradientBaseColor (see page 732)	Specifies the left or top color of the panel header in gradient style.
	GradientColor (see page 732)	Specifies the right or bottom color of the panel header in gradient style.
	GradientDirection (see page 733)	Specifies the direction of the gradient in the panel header.
	HeaderAlignment (see page 733)	Determines how the caption is aligned within the headers of the category panels in this category panel group.
	HeaderFont (see page 733)	Specifies the font of the caption of the headers of the panels.
	HeaderHeight (see page 734)	Specifies the height of the headers of the panels.
	HeaderImage (see page 734)	Specifies the image used for the background of the headers of the panels.
	HeaderStyle (see page 734)	Specifies the appearance of the headers of the the category panels.
	Images (see page 735)	Specifies an imagelist that can be used by the panels to replace the chevron. images.
R	Panels (see page 735)	Lists the panels (TCategoryPanel (see page 309) objects) in the category panel group.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.

3.1.19.2 TCustomCategoryPanelGroup.Create Constructor

Creates and initializes a new TScrollingWinControl object.

Pascal

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

C++

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

Description

Use Create to create and initialize a new TScrollingWinControl object.

AOwner is a TComponent object that owns the scrolling windowed control.

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

Create

VertScrollBar (see page 2563)

3.1.19.3 TCustomCategoryPanelGroup.Destroy Destructor

Removes the TScrollingWinControl object from memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomCategoryPanelGroup();
```

Description

Do not call Destroy.TScrollingWinControl Instead, call Free, which calls DestroyWinControl if the TScrollingWinControl object is not nil.

See Also

Destroy

System::TObject::Free

3.1.19.4 TCustomCategoryPanelGroup Enumerations

The enumerations of the TCustomCategoryPanelGroup class are listed here.

Enumerations

	Name	Description
	THeaderStyle (see page 721)	Specifies the appearance of the headers of the the category panels.

3.1.19.4.1 ExtCtrls.TCustomCategoryPanelGroup.THeaderStyle Enumeration

Specifies the appearance of the headers of the the category panels.

Pascal

```
THeaderStyle = (
  hsGradient,
  hsImage,
  hsThemed
);
```

C++

```
enum THeaderStyle {
  hsGradient,
  hsImage,
  hsThemed
};
```

File

ExtCtrls

Description

THeaderStyle is the type of the HeaderStyle (see page 734) property for a category panel group. These are the possible values of the THeaderStyle type:

Value	Meaning
hsGradient	Header is a gradient determined by the GradientBaseColor (see page 732), GradientColor (see page 732), and GradientDirection (see page 733) properties.
hsImage	Header is the stretched image defined by the HeaderImage (see page 734) property. If HeaderImage (see page 734) is undefined, the hsGradient style will be used instead.
hsThemed	Header is defined by the system's theme. If runtime themes aren't enabled or supported, the hsGradient style will be used instead.

3.1.19.5 TCustomCategoryPanelGroup Methods

The methods of the TCustomCategoryPanelGroup class are listed here.

Protected Methods

	Name	Description
	CreateWnd (see page 723)	Creates a Windows control corresponding to the TScrollingWinControl object.
	GetCategoryPanelClass (see page 723)	This is GetCategoryPanelClass, a member of class TCustomCategoryPanelGroup.
	Loaded (see page 724)	Initializes the control after it is loaded from a stream.
	Notification (see page 724)	Responds to notifications that components are being created or destroyed.
	ReorderPanels (see page 725)	This is ReorderPanels, a member of class TCustomCategoryPanelGroup.
	WndProc (see page 725)	Provides specific message responses for the control.

Public Methods

	Name	Description
	CollapseAll (see page 722)	Collapses all the category panels within the category panel group.
	CreatePanel (see page 722)	Creates a new category panel for the category panel group.
	ExpandAll (see page 723)	Expands all the category panels within the category panel group.

3.1.19.5.1 TCustomCategoryPanelGroup.CollapseAll Method

Collapses all the category panels within the category panel group.

Pascal

```
procedure CollapseAll;
```

C++

```
__fastcall CollapseAll();
```

Description

Call CollapseAll to collapse all the category panels within the category panel group.

See Also

Collapse (see page 703)

ExpandAll (see page 723)

3.1.19.5.2 TCustomCategoryPanelGroup.CreatePanel Method

Creates a new category panel for the category panel group.

Pascal

```
function CreatePanel(AOwner: TComponent): TCustomCategoryPanel;
```

C++

```
__fastcall TCustomCategoryPanel CreatePanel(TComponent * AOwner);
```

Description

Call CreatePanel to create a TCategoryPanel (see page 309) object and assign it to the panel group.

AOwner is the owner of the TCustomCategoryPanel (see page 698) object and is responsible for freeing that object during cleanup.

3.1.19.5.3 TCustomCategoryPanelGroup.CreateWnd Method

Creates a Windows control corresponding to the TScrollingWinControl object.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the window must be destroyed and recreated to reflect property changes.

See Also

CreateWnd

DestroyWnd

3.1.19.5.4 TCustomCategoryPanelGroup.ExpandAll Method

Expands all the category panels within the category panel group.

Pascal

```
procedure ExpandAll;
```

C++

```
__fastcall ExpandAll();
```

Description

Call ExpandAll to expand all the category panels within the category panel group.

See Also

Expand (see page 705)

CollapseAll (see page 722)

3.1.19.5.5 TCustomCategoryPanelGroup.GetCategoryPanelClass Method

Pascal

```
function GetCategoryPanelClass: TCategoryPanelClass; virtual;
```

C++

```
virtual __fastcall TCategoryPanelClass GetCategoryPanelClass();
```

Description

This is GetCategoryPanelClass, a member of class TCustomCategoryPanelGroup.

3.1.19.5.6 TCustomCategoryPanelGroup.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

3.1.19.5.7 TCustomCategoryPanelGroup.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

3.1.19.5.8 TCustomCategoryPanelGroup.ReorderPanels Method

Pascal

```
procedure ReorderPanels; dynamic;
```

C++

```
__fastcall ReorderPanels();
```

Description

This is ReorderPanels, a member of class TCustomCategoryPanelGroup.

3.1.19.5.9 TCustomCategoryPanelGroup.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;
```

```
const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
```

```
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
    end;

  var
    Form1: TForm1;
    MyListBoxDescendant1: TMyListBoxDescendant;
    bitmap0: TBitmap;

  implementation

  {$R *.dfm}

  const WM_STYLEMESSAGE = WM_USER + 2000;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbOwnerDrawFixed),
      0);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbStandard),
      0);
  end;

  procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
  begin
    if (Message.Msg = WM_STYLEMESSAGE) then
      Style:= TListBoxStyle(Message.WParam)
    else
      WndProc(Message);
  end;

  procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
  begin
    if On then
      WindowProc := SubClassWndProc
    else
      WindowProc := WndProc;
  end;

  procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
  begin
    MyListBoxDescendant1.ToggleSubClass(
      SubClassRadioGroup1.ItemIndex = 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;
```

```
    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DefWndProc

MainWndProc

WndProc

WindowProc

3.1.19.6 TCustomCategoryPanelGroup Properties

The properties of the TCustomCategoryPanelGroup class are listed here.

Public Properties

	Name	Description
	ChevronAlignment (see page 730)	Determines how the chevron is aligned within the header of the category panels in this category panel group.
	ChevronColor (see page 730)	Specifies the color of the default chevron in the normal state.
	ChevronHotColor (see page 731)	Specifies the color of the default chevron in the hot state.
	Color (see page 731)	Specifies the background color of the control.
	GradientBaseColor (see page 732)	Specifies the left or top color of the panel header in gradient style.
	GradientColor (see page 732)	Specifies the right or bottom color of the panel header in gradient style.
	GradientDirection (see page 733)	Specifies the direction of the gradient in the panel header.

	HeaderAlignment (see page 733)	Determines how the caption is aligned within the headers of the category panels in this category panel group.
	HeaderFont (see page 733)	Specifies the font of the caption of the headers of the panels.
	HeaderHeight (see page 734)	Specifies the height of the headers of the panels.
	HeaderImage (see page 734)	Specifies the image used for the background of the headers of the panels.
	HeaderStyle (see page 734)	Specifies the appearance of the headers of the the category panels.
	Images (see page 735)	Specifies an imagelist that can be used by the panels to replace the chevron. images.
R	Panels (see page 735)	Lists the panels (TCategoryPanel (see page 309) objects) in the category panel group.

3.1.19.6.1 TCustomCategoryPanelGroup.ChevronAlignment Property

Determines how the chevron is aligned within the header of the category panels in this category panel group.

Pascal

```
property ChevronAlignment: TAlignment;
```

C++

```
__property TAlignment ChevronAlignment;
```

Description

Use the ChevronAlignment property to control the position of the chevron of the category panels within the category panel group. These are the possible values:

Value	Meaning
taLeftJustify	Align Chevron on the left side of the panel.
taCenter	Center the chevron on the panel.
taRightJustify	Align chevron on the right side of the panel.

3.1.19.6.2 TCustomCategoryPanelGroup.ChevronColor Property

Specifies the color of the default chevron in the normal state.

Pascal

```
property ChevronColor: TColor;
```

C++

```
__property TColor ChevronColor;
```

Description

Use the ChevronColor property to select the color of the default normal chevron of the category panels in the category panel group. This property has no effect if the category panel overrides the default normal chevron with the CollapsedImageIndex property or the ExpandedImageIndex property.

See Also

THeaderState (see page 701)

CollapsedImageIndex (see page 715)

ExpandedImageIndex (see page 716)

3.1.19.6.3 TCustomCategoryPanelGroup.ChevronHotColor Property

Specifies the color of the default chevron in the hot state.

Pascal

```
property ChevronHotColor: TColor;
```

C++

```
__property TColor ChevronHotColor;
```

Description

Use the ChevronHotColor property to select the color of the default hot chevron of the category panels in the category panel group. This property has no effect if the category panel overrides the default hot chevron color with the CollpasedHotImageIndex property or the ExpandedHotImageIndex property.

See Also

THeaderState (see page 701)

CollapsedImageIndex (see page 715)

ExpandedImageIndex (see page 716)

3.1.19.6.4 TCustomCategoryPanelGroup.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.19.6.5 TCustomCategoryPanelGroup.GradientBaseColor Property

Specifies the left or top color of the panel header in gradient style.

Pascal

```
property GradientBaseColor: TColor;
```

C++

```
__property TColor GradientBaseColor;
```

Description

Use the GradientBaseColor property to select the base gradient color for the panel header when HeaderStyle (see page 734) is set to hsGradient. If GradientDirection (see page 733) is set to gdHorizontal, GradientBaseColor represents the left color of the header. If GradientDirection (see page 733) is set to gdVertical, GradientBaseColor represents the top color of the header. This property has no effect if HeaderStyle (see page 734) is set to something other than hsGradient

See Also

HeaderStyle (see page 734)

3.1.19.6.6 TCustomCategoryPanelGroup.GradientColor Property

Specifies the right or bottom color of the panel header in gradient style.

Pascal

```
property GradientColor: TColor;
```

C++

```
__property TColor GradientColor;
```

Description

Use the GradientColor property to select the gradient color for the panel header when HeaderStyle (see page 734) is set to hsGradient. If GradientDirection (see page 733) is set to gdHorizontal, GradientColor represents the right color of the header. If GradientDirection (see page 733) is set to gdVertical, GradientColor represents the bottom color of the header. This property has no effect if HeaderStyle (see page 734) is set to something other than hsGradient.

See Also

HeaderStyle (see page 734)

3.1.19.6.7 TCustomCategoryPanelGroup.GradientDirection Property

Specifies the direction of the gradient in the panel header.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

Use the GradientDirection property to select the direction of the gradient header of the category panels in the category panel group. The direction can either be horizontal or vertical. This property has no effect if HeaderStyle (see page 734) is set to something other than hsGradient.

See Also

HeaderStyle (see page 734)

3.1.19.6.8 TCustomCategoryPanelGroup.HeaderAlignment Property

Determines how the caption is aligned within the headers of the category panels in this category panel group.

Pascal

```
property HeaderAlignment: TAlignment;
```

C++

```
__property TAlignment HeaderAlignment;
```

Description

Use the HeaderAlignment property to control the position of the caption of the category panels within the category panel group. These are the possible values:

Value	Meaning
taLeftJustify	Align caption on the left side of the header.
taCenter	Center the caption in the header.
taRightJustify	Align caption on the right side of the header.

3.1.19.6.9 TCustomCategoryPanelGroup.HeaderFont Property

Specifies the font of the caption of the headers of the panels.

Pascal

```
property HeaderFont: TFont;
```

C++

```
__property TFont HeaderFont;
```

Description

Use the HeaderFont property to specify the font and font properties used to draw the caption of the headers of the category panels in the category panel group. The HeaderFont property has no effect when HeaderStyle (see page 734) is hsThemed.

3.1.19.6.10 TCustomCategoryPanelGroup.HeaderHeight Property

Specifies the height of the headers of the panels.

Pascal

```
property HeaderHeight: Integer;
```

C++

```
__property int HeaderHeight;
```

Description

Use the HeaderHeight property to specify the height of the headers of the category panels in the category panel group.

3.1.19.6.11 TCustomCategoryPanelGroup.HeaderImage Property

Specifies the image used for the background of the headers of the panels.

Pascal

```
property HeaderImage: TPicture;
```

C++

```
__property TPicture HeaderImage;
```

Description

Use the HeaderImage property to specify an image as the background for the headers of the category panels within the category panel group. This property has no effect if HeaderStyle (see page 734) is set to something other than hsImage.

See Also

HeaderStyle (see page 734)

3.1.19.6.12 TCustomCategoryPanelGroup.HeaderStyle Property

Specifies the appearance of the headers of the the category panels.

Pascal

```
property HeaderStyle: THeaderStyle;
```

C++

```
__property THeaderStyle HeaderStyle;
```

Description

Use HeaderStyle to determine the appearance of headers of the category panels in the category panel group.

Value	Meaning
hsGradient	Header is a gradient determined by the GradientBaseColor (see page 732), GradientColor (see page 732), and GradientDirection (see page 733) properties.
hsImage	Header is the stretched image defined by the HeaderImage (see page 734) property. If HeaderImage (see page 734) is undefined, the hsGradient style will be used instead.
hsThemed	Header is defined by the system's theme. If runtime themes aren't enabled or supported, the hsGradient style will be used instead.

3.1.19.6.13 TCustomCategoryPanelGroup.Images Property

Specifies an imagelist that can be used by the panels to replace the chevron. images.

Pascal

```
property Images: TImageList;
```

C++

```
__property TImageList Images;
```

Description

Use the images property to define a set of pictures that can be use by the category panels within the category panel group to replace the chevron.

See Also

CollapsedHotImageIndex (see page 714)

CollapsedImageIndex (see page 715)

CollapsedPressedImageIndex (see page 715)

ExpandedHotImageIndex (see page 716)

ExpandedImageIndex (see page 716)

ExpandedPressedImageIndex (see page 716)

3.1.19.6.14 TCustomCategoryPanelGroup.Panels Property

Lists the panels (TCategoryPanel (see page 309) objects) in the category panel group.

Pascal

```
property Panels: TList;
```

C++

```
__property TList Panels;
```

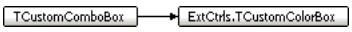
Description

Use Panels to access the list of category panels within the category panel group.

3.1.20 TCustomColorBox Class

TCustomColorBox represents a combo box that lets users select a color.

Class Hierarchy



Pascal

```
TCustomColorBox = class(TCustomComboBox);
```

C++

```
class TCustomColorBox : public TCustomComboBox;
```

File

ExtCtrls

Description

TCustomColorBox is the base class for drop-down combo boxes that display a list of colors. The combo box displays a list of all the colors for which the Graphics (see page 2619) unit defines a constant. For a list of these colors, see TColor (see page 2878). Each color is displayed next to a small square that is drawn in the corresponding color.

Do not use TCustomColorBox in applications. It is intended to act as a base class for color selection controls. Rather, to add a drop-down combo box that lets users select a color, use the TColorBox (see page 466) component, which appears on the Additional page of the component palette.

See Also

TComboBox

TColorDialog

TColorBox (see page 466)

3.1.20.1 TCustomColorBox Members

The following tables list the members exposed by TCustomColorBox.

Protected Methods

	Name	Description
	CloseUp (see page 737)	Generates an OnCloseUp event.
	CreateWnd (see page 738)	Creates the window used to implement the color box.
	DrawItem (see page 738)	Renders an individual item in the color box.
	KeyDown (see page 738)	Generates an OnKeyDown event.
	KeyPress (see page 739)	Generates an OnKeyPress event.
	Loaded (see page 739)	Initializes the combo box after its form is loaded into memory.
	PickCustomColor (see page 740)	Displays the color selection dialog so that the user can select a custom color.
	PopulateList (see page 740)	Fills the color box with items that reflect the colors specified by the Style (see page 744) property.
	Select (see page 740)	Generates an OnSelect event.
	SetStyle (see page 741)	Sets the Style (see page 744) property.

Public Events

	Name	Description
	OnGetColors (see page 744)	Occurs when the control is populated with colors.

Public Methods

	Name	Description
	Create (see page 737)	Creates a new TCustomColorBox (see page 735) object.

Public Properties

	Name	Description
	ColorNames (see page 741)	Lists the names of the colors in the color box.
	Colors (see page 742)	Lists the colors in the color box.
	DefaultColorColor (see page 742)	Specifies the color displayed in the color box for clDefault (see page 2903).
	NoneColorColor (see page 743)	Specifies the color displayed in the color box for clNone (see page 2914).
	Selected (see page 743)	Specifies the currently selected color.
	Style (see page 744)	Controls what colors are listed and how their names are formatted.

3.1.20.2 TCustomColorBox.Create Constructor

Creates a new TCustomColorBox (see page 735) object.

Pascal

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

C++

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

Description

Call Create to instantiate a color box at runtime. TCustomColorBox (see page 735) objects placed on a form at design time are created automatically.

AOwner specifies the owner of the color box. This component (typically the form) is responsible for freeing the color box.

See Also

Owner

3.1.20.3 TCustomColorBox Methods

The methods of the TCustomColorBox class are listed here.

Protected Methods

	Name	Description
	CloseUp (see page 737)	Generates an OnCloseUp event.
	CreateWnd (see page 738)	Creates the window used to implement the color box.
	DrawItem (see page 738)	Renders an individual item in the color box.
	KeyDown (see page 738)	Generates an OnKeyDown event.
	KeyPress (see page 739)	Generates an OnKeyPress event.
	Loaded (see page 739)	Initializes the combo box after its form is loaded into memory.
	PickCustomColor (see page 740)	Displays the color selection dialog so that the user can select a custom color.
	PopulateList (see page 740)	Fills the color box with items that reflect the colors specified by the Style (see page 744) property.
	Select (see page 740)	Generates an OnSelect event.
	SetStyle (see page 741)	Sets the Style (see page 744) property.

3.1.20.3.1 TCustomColorBox.CloseUp Method

Generates an OnCloseUp event.

Pascal

```
procedure CloseUp; override;
```

C++

```
virtual __fastcall CloseUp();
```

Description

Applications can't call this protected method. It is called automatically when the drop-down list of the combo box closes.

CloseUp generates an OnCloseUp event so that applications can respond when the list closes.

See Also

OnCloseUp

3.1.20.3.2 TCustomColorBox.CreateWnd Method

Creates the window used to implement the color box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

Applications can't call this protected method. It is called automatically when the color box is first created or when the window must be destroyed and recreated to reflect property changes.

TCustomColorBox (see page 735) overrides the inherited CreateWnd method to populate the combo box with the colors specified by the Style (see page 744) property.

See Also

CreateWnd

Style (see page 744)

3.1.20.3.3 TCustomColorBox.DrawItem Method

Renders an individual item in the color box.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

Applications can't call this protected method. It is called automatically when the color box needs to paint one of the items in the list. DrawItem draws a small box that displays the color that the item represents, followed by the name of the color.

Index is the index of the item to draw, where 0 specifies the first item, 1 specifies the second item, and so on.

Rect indicates the boundaries of the item on the color box's canvas, in client coordinates.

State indicates state information that can affect the way the item is drawn. In TCustomColorBox (see page 735), DrawItem uses State to determine whether an item is selected, but ignores all other state information.

See Also

Items

3.1.20.3.4 TCustomColorBox.KeyDown Method

Generates an OnKeyDown event.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

Applications can't call this protected method. It is called automatically when the user first presses a key while the color box has focus.

Key is the key code of the key that was pressed.

Shift indicates the state of the Ctrl, Shift, and Alt keys when Key was pressed.

TCustomColorBox (see page 735) overrides the inherited KeyDown method to prepare for the selection of a new color. It then calls the inherited method, which generates an OnKeyDown event.

See Also

OnKeyDown

3.1.20.3.5 TCustomColorBox.KeyPress Method

Generates an OnKeyPress event.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

Applications can't call this protected method. It is called automatically when the user presses a key while the color box has focus.

Key is the key code of the key that was pressed.

TCustomColorBox (see page 735) overrides the inherited KeyPress method to display a color selection dialog if the user presses the Enter key while the custom color item has focus. Otherwise, KeyPress calls the inherited method, which generates an OnKeyPress event.

See Also

OnKeyPress

TColorDialog

3.1.20.3.6 TCustomColorBox.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form

from a stream.

TCustomCombo overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded

ItemIndex

3.1.20.3.7 TCustomColorBox.PickCustomColor Method

Displays the color selection dialog so that the user can select a custom color.

Pascal

```
function PickCustomColor: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PickCustomColor();
```

Description

Applications can't call this protected method. It is called automatically when the user selects the custom color item. PickCustomColor displays the color selection dialog, with the selected color initialized to the last-selected custom color. If the user selects a color in this dialog, PickCustomColor updates the image of the item to display the selected color and returns true. If the user cancels from the color selection dialog, PickCustomColor returns false.

Override PickCustomColor to provide a different response when the user selects the custom color item.

See Also

TColorDialog

3.1.20.3.8 TCustomColorBox.PopulateList Method

Fills the color box with items that reflect the colors specified by the Style (see page 744) property.

Pascal

```
procedure PopulateList;
```

C++

```
__fastcall PopulateList();
```

Description

Applications can't call this protected method. It is called automatically when the color box needs to fill itself with items. (For example, when the color box is first created or when the Style (see page 744) property changes)

PopulateList fills the color box with the items indicated by the Style (see page 744) property.

See Also

Style (see page 744)

3.1.20.3.9 TCustomColorBox.Select Method

Generates an OnSelect event.

Pascal

```
procedure Select; override;
```

C++

```
virtual __fastcall Select();
```

Description

Applications can't call this protected method. It is called automatically when the user selects a color.

TCustomColorBox (see page 735) overrides the inherited method to display a color selection dialog if the user selects the custom color item. After checking for a custom color, TCustomColorBox (see page 735) calls the inherited method, which generates an OnSelect event.

See Also

OnSelect

3.1.20.3.10 TCustomColorBox.SetStyle Method

Sets the Style (see page 744) property.

Pascal

```
procedure SetStyle(AStyle: TColorBoxStyle); reintroduce;
```

C++

```
__fastcall SetStyle(TColorBoxStyle AStyle);
```

Description

SetStyle is the protected write implementation of the Style (see page 744) property. It repopulates the list to reflect the new value of Style (see page 744), and disables the color box if the new value for Style (see page 744) results in an empty color box.

See Also

Style (see page 744)

3.1.20.4 TCustomColorBox Properties

The properties of the TCustomColorBox class are listed here.

Public Properties

	Name	Description
	ColorNames (see page 741)	Lists the names of the colors in the color box.
	Colors (see page 742)	Lists the colors in the color box.
	DefaultColorColor (see page 742)	Specifies the color displayed in the color box for clDefault (see page 2903).
	NoneColorColor (see page 743)	Specifies the color displayed in the color box for clNone (see page 2914).
	Selected (see page 743)	Specifies the currently selected color.
	Style (see page 744)	Controls what colors are listed and how their names are formatted.

3.1.20.4.1 TCustomColorBox.ColorNames Property

Lists the names of the colors in the color box.

Pascal

```
property ColorNames [Index: Integer]: string;
```

C++

```
__property UnicodeString ColorNames[int Index];
```

Description

Use ColorNames to obtain the name of one of the colors listed in the color box.

Index is the index of the name, where 0 is the index of the first color in the box, 1 is the index of the second color, and so on.

The value of each name is either the name of a color constant defined in the Graphics (see page 2619) unit (such as clBlack (see page 2899)), or it is the name of the color with the "cl" stripped off (Black). The Style (see page 744) property determines which name is used for each color.

The Style (see page 744) property also controls which colors are included in the list.

See Also

Colors (see page 742)

Items

ItemIndex

Style (see page 744)

3.1.20.4.2 TCustomColorBox.Colors Property

Lists the colors in the color box.

Pascal

```
property Colors [Index: Integer]: TColor;
```

C++

```
__property TColor Colors[int Index];
```

Description

Use Colors to obtain the value of a color in the color box.

Index is the index of the color, where 0 is the index of the first color in the box, 1 is the index of the second color, and so on.

Note: The Style (see page 744) property controls which colors are included in the list.

See Also

ColorNames (see page 741)

Selected (see page 743)

DefaultColorColor (see page 742)

NoneColorColor (see page 743)

Style (see page 744)

3.1.20.4.3 TCustomColorBox.DefaultColorColor Property

Specifies the color displayed in the color box for clDefault (see page 2903).

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

Set DefaultColorColor to indicate what color the color box should use when drawing the icon for clDefault (see page 2903). This property only affects the color box if the Style (see page 744) property includes cbSystemColors and cbIncludeDefault.

Note: DefaultColorColor only affects the way clDefault (see page 2903) appears in the color box. When you assign clDefault (see page 2903) to a control, the control determines its own default color.

See Also

Colors (see page 742)

NoneColorColor (see page 743)

Style (see page 744)

3.1.20.4.4 TCustomColorBox.NoneColorColor Property

Specifies the color displayed in the color box for clNone (see page 2914).

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

Set NoneColorColor to indicate what color the color box should use when drawing the icon for clNone (see page 2914). This property only affects the color box if the Style (see page 744) property includes cbSystemColors and cbIncludeNone.

Note: NoneColorColor only affects the way clNone (see page 2914) appears in the color box. When you assign clNone (see page 2914) to a control, the control determines how to interpret clNone (see page 2914).

See Also

Colors (see page 742)

DefaultColorColor (see page 742)

Style (see page 744)

3.1.20.4.5 TCustomColorBox.Selected Property

Specifies the currently selected color.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to get or set the selected color as a TColor (see page 2878) value.

See Also

Colors (see page 742)

ItemIndex

3.1.20.4.6 TCustomColorBox.Style Property

Controls what colors are listed and how their names are formatted.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to configure what colors appear in the list and how the names appear. Style is a set that includes 0 or more of the TColorBoxStyles (see page 1541) flags.

See Also

DefaultColorColor (see page 742)

NoneColorColor (see page 743)

ColorNames (see page 741)

Colors (see page 742)

TColor (see page 2878)

TColorDialog

3.1.20.5 TCustomColorBox Events

The events of the TCustomColorBox class are listed here.

Public Events

	Name	Description
	OnGetColors (see page 744)	Occurs when the control is populated with colors.

3.1.20.5.1 TCustomColorBox.OnGetColors Event

Occurs when the control is populated with colors.

Pascal

```
property OnGetColors: TGetColorsEvent;
```

C++

```
__property TGetColorsEvent OnGetColors;
```

3.1.21 TCustomColorListBox Class

Class Hierarchy

Pascal

```
TCustomColorListBox = class(TCustomListBox);
```

C++

```
class TCustomColorListBox : public TCustomListBox;
```

File

ExtCtrls

Description

This is class ExtCtrls.TCustomColorListBox.

3.1.21.1 TCustomColorListBox Members

The following tables list the members exposed by TCustomColorListBox.

Protected Methods

	Name	Description
	CreateWnd (see page 746)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 747)	Generates an OnDrawItem event.
	KeyDown (see page 747)	Respond to key press events.
	KeyPress (see page 748)	Responds when the user presses a key.
	Loaded (see page 748)	Initializes the control after it is loaded from a stream.
	PickCustomColor (see page 749)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 749)	This is PopulateList, a member of class TCustomColorListBox.
	SetStyle (see page 749)	This is SetStyle, a member of class TCustomColorListBox.

Public Events

	Name	Description
	OnGetColors (see page 753)	This is OnGetColors, a member of class TCustomColorListBox.

Public Methods

	Name	Description
	Create (see page 746)	Instantiates a list box control.

Public Properties

	Name	Description
	ColorNames (see page 750)	This is ColorNames, a member of class TCustomColorListBox.
	Colors (see page 750)	This is Colors, a member of class TCustomColorListBox.
	DefaultColorColor (see page 750)	This is DefaultColorColor, a member of class TCustomColorListBox.
	NoneColorColor (see page 750)	This is NoneColorColor, a member of class TCustomColorListBox.
	Selected (see page 751)	Indicates whether a particular item is selected.
	Style (see page 753)	Determines whether the list box is standard or owner-draw and whether it is virtual.

3.1.21.2 TCustomColorListBox.Create Constructor

Instantiates a list box control.

Pascal

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

C++

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

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner

3.1.21.3 TCustomColorListBox Methods

The methods of the TCustomColorListBox class are listed here.

Protected Methods

	Name	Description
	CreateWnd (see page 746)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 747)	Generates an OnDrawItem event.
	KeyDown (see page 747)	Respond to key press events.
	KeyPress (see page 748)	Responds when the user presses a key.
	Loaded (see page 748)	Initializes the control after it is loaded from a stream.
	PickCustomColor (see page 749)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 749)	This is PopulateList, a member of class TCustomColorListBox.
	SetStyle (see page 749)	This is SetStyle, a member of class TCustomColorListBox.

3.1.21.3.1 TCustomColorListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns

CreateHandle

CreateParams

CreateWindowHandle

CreateWnd

ItemIndex

Items

TabWidth

TopIndex

3.1.21.3.2 TCustomColorListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight

MeasureItem

OnDrawItem

OnMeasureItem

Style

3.1.21.3.3 TCustomColorListBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

3.1.21.3.4 TCustomColorListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress

3.1.21.3.5 TCustomColorListBox.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

3.1.21.3.6 TCustomColorListBox.PickCustomColor Method

Pascal

```
function PickCustomColor: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PickCustomColor();
```

Description

This is PickCustomColor, a member of class TCustomColorListBox.

3.1.21.3.7 TCustomColorListBox.PopulateList Method

Pascal

```
procedure PopulateList;
```

C++

```
__fastcall PopulateList();
```

Description

This is PopulateList, a member of class TCustomColorListBox.

3.1.21.3.8 TCustomColorListBox.SetStyle Method

Pascal

```
procedure SetStyle(AStyle: TColorBoxStyle); reintroduce;
```

C++

```
__fastcall SetStyle(TColorBoxStyle AStyle);
```

Description

This is SetStyle, a member of class TCustomColorListBox.

3.1.21.4 TCustomColorListBox Properties

The properties of the TCustomColorListBox class are listed here.

Public Properties

	Name	Description
	ColorNames (see page 750)	This is ColorNames, a member of class TCustomColorListBox.
	Colors (see page 750)	This is Colors, a member of class TCustomColorListBox.
	DefaultColorColor (see page 750)	This is DefaultColorColor, a member of class TCustomColorListBox.
	NoneColorColor (see page 750)	This is NoneColorColor, a member of class TCustomColorListBox.
	Selected (see page 751)	Indicates whether a particular item is selected.
	Style (see page 753)	Determines whether the list box is standard or owner-draw and whether it is virtual.

3.1.21.4.1 TCustomColorListBox.ColorNames Property

Pascal

```
property ColorNames [Index: Integer]: string;
```

C++

```
__property UnicodeString ColorNames[int Index];
```

Description

This is ColorNames, a member of class TCustomColorListBox.

3.1.21.4.2 TCustomColorListBox.Colors Property

Pascal

```
property Colors [Index: Integer]: TColor;
```

C++

```
__property TColor Colors[int Index];
```

Description

This is Colors, a member of class TCustomColorListBox.

3.1.21.4.3 TCustomColorListBox.DefaultColorColor Property

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

This is DefaultColorColor, a member of class TCustomColorListBox.

3.1.21.4.4 TCustomColorListBox.NoneColorColor Property

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

This is NoneColorColor, a member of class TCustomColorListBox.

3.1.21.4.5 TCustomColorListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").t_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).t_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
```

```
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
        fclose(F);
      }
    }
  }
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
```

See Also

extendedselect

Items

MultiSelect

SelCount

3.1.21.4.6 TCustomColorListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem

ItemHeight

Items

MeasureItem

OnDrawItem

OnMeasureItem

OnData

OnDataFind

OnDataObject

3.1.21.5 TCustomColorListBox Events

The events of the TCustomColorListBox class are listed here.

Public Events

	Name	Description
	OnGetColors (see page 753)	This is OnGetColors, a member of class TCustomColorListBox.

3.1.21.5.1 TCustomColorListBox.OnGetColors Event

Pascal

```
property OnGetColors: TLBGetColorsEvent;
```

C++

```
__property TLBGetColorsEvent OnGetColors;
```

Description

This is OnGetColors, a member of class TCustomColorListBox.

3.1.22 TCustomControlBar Class

TCustomControlBar is the base class for control bar components.

Class Hierarchy

Pascal

```
TCustomControlBar = class(TCustomControl);
```

C++

```
class TCustomControlBar : public TCustomControl;
```

File

ExtCtrls

Description

Use TCustomControlBar as a base class when defining control bar components. Control bars are designed to manage the layout of toolbar controls, although they can contain other controls as well. They provide a convenient docking site in applications that have dockable toolbars.

Do not create instances of TCustomControlBar. To add a control bar to a form, use TControlBar (see page 590) instead.

See Also

TControlBar (see page 590)

TToolBar

3.1.22.1 TCustomControlBar Members

The following tables list the members exposed by TCustomControlBar.

Protected Events

	Name	Description
	OnBandDrag (see page 771)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 771)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 772)	Occurs when a control band is moving.
	OnBandPaint (see page 772)	Occurs when a control band is painted.
	OnBeginBandMove (see page 773)	This is OnBeginBandMove, a member of class TCustomControlBar.
	OnEndBandMove (see page 773)	This is OnEndBandMove, a member of class TCustomControlBar.
	OnPaint (see page 773)	Occurs when the control bar is painted.

Protected Methods

	Name	Description
	AlignControls (see page 757)	Aligns controls in the control bar.
	CanAutoSize (see page 757)	Indicates how the control can be resized.
	CreateParams (see page 758)	Sets the control bar's window flags.
	DoBandMove (see page 758)	Generates an OnBandMove (see page 772) event.
	DoBandPaint (see page 758)	Generates an OnBandPaint (see page 772) event.
	DoBeginBandMove (see page 759)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 759)	Generates an OnDockOver event and ensures proper docking of floating controls.

	DoEndBandMove (see page 759)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoPaletteChange (see page 760)	Prepares the control bar to realize a new palette.
V	DragControl (see page 760)	Starts a drag operation on a control band.
V	GetControlInfo (see page 761)	Generates an OnBandInfo (see page 771) event.
V	GetPalette (see page 762)	Returns the handle to a palette for use by the control.
V	GetSiteInfo (see page 762)	Generates an OnGetSiteInfo event
	HitTest (see page 763)	Returns the control at the specified coordinates.
V	MouseDown (see page 763)	Handles mouse button clicks.
V	MouseMove (see page 763)	Handles mouse movements.
V	MouseUp (see page 764)	Handles releasing of a mouse button.
V	Paint (see page 764)	Paints the control bar.
V	PaintControlFrame (see page 765)	Paints a control band's frame.

Protected Properties

	Name	Description
	AutoDock (see page 766)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 766)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 767)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelKind (see page 767)	Specifies the control's bevel style.
	CornerEdge (see page 768)	This is CornerEdge, a member of class TCustomControlBar.
	DockSite (see page 768)	Specifies whether the control can be the target of drag-and-dock operations.
	DrawingStyle (see page 769)	This is DrawingStyle, a member of class TCustomControlBar.
	GradientDirection (see page 769)	This is GradientDirection, a member of class TCustomControlBar.
	GradientEndColor (see page 769)	This is GradientEndColor, a member of class TCustomControlBar.
	GradientStartColor (see page 769)	This is GradientStartColor, a member of class TCustomControlBar.
	RowSize (see page 770)	Specifies the height of the control bar's rows.
	RowSnap (see page 770)	Specifies whether controls are snapped to the control bar's rows when docked.

Public Methods

	Name	Description
V	Create (see page 755)	Creates and initializes an instance of TCustomControlBar (see page 754).
V	Destroy (see page 756)	Destroys the TCustomControlBar (see page 754) instance.
V	FlipChildren (see page 761)	Reverses the positions of child controls.
V	StickControls (see page 765)	Places controls in their final positions after a control is moved in or dropped on the control bar.

Public Properties

	Name	Description
	Picture (see page 770)	Specifies the control bar's picture.

3.1.22.2 TCustomControlBar.Create Constructor

Creates and initializes an instance of TCustomControlBar (see page 754).

Pascal

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

C++

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

Description

Call Create to instantiate a control bar at runtime. Control bars added to forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the control bar instance. It becomes the value of the Owner property.

See Also

Owner

3.1.22.3 TCustomControlBar.Destroy Destructor

Destroys the TCustomControlBar (see page 754) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomControlBar();
```

Description

Do not call Destroy. Instead, call Free.

Destroy frees the memory allocated for the TCustomControlBar (see page 754) properties.

See Also

System::TObject::Free

3.1.22.4 TCustomControlBar Methods

The methods of the TCustomControlBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 757)	Aligns controls in the control bar.
	CanAutoSize (see page 757)	Indicates how the control can be resized.
	CreateParams (see page 758)	Sets the control bar's window flags.
	DoBandMove (see page 758)	Generates an OnBandMove (see page 772) event.
	DoBandPaint (see page 758)	Generates an OnBandPaint (see page 772) event.
	DoBeginBandMove (see page 759)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 759)	Generates an OnDockOver event and ensures proper docking of floating controls.
	DoEndBandMove (see page 759)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoPaletteChange (see page 760)	Prepares the control bar to realize a new palette.
	DragControl (see page 760)	Starts a drag operation on a control band.
	GetControlInfo (see page 761)	Generates an OnBandInfo (see page 771) event.
	GetPalette (see page 762)	Returns the handle to a palette for use by the control.
	GetSiteInfo (see page 762)	Generates an OnGetSiteInfo event
	HitTest (see page 763)	Returns the control at the specified coordinates.
	MouseDown (see page 763)	Handles mouse button clicks.
	MouseMove (see page 763)	Handles mouse movements.
	MouseUp (see page 764)	Handles releasing of a mouse button.
	Paint (see page 764)	Paints the control bar.
	PaintControlFrame (see page 765)	Paints a control band's frame.

Public Methods

	Name	Description
	FlipChildren (see page 761)	Reverses the positions of child controls.

	StickControls (see page 765)	Places controls in their final positions after a control is moved in or dropped on the control bar.

3.1.22.4.1 TCustomControlBar.AlignControls Method

Aligns controls in the control bar.

Pascal

```
procedure AlignControls(AControl: TControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect ARect);
```

Description

Use AlignControls to align all controls within an area of the control bar. The alignment of child controls is based on their Align property.

AControl is a child control that takes precedence in alignment over other, similarly-aligned controls. This parameter can be nil (Delphi) or NULL (C++).

Rect specifies the area in which to align the controls.

TCustomControlBar (see page 754) overrides the inherited method to enforce the alignment of child controls in a manner appropriate to a control bar. AlignControls determines the best alignment for each control according to its size and the available space.

3.1.22.4.2 TCustomControlBar.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

3.1.22.4.3 TCustomControlBar.CreateParams Method

Sets the control bar's window flags.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams fills in the parameter record that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

In TCustomControlBar (see page 754), CreateParams overrides the inherited method in order to turn off the CS_HREDRAW and CS_VREDRAW windows flags.

See Also

CreateWnd

CreateWindowHandle

3.1.22.4.4 TCustomControlBar.DoBandMove Method

Generates an OnBandMove (see page 772) event.

Pascal

```
procedure DoBandMove(Control: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall DoBandMove(TControl * Control, TRect ARect);
```

Description

Do not call DoBandMove. It is called automatically when a band is moved in the control bar.

Control is the band that is about to be moved.

ARect is the area to which the control is about to be moved. This value can be changed by the OnBandMove (see page 772) event handler.

Override DoBandMove to perform actions in addition to calling the OnBandMove (see page 772) event handler when a band is moved.

See Also

onbandmove (see page 772)

3.1.22.4.5 TCustomControlBar.DoBandPaint Method

Generates an OnBandPaint (see page 772) event.

Pascal

```
procedure DoBandPaint(Control: TControl; Canvas: TCanvas; var ARect: TRect; var Options:
TBandPaintOptions); virtual;
```

C++

```
virtual __fastcall DoBandPaint(TControl * Control, TCanvas Canvas, TRect ARect,
TBandPaintOptions Options);
```

Description

Do not call DoBandPaint. It is called automatically for each band when the control bar is painted. Override DoBandPaint to provide class-specific paint code for the bands or to suppress the OnBandPaint (see page 772) event.

Control is the control (usually a tool bar) whose band is about to be painted.

Canvas is the control bar's canvas. This is the canvas to use for drawing the band.

ARect is the dimensions of the band on the canvas. The OnBandPaint (see page 772) event handler can change this value to alter where the frame and grabber are drawn.

Options specifies which parts of the band should be drawn after DoBandPaint exits.

See Also

OnBandPaint (see page 772)

3.1.22.4.6 TCustomControlBar.DoBeginBandMove Method

Pascal

```
function DoBeginBandMove(Control: TControl): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoBeginBandMove(TControl * Control);
```

Description

This is DoBeginBandMove, a member of class TCustomControlBar.

3.1.22.4.7 TCustomControlBar.DockOver Method

Generates an OnDockOver event and ensures proper docking of floating controls.

Pascal

```
procedure DockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); override;
```

C++

```
virtual __fastcall DockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean
Accept);
```

Description

DockOver overrides the inherited method to ensure proper docking of floating controls that are accepted by the OnDockOver event handler.

See Also

OnDockOver

3.1.22.4.8 TCustomControlBar.DoEndBandMove Method

Pascal

```
procedure DoEndBandMove(Control: TControl); dynamic;
```

C++

```
__fastcall DoEndBandMove(TControl * Control);
```

Description

This is DoEndBandMove, a member of class TCustomControlBar.

3.1.22.4.9 TCustomControlBar.DoPaletteChange Method

Prepares the control bar to realize a new palette.

Pascal

```
function DoPaletteChange: Boolean;
```

C++

```
__fastcall Boolean DoPaletteChange();
```

Description

Call DoPaletteChange to prepare the control bar to realize a new palette. DoPaletteChange returns true if the control bar should be redrawn to reflect the new palette, and false otherwise.

DoPaletteChange is called when the Picture (see page 770) property changes.

See Also

picture (see page 770)

getpalette (see page 762)

3.1.22.4.10 TCustomControlBar.DragControl Method

Starts a drag operation on a control band.

Pascal

```
function DragControl(AControl: TControl; X: Integer; Y: Integer; KeepCapture: Boolean =
False): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DragControl(TControl * AControl, int X, int Y, Boolean KeepCapture
= False);
```

Description

Call DragControl to start a drag operation on a control band.

If AutoDrag (see page 766) is true, DragControl is called automatically when a control is undocked from the control bar.

AControl is the control on the band that should be dragged.

X and Y are the current positions of the mouse pointer.

KeepCapture indicates whether the control bar should capture the mouse when processing the drag.

See Also

AutoDrag (see page 766)

Capture

3.1.22.4.11 TCustomControlBar.FlipChildren Method

Reverses the positions of child controls.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

Call FlipChildren to flip the control's children; that is, to move children on the left side of the control to the right side and vice versa. FlipChildren reverses the position of all child controls and adjusts their Align properties, if necessary, to enforce the new position.

AllLevels specifies whether FlipChildren should be called recursively on the control's children.

FlipChildren can be used to reverse the layout of an application when it is running in Middle Eastern locales where users read from right to left instead of left to right: Call UseRightToLeftAlignment to determine whether the BiDiMode property dictates a reversal of alignment based on the system locale.

Note: FlipChildren does the same thing as the Flip Children command on the control's context menu in the IDE.

See Also

BiDiMode

UseRightToLeftAlignment

3.1.22.4.12 TCustomControlBar.GetControlInfo Method

Generates an OnBandInfo (see page 771) event.

Pascal

```
procedure GetControlInfo(AControl: TControl; var Insets: TRect; var PreferredSize: Integer;
var RowCount: Integer); virtual;
```

C++

```
virtual __fastcall GetControlInfo(TControl * AControl, TRect Insets, int PreferredSize, int
RowCount);
```

Description

GetControlInfo is called automatically when the control bar needs to arrange the position of its bands. It generates an OnBandInfo (see page 771) event to obtain information about a control band.

AControl is the control associated with the band being positioned.

Insets returns the margin between the edges of the control and the edges of its band.

PreferredSize returns the control's preferred width. (The control's width is reduced if it can't fit inside the control bar.)

RowCount returns the number of control rows that the control spans in the control bar.

See Also

OnBandInfo (see page 771)

3.1.22.4.13 TCustomControlBar.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged

3.1.22.4.14 TCustomControlBar.GetSiteInfo Method

Generates an OnGetSiteInfo event

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

GetSiteInfo is called automatically during drag-and-dock operations to initialize a TDragDockObject with information about where a dragged object can be docked, if at all.

Client is a potential docking client.

InfluenceRect returns the area in which the client would be docked.

MousePos is the current mouse position.

CanDock returns whether the client can dock to the windowed control.

As implemented in TWinControl, GetSiteInfo generates an influence rectangle equal to BoundsRect expanded by 10 pixels on every side, and then calls the OnGetSiteInfo event handler.

See Also

OnGetSiteInfo

BoundsRect

3.1.22.4.15 TCustomControlBar.HitTest Method

Returns the control at the specified coordinates.

Pascal

```
function HitTest(X: Integer; Y: Integer): TControl;
```

C++

```
__fastcall TControl * HitTest(int X, int Y);
```

Description

Call HitTest to get the control at the specified coordinates.

X and Y are the coordinates.

See Also

MouseDown (see page 763)

3.1.22.4.16 TCustomControlBar.MouseDown Method

Handles mouse button clicks.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown is called automatically when the user clicks the control bar with the mouse. After generating an OnMouseDown event, MouseDown determines whether a band is being dragged and if so, computes its position.

Button indicates which button was clicked.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the position where the mouse was clicked.

See Also

OnMouseDown

HitTest (see page 763)

3.1.22.4.17 TCustomControlBar.MouseMove Method

Handles mouse movements.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

MouseMove generates an OnMouseMove event and then performs special processing if a control is being undocked from the

control bar or moved within the control bar.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the current position of the mouse.

See Also

OnMouseMove

AutoDrag (see page 766)

3.1.22.4.18 TCustomControlBar.MouseUp Method

Handles releasing of a mouse button.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp handles the dropping of a control onto the control bar and, once the control (if any) is docked, generates an OnMouseUp event.

Button indicates which button was released.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the position where the mouse was released.

See Also

OnMouseUp

3.1.22.4.19 TCustomControlBar.Paint Method

Paints the control bar.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically when the control bar needs to paint itself.

To customize the painting, either override the Paint method, or specify options in the OnPaint (see page 773) and OnBandPaint (see page 772) event handlers.

See Also

OnPaint (see page 773)

onbandpaint (see page 772)

PaintControlFrame (see page 765)

3.1.22.4.20 TCustomControlBar.PaintControlFrame Method

Paints a control band's frame.

Pascal

```
procedure PaintControlFrame(Canvas: TCanvas; AControl: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall PaintControlFrame(TCanvas Canvas, TControl * AControl, TRect ARect);
```

Description

The Paint (see page 764) method calls PaintControlFrame for each band to render its image. PaintControlFrame generates an OnBandPaint (see page 772) event, and then draws the frame and grabber for the band, depending on the values returned by the event handler.

Canvas is the control bar's canvas.

AControl is the control on the band to be painted.

ARect indicates the dimensions of the band.

See Also

OnPaint (see page 773)

onbandpaint (see page 772)

PaintControlFrame

3.1.22.4.21 TCustomControlBar.StickControls Method

Places controls in their final positions after a control is moved in or dropped on the control bar.

Pascal

```
procedure StickControls; virtual;
```

C++

```
virtual __fastcall StickControls();
```

Description

Override StickControls to specify how controls are set in their final positions after a control is moved in or dropped on the control bar.

See Also

rowsnap (see page 770)

3.1.22.5 TCustomControlBar Properties

The properties of the TCustomControlBar class are listed here.

Protected Properties

	Name	Description
	AutoDock (see page 766)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 766)	Specifies whether control bands can be undocked by dragging.

	AutoSize (see page 767)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelKind (see page 767)	Specifies the control's bevel style.
	CornerEdge (see page 768)	This is CornerEdge, a member of class TCustomControlBar.
	DockSite (see page 768)	Specifies whether the control can be the target of drag-and-dock operations.
	DrawingStyle (see page 769)	This is DrawingStyle, a member of class TCustomControlBar.
	GradientDirection (see page 769)	This is GradientDirection, a member of class TCustomControlBar.
	GradientEndColor (see page 769)	This is GradientEndColor, a member of class TCustomControlBar.
	GradientStartColor (see page 769)	This is GradientStartColor, a member of class TCustomControlBar.
	RowSize (see page 770)	Specifies the height of the control bar's rows.
	RowSnap (see page 770)	Specifies whether controls are snapped to the control bar's rows when docked.

Public Properties

	Name	Description
	Picture (see page 770)	Specifies the control bar's picture.

3.1.22.5.1 TCustomControlBar.AutoDock Property

Determines whether the control bar temporarily docks drag clients to provide visual feedback.

Pascal

```
property AutoDock: Boolean;
```

C++

```
__property Boolean AutoDock;
```

Description

If AutoDock is true (the default), when a dockable object is dragged over the control bar, the control bar temporarily docks the object so that the user can see the impact of the drag operation. That is, the user can see the dock position and how other docked controls will be repositioned to make room. When the user releases the mouse, the dragged object is docked at the current spot, and any temporary dockings are canceled. However, if the user cancels the drag-and-dock operation, the dragged object reverts to its last temporary dock site rather than to its location at the start of the drag operation.

If AutoDock is false, the control bar does not temporarily dock dragged objects. Instead, the user only sees the dragging rectangle as it passes over the control bar, with no feedback as to the adjustments the control bar would make if the dragged object were released.

See Also

DockClients

DockSite

OnDockOver

3.1.22.5.2 TCustomControlBar.AutoDrag Property

Specifies whether control bands can be undocked by dragging.

Pascal

```
property AutoDrag: Boolean;
```

C++

```
__property Boolean AutoDrag;
```

Description

Use AutoDrag to specify whether control bands are automatically undocked when the user drags them. If AutoDrag is true, control bands (TToolBar objects) are undocked when dragged off the control bar. Otherwise, control bands are kept within the control bar.

See Also

DragMode

OnBandDrag (see page 771)

onbandmove (see page 772)

3.1.22.5.3 TCustomControlBar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.22.5.4 TCustomControlBar.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.22.5.5 TCustomControlBar.CornerEdge Property

Pascal

```
property CornerEdge: TCornerEdge;
```

C++

```
__property TCornerEdge CornerEdge;
```

Description

This is CornerEdge, a member of class TCustomControlBar.

3.1.22.5.6 TCustomControlBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
```

```
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.22.5.7 TCustomControlBar.DrawingStyle Property

Pascal

```
property DrawingStyle: TBandDrawingStyle;
```

C++

```
__property TBandDrawingStyle DrawingStyle;
```

Description

This is DrawingStyle, a member of class TCustomControlBar.

3.1.22.5.8 TCustomControlBar.GradientDirection Property

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

This is GradientDirection, a member of class TCustomControlBar.

3.1.22.5.9 TCustomControlBar.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TCustomControlBar.

3.1.22.5.10 TCustomControlBar.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TCustomControlBar.

3.1.22.5.11 TCustomControlBar.Picture Property

Specifies the control bar's picture.

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

Use Picture to specify the control bar's picture. The picture is tiled across the control bar.

See Also

OnBandPaint (see page 772)

OnPaint (see page 773)

3.1.22.5.12 TCustomControlBar.RowSize Property

Specifies the height of the control bar's rows.

Pascal

```
property RowSize: TRowSize;
```

C++

```
__property TRowSize RowSize;
```

Description

Use RowSize to specify the height, in pixels, of the control bar's rows.

See Also

CanAutoSize (see page 757)

AutoSize

OnBandInfo (see page 771)

ClientHeight

3.1.22.5.13 TCustomControlBar.RowSnap Property

Specifies whether controls are snapped to the control bar's rows when docked.

Pascal

```
property RowSnap: Boolean;
```

C++

```
__property Boolean RowSnap;
```

Description

Use RowSnap to specify whether controls are snapped to the control bar's rows when docked. Controls that are snapped have

their Top and Height properties set automatically to correspond to the control bar's rows.

See Also

StickControls (see page 765)

Height

Top

3.1.22.6 TCustomControlBar Events

The events of the TCustomControlBar class are listed here.

Protected Events

	Name	Description
	OnBandDrag (see page 771)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 771)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 772)	Occurs when a control band is moving.
	OnBandPaint (see page 772)	Occurs when a control band is painted.
	OnBeginBandMove (see page 773)	This is OnBeginBandMove, a member of class TCustomControlBar.
	OnEndBandMove (see page 773)	This is OnEndBandMove, a member of class TCustomControlBar.
	OnPaint (see page 773)	Occurs when the control bar is painted.

3.1.22.6.1 TCustomControlBar.OnBandDrag Event

Occurs when the user starts a drag operation on a specified control.

Pascal

```
property OnBandDrag: TBandDragEvent;
```

C++

```
__property TBandDragEvent OnBandDrag;
```

Description

Use OnBandDrag to conditionally allow the user to start dragging a control in the control bar.

The event handler indicates the control that the user is trying to drag. Change its Drag parameter to false to prevent the drag operation. If Drag remains true on exit from the event handler, the drag operation proceeds.

See Also

OnStartDock

onbandmove (see page 772)

3.1.22.6.2 TCustomControlBar.OnBandInfo Event

Occurs when the control bar adjusts the docking position of a band.

Pascal

```
property OnBandInfo: TBandInfoEvent;
```

C++

```
__property TBandInfoEvent OnBandInfo;
```

Description

Write code in the OnBandInfo event handler to provide information to the control bar that it can use when arranging its bands. The OnBandInfo event handler has several parameters that can be used to return information about how the band should be positioned, including its size, inset, and number of rows.

See Also

TBDockHeight

onbandmove (see page 772)

LRDockWidth

Height

Width

3.1.22.6.3 TCustomControlBar.OnBandMove Event

Occurs when a control band is moving.

Pascal

```
property OnBandMove: TBandMoveEvent;
```

C++

```
__property TBandMoveEvent OnBandMove;
```

Description

Write an OnBandMove event handler to perform actions when a band is moving.

Use OnBandMove to specify the destination rectangle for the band. For example, adjust the destination rectangle so that its right edge matches the right edge of the control bar. This means the control extends to the control bar's right edge no matter where it is moved.

See Also

AutoDrag (see page 766)

RowSnap (see page 770)

3.1.22.6.4 TCustomControlBar.OnBandPaint Event

Occurs when a control band is painted.

Pascal

```
property OnBandPaint: TBandPaintEvent;
```

C++

```
__property TBandPaintEvent OnBandPaint;
```

Description

Write an OnBandPaint event handler to customize the way the control bar draws its bands. From the event handler, you can draw to the control bar's canvas, using the dimensions of the band that are passed in as the ARect parameter. You can change the dimensions so that the automatically drawn frame and grabber appear in a different portion of the control bar (for example, indented slightly). You can suppress the automatic painting of the frame or grabber by removing options from the Options parameter.

See Also

OnPaint (see page 773)

3.1.22.6.5 TCustomControlBar.OnBeginBandMove Event

Pascal

```
property OnBeginBandMove: TBeginBandMoveEvent;
```

C++

```
__property TBeginBandMoveEvent OnBeginBandMove;
```

Description

This is OnBeginBandMove, a member of class TCustomControlBar.

3.1.22.6.6 TCustomControlBar.OnEndBandMove Event

Pascal

```
property OnEndBandMove: TEndBandMoveEvent;
```

C++

```
__property TEndBandMoveEvent OnEndBandMove;
```

Description

This is OnEndBandMove, a member of class TCustomControlBar.

3.1.22.6.7 TCustomControlBar.OnPaint Event

Occurs when the control bar is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the control bar is rendered. When a control bar receives a Windows paint message, it first draws the background picture, if any, and then generates an OnPaint event. After the OnPaint event exits, the control bar generates an OnBandPaint (see page 772) event for each band, finally rendering the grabbers and frames for each band.

Use the Canvas property to draw the control bar's image. Use the ClientRect property to obtain the dimensions on which to draw.

If the custom drawing is intended to change the appearance of individual bands, use the OnBandPaint (see page 772) event instead.

See Also

onbandpaint (see page 772)

Canvas

ClientRect

picture (see page 770)

3.1.23 TCustomFlowPanel Class

The base class for all flow panel components.

Class Hierarchy

Pascal

```
TCustomFlowPanel = class(TCustomPanel);
```

C++

```
class TCustomFlowPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TCustomFlowPanel as a base class when defining flow panels that include other controls as constituent parts.

See Also

TFlowPanel (see page 857)

3.1.23.1 TCustomFlowPanel Members

The following tables list the members exposed by TCustomFlowPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AlignControls (see page 776)	Aligns any controls for which the control is the parent within a specified area of the control.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 806)	Renders the image of the panel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 812)	Specifies the background color of the control.
	FullRepaint (see page 813)	Determines how the panel repaints itself when it is resized.
	Locked (see page 813)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 814)	Determines where a control looks for its color information.
	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Methods

	Name	Description
	Create (see page 775)	Creates and initializes an instance of TCustomPanel.
	Destroy (see page 775)	Destroys an instance of TCustomControl.
	GetChildren (see page 776)	Calls a specified method for each child of the control.
	GetControlIndex (see page 777)	This is GetControlIndex, a member of class TCustomFlowPanel.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	SetControlIndex (see page 777)	This is SetControlIndex, a member of class TCustomFlowPanel.

Public Properties

	Name	Description
	AutoWrap (see page 778)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	FlowStyle (see page 778)	Specifies the starting point and direction for controls added to a flow panel.
	ParentBackground (see page 813)	Determines whether control uses parent's theme background.

3.1.23.2 TCustomFlowPanel.Create Constructor

Creates and initializes an instance of TCustomPanel.

Pascal

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

C++

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

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner

3.1.23.3 TCustomFlowPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomFlowPanel();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

3.1.23.4 TCustomFlowPanel Methods

The methods of the TCustomFlowPanel class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 776)	Aligns any controls for which the control is the parent within a specified area of the control.

Public Methods

	Name	Description
	GetChildren (see page 776)	Calls a specified method for each child of the control.
	GetControlIndex (see page 777)	This is GetControlIndex, a member of class TCustomFlowPanel.
	SetControlIndex (see page 777)	This is SetControlIndex, a member of class TCustomFlowPanel.

3.1.23.4.1 TCustomFlowPanel.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

3.1.23.4.2 TCustomFlowPanel.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

3.1.23.4.3 TCustomFlowPanel.GetControlIndex Method

Pascal

```
function GetControlIndex(AControl: TControl): Integer;
```

C++

```
__fastcall int GetControlIndex(TControl * AControl);
```

Description

This is GetControlIndex, a member of class TCustomFlowPanel.

3.1.23.4.4 TCustomFlowPanel.SetControlIndex Method

Pascal

```
procedure SetControlIndex(AControl: TControl; Index: Integer);
```

C++

```
__fastcall SetControlIndex(TControl * AControl, int Index);
```

Description

This is SetControlIndex, a member of class TCustomFlowPanel.

3.1.23.5 TCustomFlowPanel Properties

The properties of the TCustomFlowPanel class are listed here.

Public Properties

	Name	Description
	AutoWrap (see page 778)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	FlowStyle (see page 778)	Specifies the starting point and direction for controls added to a flow panel.

3.1.23.5.1 TCustomFlowPanel.AutoWrap Property

Specifies whether controls are automatically wrapped to fit on the flow panel.

Pascal

```
property AutoWrap: Boolean;
```

C++

```
__property Boolean AutoWrap;
```

Description

If set to True, a new control is automatically placed in a location on the flow panel in which it is visible. If set to False, a new control continues in the same direction as the previous control, even if it would no longer be visible on the panel. You can increase the size of the panel to view controls that are not automatically wrapped to the next line or column.

3.1.23.5.2 TCustomFlowPanel.FlowStyle Property

Specifies the starting point and direction for controls added to a flow panel.

Pascal

```
property FlowStyle: TFlowStyle;
```

C++

```
__property TFlowStyle FlowStyle;
```

Description

Use FlowStyle to set the starting position for the first control added to a flow panel, as well as the subsequent position for additional controls.

Flow Style	Description
BottomTopLeftRight	The first control is at the bottom left corner of the panel. Subsequent controls are added from bottom to top and then from left to right.
BottomTopRightLeft	The first control is at the bottom right corner of the panel. Subsequent controls are added from bottom to top and then from right to left.
LeftRightBottomTop	The first control is at the bottom left corner of the panel. Subsequent controls are added from left to right and then from bottom to top.
LeftRightTopBottom	The first control is at the top left corner of the panel. Subsequent controls are added from left to right and then from top to bottom. This is the default setting.
RightLeftBottomTop	The first control is at the bottom right corner of the panel. Subsequent controls are added from right to left and then from bottom to top.
RightLeftTopBottom	The first control is at the top right corner of the panel. Subsequent controls are added from right to left and then from top to bottom.
TopBottomLeftRight	The first control is at the top left corner of the panel. Subsequent controls are added from top to bottom and then from left to right.
TopBottomRightLeft	The first control is at the top right corner of the panel. Subsequent controls are added from top to bottom and then from right to left.

C++ Examples:

```
//
//This example demonstrates the use of Flow panel to automatically
//align a set of controls and the different methods that can be used for that.
```

```
//A Stack is used to control the order of destruction of buttons.
//
void __fastcall TForm2::btAButtonClick(TObject *Sender)
{
    TButton *thisButton;
    TButton *lastButton;
    String temp;

    /* Sender is a TButton, cast it */
    thisButton = dynamic_cast<TButton*>(Sender);

    if (m_Stack->Count() > 0)
    {
        /* If there are buttons in stack, peek it */
        lastButton = dynamic_cast<TButton*>(m_Stack->Peek());

        /* Change the captions of this button and the last one */
        temp = thisButton->Caption;
        thisButton->Caption = lastButton->Caption;
        lastButton->Caption = temp;
    }
}

void __fastcall TForm2::btAddButtonClick(TObject *Sender)
{
    /* Create a new button */
    TButton *button = new TButton(m_FlowPane);

    button->Parent = m_FlowPane;
    button->Caption = IntToStr(m_Stack->Count());
    button->OnClick = btAButtonClick;

    /* Push the button to the stack */
    m_Stack->Push(button);
}

void __fastcall TForm2::btReArrangeClick(TObject *Sender)
{
    /* Change the current flow style (rotate them) */
    TFlowStyle newFlowStyle = m_FlowPane->FlowStyle;

    if (newFlowStyle == fsBottomTopRightLeft)
        newFlowStyle = fsLeftRightTopBottom;
    else
        newFlowStyle = newFlowStyle + 1;

    m_FlowPane->FlowStyle = newFlowStyle;
}

void __fastcall TForm2::btRemoveButtonClick(TObject *Sender)
{
    TButton *button;

    if (m_Stack->Count() > 0)
    {
        /* if the stack is not empty, pop the last button */
        button = dynamic_cast<TButton*>(m_Stack->Pop());
        delete button;
    }
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Create a new TFlowPanel pane and align it to client */
    m_FlowPane = new TFlowPanel(this);
    m_FlowPane->Parent = this;
```

```
    m_FlowPane->Align = alClient;

    /* Set the initial flowing style */
    m_FlowPane->FlowStyle = fsLeftRightTopBottom;

    /* Create the button stack */
    m_Stack = new TObjectStack();
}

void __fastcall TForm2::FormDestroy(TObject *Sender)
{
    /* Destroy the stack also */
    delete m_Stack;
}
```

Delphi Examples:

```
{
This example demonstrates the use of Flow panel to automatically
align a set of controls and the different methods that can be used for that.
A Stack is used to control the order of destruction of buttons.
}
procedure TForm2.btAButtonClick(Sender: TObject);
var
  ThisButton : TButton;
  LastButton : TButton;
  Temp       : String;
begin
  { Sender is a TButton, cast it }
  ThisButton := Sender as TButton;

  if FStack.Count > 0 then
  begin
    { If there are buttons in stack, peek it }
    LastButton := FStack.Peek() as TButton;

    { Change the captions of this button and the last one }
    Temp := ThisButton.Caption;
    ThisButton.Caption := LastButton.Caption;
    LastButton.Caption := Temp;
  end;
end;

procedure TForm2.btAddButtonClick(Sender: TObject);
var
  Button : TButton;
begin
  { Create a new button }
  Button := TButton.Create(FFlowPane);
  Button.Parent := FFlowPane;
  Button.Caption := IntToStr(FStack.Count);
  Button.OnClick := btAButtonClick;

  { Push the button to the stack }
  FStack.Push(Button);
end;

procedure TForm2.btReArrangeClick(Sender: TObject);
var
  NewFlowStyle : TFlowStyle;
begin
  { Chnage the flow order (rotate) }
  NewFlowStyle := FFlowPane.FlowStyle;

  if NewFlowStyle = fsBottomTopRightLeft then
     NewFlowStyle := fsLeftRightTopBottom
  else
```

```
    Inc(NewFlowStyle);

  FFlowPane.FlowStyle := NewFlowStyle;
end;

procedure TForm2.btRemoveButtonClick(Sender: TObject);
var
  Button : TButton;
begin
  if FStack.Count > 0 then
  begin
    { if the stack is not empty, pop the last button }
    Button := FStack.Pop() as TButton;
    Button.Free();
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Create a new TFlowPanel pane and align it to client }
  FFlowPane := TFlowPanel.Create(Self);
  FFlowPane.Parent := Self;
  FFlowPane.Align := alClient;

  { Set the initial flowing style }
  FFlowPane.FlowStyle := fsLeftRightTopBottom;

  { Create the button stack }
  FStack := TObjectStack.Create();
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
  { Destroy the stack also }
  FStack.Free();
end;
```

3.1.24 TCustomGridPanel Class

The base class for all grid panel components.

Class Hierarchy

Pascal

```
TCustomGridPanel = class(TCustomPanel);
```

C++

```
class TCustomGridPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TCustomGridPanel as a base class when defining grid panels that include other controls as constituent parts.

See Also

TGridPanel (see page 941)

3.1.24.1 TCustomGridPanel Members

The following tables list the members exposed by TCustomGridPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AlignControls (see page 784)	Aligns any controls for which the control is the parent within a specified area of the control.
	AutoAddColumn (see page 785)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 785)	This is AutoAddRow, a member of class TCustomGridPanel.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CellIndexToCell (see page 785)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 785)	This is CellToCellIndex, a member of class TCustomGridPanel.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Loaded (see page 786)	Initializes the control after it is loaded from a stream.
	Paint (see page 787)	Renders the image of the panel.
	RemoveEmptyAutoAddColumns (see page 787)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 787)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.
	UpdateControlOriginalParentSize (see page 788)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 812)	Specifies the background color of the control.
	FullRepaint (see page 813)	Determines how the panel repaints itself when it is resized.
	Locked (see page 813)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 814)	Determines where a control looks for its color information.
	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Methods

	Name	Description
	Create (see page 783)	Creates and initializes an instance of TCustomPanel.
	Destroy (see page 783)	Destroys an instance of TCustomControl.
	GetChildren (see page 785)	Calls a specified method for each child of the control.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	IsColumnEmpty (see page 786)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsRowEmpty (see page 786)	This is IsRowEmpty, a member of class TCustomGridPanel.

Public Properties

	Name	Description
	CellCount (see page 788)	This is CellCount, a member of class TCustomGridPanel.
	CellRect (see page 788)	Returns the screen coordinates of a cell in the grid.
	CellSize (see page 789)	This is CellSize, a member of class TCustomGridPanel.
	ColumnCollection (see page 789)	Sets the number and size of the columns on the grid panel.
	ColumnSpanIndex (see page 789)	This is ColumnSpanIndex, a member of class TCustomGridPanel.

	ControlCollection (see page 789)	This is ControlCollection, a member of class TCustomGridPanel.
	ExpandStyle (see page 790)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	ParentBackground (see page 813)	Determines whether control uses parent's theme background.
	RowCollection (see page 790)	Sets the number and size of the rows on the grid panel.
R	RowSpanIndex (see page 790)	This is RowSpanIndex, a member of class TCustomGridPanel.

3.1.24.2 TCustomGridPanel.Create Constructor

Creates and initializes an instance of TCustomPanel.

Pascal

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

C++

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

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner

3.1.24.3 TCustomGridPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomGridPanel();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

3.1.24.4 TCustomGridPanel Methods

The methods of the TCustomGridPanel class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 784)	Aligns any controls for which the control is the parent within a specified area of the control.
	AutoAddColumn (see page 785)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 785)	This is AutoAddRow, a member of class TCustomGridPanel.
	CellIndexToCell (see page 785)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 785)	This is CellToCellIndex, a member of class TCustomGridPanel.
	Loaded (see page 786)	Initializes the control after it is loaded from a stream.
	Paint (see page 787)	Renders the image of the panel.
	RemoveEmptyAutoAddColumns (see page 787)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 787)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.
	UpdateControlOriginalParentSize (see page 788)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

Public Methods

	Name	Description
	GetChildren (see page 785)	Calls a specified method for each child of the control.
	IsColumnEmpty (see page 786)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsRowEmpty (see page 786)	This is IsRowEmpty, a member of class TCustomGridPanel.

3.1.24.4.1 TCustomGridPanel.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

3.1.24.4.2 TCustomGridPanel.AutoAddColumn Method

Pascal

```
function AutoAddColumn: TColumnItem;
```

C++

```
__fastcall TColumnItem AutoAddColumn();
```

Description

This is AutoAddColumn, a member of class TCustomGridPanel.

3.1.24.4.3 TCustomGridPanel.AutoAddRow Method

Pascal

```
function AutoAddRow: TRowItem;
```

C++

```
__fastcall TRowItem AutoAddRow();
```

Description

This is AutoAddRow, a member of class TCustomGridPanel.

3.1.24.4.4 TCustomGridPanel.CellIndexToCell Method

Pascal

```
procedure CellIndexToCell(AIndex: Integer; var AColumn: Integer; var ARow: Integer);
```

C++

```
__fastcall CellIndexToCell(int AIndex, int AColumn, int ARow);
```

Description

This is CellIndexToCell, a member of class TCustomGridPanel.

3.1.24.4.5 TCustomGridPanel.CellToCellIndex Method

Pascal

```
function CellToCellIndex(AColumn: Integer; ARow: Integer): Integer;
```

C++

```
__fastcall int CellToCellIndex(int AColumn, int ARow);
```

Description

This is CellToCellIndex, a member of class TCustomGridPanel.

3.1.24.4.6 TCustomGridPanel.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

3.1.24.4.7 TCustomGridPanel.IsColumnEmpty Method

Pascal

```
function IsColumnEmpty(AColumn: Integer): Boolean;
```

C++

```
__fastcall Boolean IsColumnEmpty(int AColumn);
```

Description

This is IsColumnEmpty, a member of class TCustomGridPanel.

3.1.24.4.8 TCustomGridPanel.IsRowEmpty Method

Pascal

```
function IsRowEmpty(ARow: Integer): Boolean;
```

C++

```
__fastcall Boolean IsRowEmpty(int ARow);
```

Description

This is IsRowEmpty, a member of class TCustomGridPanel.

3.1.24.4.9 TCustomGridPanel.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

3.1.24.4.10 TCustomGridPanel.Paint Method

Renders the image of the panel.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Override the Paint method to change the way the panel is drawn. The Paint method for TCustomPanel draws the bevels of the panel, sets the brush and font for the panel, and fills in the interior of the panel. Finally, Paint writes the value of Caption on the panel.

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

Brush

Caption

Canvas

3.1.24.4.11 TCustomGridPanel.RemoveEmptyAutoAddColumns Method

Pascal

```
procedure RemoveEmptyAutoAddColumns;
```

C++

```
__fastcall RemoveEmptyAutoAddColumns();
```

Description

This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.

3.1.24.4.12 TCustomGridPanel.RemoveEmptyAutoAddRows Method

Pascal

```
procedure RemoveEmptyAutoAddRows;
```

C++

```
__fastcall RemoveEmptyAutoAddRows();
```

Description

This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.

3.1.24.4.13 TCustomGridPanel.UpdateControlOriginalParentSize Method

Pascal

```
procedure UpdateControlOriginalParentSize(AControl: TControl; var AOriginalParentSize:
TPoint); override;
```

C++

```
virtual __fastcall UpdateControlOriginalParentSize(TControl * AControl, TPoint
AOriginalParentSize);
```

Description

This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

3.1.24.5 TCustomGridPanel Properties

The properties of the TCustomGridPanel class are listed here.

Public Properties

	Name	Description
R	CellCount (see page 788)	This is CellCount, a member of class TCustomGridPanel.
R	CellRect (see page 788)	Returns the screen coordinates of a cell in the grid.
R	CellSize (see page 789)	This is CellSize, a member of class TCustomGridPanel.
	ColumnCollection (see page 789)	Sets the number and size of the columns on the grid panel.
R	ColumnSpanIndex (see page 789)	This is ColumnSpanIndex, a member of class TCustomGridPanel.
	ControlCollection (see page 789)	This is ControlCollection, a member of class TCustomGridPanel.
	ExpandStyle (see page 790)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	RowCollection (see page 790)	Sets the number and size of the rows on the grid panel.
R	RowSpanIndex (see page 790)	This is RowSpanIndex, a member of class TCustomGridPanel.

3.1.24.5.1 TCustomGridPanel.CellCount Property

Pascal

```
property CellCount: Integer;
```

C++

```
__property int CellCount;
```

Description

This is CellCount, a member of class TCustomGridPanel.

3.1.24.5.2 TCustomGridPanel.CellRect Property

Returns the screen coordinates of a cell in the grid.

Pascal

```
property CellRect [AColumn, ARow: Integer]: TRect;
```

C++

```
__property TRect CellRect[int AColumn, ARow];
```

Description

Use the CellRect property to return the screen coordinates of a cell in the grid.

3.1.24.5.3 TCustomGridPanel.CellSize Property

Pascal

```
property CellSize [AColumn, ARow: Integer]: TPoint;
```

C++

```
__property TPoint CellSize[int AColumn, ARow];
```

Description

This is CellSize, a member of class TCustomGridPanel.

3.1.24.5.4 TCustomGridPanel.ColumnCollection Property

Sets the number and size of the columns on the grid panel.

Pascal

```
property ColumnCollection: TColumnCollection;
```

C++

```
__property TColumnCollection ColumnCollection;
```

Description

Use the ColumnCollection property to set the number of columns as well as the size of each column. You can use the SizeStyle property to specify whether column sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

3.1.24.5.5 TCustomGridPanel.ColumnSpanIndex Property

Pascal

```
property ColumnSpanIndex [AColumn, ARow: Integer]: Integer;
```

C++

```
__property int ColumnSpanIndex[int AColumn, ARow];
```

Description

This is ColumnSpanIndex, a member of class TCustomGridPanel.

3.1.24.5.6 TCustomGridPanel.ControlCollection Property

Pascal

```
property ControlCollection: TControlCollection;
```

C++

```
__property TControlCollection ControlCollection;
```

Description

This is ControlCollection, a member of class TCustomGridPanel.

3.1.24.5.7 TCustomGridPanel.ExpandStyle Property

Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.

Pascal

```
property ExpandStyle: TExpandStyle;
```

C++

```
__property TExpandStyle ExpandStyle;
```

Description

Use the ExpandStyle property to specify how a grid panel is expanded when each cell of the grid contains a control, and a new control is added.

3.1.24.5.8 TCustomGridPanel.RowCollection Property

Sets the number and size of the rows on the grid panel.

Pascal

```
property RowCollection: TRowCollection;
```

C++

```
__property TRowCollection RowCollection;
```

Description

Use the RowCollection property to set the number of rows as well as the size of each row. You can use the SizeStyle property to specify whether row sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

3.1.24.5.9 TCustomGridPanel.RowSpanIndex Property

Pascal

```
property RowSpanIndex [AColumn, ARow: Integer]: Integer;
```

C++

```
__property int RowSpanIndex[int AColumn, ARow];
```

Description

This is RowSpanIndex, a member of class TCustomGridPanel.

3.1.25 TCustomLabeledEdit Class

TCustomLabeledEdit is the base class for edit controls that have an integrated label.

Class Hierarchy

Pascal

```
TCustomLabeledEdit = class(TCustomEdit);
```

C++

```
class TCustomLabeledEdit : public TCustomEdit;
```

File

ExtCtrls

Description

TCustomLabeledEdit is the base class for edit controls that have an associated label. This class introduces new properties for managing a label that sits on one side of the edit control.

Do not create instances of TCustomLabeledEdit. Use TCustomLabeledEdit as a base class when declaring edit controls that include a label. Properties and methods of TCustomLabeledEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TEdit

TLabeledEdit (see page 1100)

TBoundLabel (see page 174)

3.1.25.1 TCustomLabeledEdit Members

The following tables list the members exposed by TCustomLabeledEdit.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 792)	Responds when the BiDiMode property changes.
	CMEnabledchanged (see page 793)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 793)	Responds when the Visible property changes.
V	Notification (see page 793)	Responds to notifications when objects are created or destroyed.
V	SetName (see page 794)	Sets the name of the control.
V	SetParent (see page 795)	Sets the value of the Parent property.

Public Methods

	Name	Description
V	Create (see page 791)	Creates an instance of TCustomLabeledEdit (see page 790).
V	SetBounds (see page 794)	Sets the windowed control's boundary properties all at once.
	SetupInternalLabel (see page 795)	Initializes the control that implements the EditLabel (see page 796) property.

Public Properties

	Name	Description
R	EditLabel (see page 796)	Specifies the control that implements the label.
	LabelPosition (see page 796)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 796)	Specifies the distance, in pixels, between the label and the edit region.

3.1.25.2 TCustomLabeledEdit.Create Constructor

Creates an instance of TCustomLabeledEdit (see page 790).

Pascal

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

C++

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

Description

Call Create to instantiate a labeled edit control. AOwner specifies a component, typically the form, that is responsible for freeing the labeled edit control.

See Also

Owner

3.1.25.3 TCustomLabeledEdit Methods

The methods of the TCustomLabeledEdit class are listed here.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 792)	Responds when the BiDiMode property changes.
	CMEnabledchanged (see page 793)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 793)	Responds when the Visible property changes.
V	Notification (see page 793)	Responds to notifications when objects are created or destroyed.
V	SetName (see page 794)	Sets the name of the control.
V	SetParent (see page 795)	Sets the value of the Parent property.

Public Methods

	Name	Description
V	SetBounds (see page 794)	Sets the windowed control's boundary properties all at once.
	SetupInternalLabel (see page 795)	Initializes the control that implements the EditLabel (see page 796) property.

3.1.25.3.1 TCustomLabeledEdit.CMBidimodechanged Method

Responds when the BiDiMode property changes.

Pascal

```
procedure CMBidimodechanged(var Message: TMessage); message CM_BIDIMODECHANGED;
```

C++

```
__fastcall CMBidimodechanged(TMessage Message);
```

Description

CMBidimodechanged is called automatically when an application changes the BiDiMode property.

Message is a CM_BIDIMODECHANGED message that notifies the labeled edit of the change.

The labeled edit control responds by changing the BiDiMode property of the associated label to the value of its own BiDiMode property.

See Also

BiDiMode

EditLabel (see page 796)

3.1.25.3.2 TCustomLabeledEdit.CMEnabledchanged Method

Responds when the Enabled property changes.

Pascal

```
procedure CMEnabledchanged(var Message: TMessage); message CM_ENABLEDCHANGED;
```

C++

```
__fastcall CMEnabledchanged(TMessage Message);
```

Description

CMEnabledchanged is called automatically when an application changes the Enabled property.

Message is a CM_ENABLEDCHANGED message that notifies the labeled edit of the change.

The labeled edit control responds by changing the Enabled property of the associated label to the value of its own Enabled property.

See Also

Enabled

EditLabel (see page 796)

3.1.25.3.3 TCustomLabeledEdit.CMVisiblechanged Method

Responds when the Visible property changes.

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

CMVisiblechanged is called automatically when an application changes the Visible property.

Message is a CM_VISIBLECHANGED message that notifies the labeled edit of the change.

The labeled edit control responds by changing the Visible property of the associated label to the value of its own Visible property.

See Also

Visible

EditLabel (see page 796)

3.1.25.3.4 TCustomLabeledEdit.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This protected method is called automatically immediately after objects are created and immediately before they are destroyed. TCustomLabeledEdit (see page 790) overrides this method to update the EditLabel (see page 796) property when it receives a notification that the label component is about to be destroyed.

AComponent is the component that was just created or that is about to be destroyed.

Operation indicates whether the notification indicates the creation or destruction of AComponent.

See Also

EditLabel (see page 796)

3.1.25.3.5 TCustomLabeledEdit.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

3.1.25.3.6 TCustomLabeledEdit.SetName Method

Sets the name of the control.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. TControl updates the Text or Caption property, if appropriate, after changing the value of Name.

Override SetName to execute additional code when the value of the Name property changes.

See Also

Name

Caption

Text

3.1.25.3.7 TCustomLabeledEdit.SetParent Method

Sets the value of the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected property setter for the Parent property. TCustomLabeledEdit (see page 790) overrides SetParent to ensure that the label has the same parent as the edit control.

See Also

Parent

EditLabel (see page 796)

3.1.25.3.8 TCustomLabeledEdit.SetupInternalLabel Method

Initializes the control that implements the EditLabel (see page 796) property.

Pascal

```
procedure SetupInternalLabel;
```

C++

```
__fastcall SetupInternalLabel();
```

Description

Applications do not need to call SetupInternalLabel because this method is called when you set the EditLabel (see page 796) property.

SetupInternalLabel ensures that there is a control to implement the EditLabel (see page 796) property, adds the labeled edit control to the list of components that should be notified when the label is freed, and sets the label's FocusControl property to the labeled edit control.

See Also

EditLabel (see page 796)

FocusControl

3.1.25.4 TCustomLabeledEdit Properties

The properties of the TCustomLabeledEdit class are listed here.

Public Properties

	Name	Description
	EditLabel (see page 796)	Specifies the control that implements the label.
	LabelPosition (see page 796)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 796)	Specifies the distance, in pixels, between the label and the edit region.

3.1.25.4.1 TCustomLabeledEdit.EditLabel Property

Specifies the control that implements the label.

Pascal

```
property EditLabel: TBoundLabel;
```

C++

```
__property TBoundLabel EditLabel;
```

Description

Use EditLabel to work with the label that is associated with this labeled edit control. Use this label's properties to specify the caption that appears on the label, the position of the caption within the label, the font used to write that caption, and so on. At design time, expand the EditLabel property in the Object Inspector to set its properties.

See Also

LabelPosition (see page 796)

LabelSpacing (see page 796)

3.1.25.4.2 TCustomLabeledEdit.LabelPosition Property

Specifies the position of the label relative to the edit control.

Pascal

```
property LabelPosition: TLabelPosition;
```

C++

```
__property TLabelPosition LabelPosition;
```

Description

Set LabelPosition to indicate where the label sits relative to the edit control.

See Also

EditLabel (see page 796)

LabelSpacing (see page 796)

3.1.25.4.3 TCustomLabeledEdit.LabelSpacing Property

Specifies the distance, in pixels, between the label and the edit region.

Pascal

```
property LabelSpacing: Integer;
```

C++

```
__property int LabelSpacing;
```

Description

Set LabelSpacing to indicate the distance, in pixels, between the label and the edit control.

See Also

EditLabel (see page 796)

LabelPosition (see page 796)

3.1.26 TCustomLinkLabel Class

Base class for label containing HTML link.

Class Hierarchy

Pascal

```
TCustomLinkLabel = class(TWinControl);
```

C++

```
class TCustomLinkLabel : public TWinControl;
```

File

ExtCtrls

Description

TLinkLabel (see page 1167) is the base class for a label that can contain HTML tags, including a link. The OnLinkClick (see page 802) event is triggered by clicking the link.

See Also

TLinkLabel (see page 1167)

OnLinkClick (see page 802)

TSysLinkEvent (see page 1555)

TSysLinkType (see page 1546)

3.1.26.1 TCustomLinkLabel Members

The following tables list the members exposed by TCustomLinkLabel.

Protected Methods

	Name	Description
	CreateParams (see page 799)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 799)	Creates the underlying window.
	SetAutoSize (see page 800)	This is SetAutoSize, a member of class TCustomLinkLabel.
	UseThemes (see page 800)	This is UseThemes, a member of class TCustomLinkLabel.

Public Events

	Name	Description
	OnLinkClick (see page 802)	Event for click on TCustomLinkLabel (see page 797) HTML link.

Public Methods

	Name	Description
	Create (see page 798)	Creates an instance of TWinControl.

Public Nested Types

Name	Description
TLinkAlignment (see page 801)	This is nested type ExtCtrls.TCustomLinkLabel.TLinkAlignment.

Public Properties

	Name	Description
	Alignment (see page 800)	This is Alignment, a member of class TCustomLinkLabel.
	AutoSize (see page 800)	Specifies whether the control sizes itself automatically to accommodate its contents.
	UseVisualStyle (see page 801)	This is UseVisualStyle, a member of class TCustomLinkLabel.

3.1.26.2 TCustomLinkLabel.Create Constructor

Creates an instance of TWinControl.

Pascal

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

C++

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

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy method to free those resources.

See Also

Brush

TabOrder

CreateParented

InsertComponent

Create

Owner

3.1.26.3 TCustomLinkLabel Methods

The methods of the TCustomLinkLabel class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 799)	Initializes a window-creation parameter data structure.

	CreateWnd (see page 799)	Creates the underlying window.
	SetAutoSize (see page 800)	This is SetAutoSize, a member of class TCustomLinkLabel.
	UseThemes (see page 800)	This is UseThemes, a member of class TCustomLinkLabel.

3.1.26.3.1 TCustomLinkLabel.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

3.1.26.3.2 TCustomLinkLabel.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

3.1.26.3.3 TCustomLinkLabel.SetAutoSize Method

Pascal

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

C++

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

Description

This is SetAutoSize, a member of class TCustomLinkLabel.

3.1.26.3.4 TCustomLinkLabel.UseThemes Method

Pascal

```
function UseThemes: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseThemes();
```

Description

This is UseThemes, a member of class TCustomLinkLabel.

3.1.26.4 TCustomLinkLabel Properties

The properties of the TCustomLinkLabel class are listed here.

Public Properties

	Name	Description
	Alignment (see page 800)	This is Alignment, a member of class TCustomLinkLabel.
	AutoSize (see page 800)	Specifies whether the control sizes itself automatically to accommodate its contents.
	UseVisualStyle (see page 801)	This is UseVisualStyle, a member of class TCustomLinkLabel.

3.1.26.4.1 TCustomLinkLabel.Alignment Property

Pascal

```
property Alignment: TLinkAlignment;
```

C++

```
__property TLinkAlignment Alignment;
```

Description

This is Alignment, a member of class TCustomLinkLabel.

3.1.26.4.2 TCustomLinkLabel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.26.4.3 TCustomLinkLabel.UseVisualStyle Property

Pascal

```
property UseVisualStyle: Boolean;
```

C++

```
__property Boolean UseVisualStyle;
```

Description

This is UseVisualStyle, a member of class TCustomLinkLabel.

3.1.26.5 TCustomLinkLabel Nested Types

The nested types of the TCustomLinkLabel class are listed here.

Public Nested Types

Name	Description
TLinkAlignment (see page 801)	This is nested type ExtCtrls.TCustomLinkLabel.TLinkAlignment.

3.1.26.5.1 TCustomLinkLabel.TLinkAlignment Nested Type

Pascal

```
TLinkAlignment = taLeftJustify..taRightJustify;
```

C++

```
taLeftJustify..taRightJustify TLinkAlignment;
```

Description

This is nested type ExtCtrls.TCustomLinkLabel.TLinkAlignment.

3.1.26.6 TCustomLinkLabel Events

The events of the TCustomLinkLabel class are listed here.

Public Events

	Name	Description
	OnLinkClick (see page 802)	Event for click on TCustomLinkLabel (see page 797) HTML link.

3.1.26.6.1 TCustomLinkLabel.OnLinkClick Event

Event for click on TCustomLinkLabel (see page 797) HTML link.

Pascal

```
property OnLinkClick: TSysLinkEvent;
```

C++

```
__property TSysLinkEvent OnLinkClick;
```

Description

OnLinkClick is triggered by a click on a TCustomLinkLabel (see page 797)'s HTML link. The event identifies the link URI and link type.

See Also

TCustomLinkLabel (see page 797)

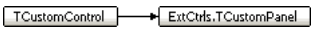
TSysLinkEvent (see page 1555)

TSysLinkType (see page 1546)

3.1.27 TCustomPanel Class

TCustomPanel is the base class for all panel components.

Class Hierarchy



Pascal

```
TCustomPanel = class(TCustomControl);
```

C++

```
class TCustomPanel : public TCustomControl;
```

File

ExtCtrls

Description

Use TCustomPanel as a base class when defining objects that include other controls as constituent parts. Panel components are controls can contain other controls, grouping them together.

When a panel is aligned with the form using the Align property, it maintains the same relative position to the form even when the form is resized. For example, a panel can be aligned so that it always remains on the top of the form, even when the user changes the shape and size of the form. This makes panels useful for implementing components that act like tool bars or status bars. For example, construct a tool bar or tool palette by adding speed buttons that execute commands or set modes to a panel .

See Also

TForm (see page 2241)

TPanel (see page 1326)

TCustomFrame (see page 2237)

TGroupBox

3.1.27.1 TCustomPanel Members

The following tables list the members exposed by TCustomPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 806)	Renders the image of the panel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 812)	Specifies the background color of the control.
	FullRepaint (see page 813)	Determines how the panel repaints itself when it is resized.
	Locked (see page 813)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 814)	Determines where a control looks for its color information.
	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Methods

	Name	Description
	Create (see page 803)	Creates and initializes an instance of TCustomPanel (see page 802).
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.

Public Properties

	Name	Description
	ParentBackground (see page 813)	Determines whether control uses parent's theme background.

3.1.27.2 TCustomPanel.Create Constructor

Creates and initializes an instance of TCustomPanel (see page 802).

Pascal

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

C++

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

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner

3.1.27.3 TCustomPanel Methods

The methods of the TCustomPanel class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 806)	Renders the image of the panel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.

Public Methods

	Name	Description
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.

3.1.27.3.1 TCustomPanel.AdjustClientRect Method

Adjusts the ClientRect property for the idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the panel needs accurate information on where to place child controls within the client area. AdjustClientRect reduces ClientRect by the width of the inner and outer bevels so that child objects are not placed on the beveled region.

See Also

ClientRect

Controls

3.1.27.3.2 TCustomPanel.CanAutoSize Method

Adjusts the way the panel responds to resize attempts.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the panel. It allows the

panel to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the panel. CanAutoSize adjusts these values so that the panel's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the panel's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

See Also

AutoSize

AdjustSize

ConstrainedResize

OnCanResize

3.1.27.3.3 TCustomPanel.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically to initialize the window-creation parameters whenever the window for the panel needs to be created. CreateParams overrides the inherited method to implement the BorderStyle (see page 810) property.

See Also

BorderStyle (see page 810)

CreateParams

3.1.27.3.4 TCustomPanel.GetControlsAlignment Method

Returns the alignment of text in the panel.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

GetControlsAlignment returns the value of the Alignment (see page 807) property.

See Also

Alignment (see page 807)

UseRightToLeftAlignment

3.1.27.3.5 TCustomPanel.Paint Method

Renders the image of the panel.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Override the Paint method to change the way the panel is drawn. The Paint method for TCustomPanel (see page 802) draws the bevels of the panel, sets the brush and font for the panel, and fills in the interior of the panel. Finally, Paint writes the value of Caption on the panel.

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

Brush

Caption

Canvas

3.1.27.3.6 TCustomPanel.SetParentBackground Method

Pascal

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

C++

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

Description

This is SetParentBackground, a member of class TCustomPanel.

3.1.27.4 TCustomPanel Properties

The properties of the TCustomPanel class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 807)	Determines how the caption is aligned within the panel.
	BevelInner (see page 807)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 808)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 809)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 810)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 811)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 812)	Specifies the background color of the control.
	FullRepaint (see page 813)	Determines how the panel repaints itself when it is resized.
	Locked (see page 813)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 814)	Determines where a control looks for its color information.

	ShowCaption (see page 814)	This is ShowCaption, a member of class TCustomPanel.
	VerticalAlignment (see page 814)	Sets the vertical position of the caption.

Public Properties

	Name	Description
	ParentBackground (see page 813)	Determines whether control uses parent's theme background.

3.1.27.4.1 TCustomPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication (see page 1924). ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment (see page 814) property to control the vertical position of the caption.

See Also

Hint (see page 1976)

Caption

3.1.27.4.2 TCustomPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TPanelBevel;
```

C++

```
__property TPanelBevel BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth (see page 811) property. The width of the bevels, if they are drawn, is the BevelWidth (see page 809) property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
```

```
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 808)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.27.4.3 TCustomPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TPanelBevel;
```

C++

```
__property TPanelBevel BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth (see page 811) property. The width of the outer bevel is specified in the BevelWidth (see page 809) property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
```

```
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.27.4.4 TCustomPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth (see page 811), which is the space between the bevels.

If both the BevelInner (see page 807) and BevelOuter (see page 808) properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner (see page 807) and BevelOuter (see page 808) properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
```

```
  BorderWidth properties.
  */
  void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
  {
    Panel1->Align = alBottom;
    Panel1->BevelInner = bvLowered;
    Panel1->BevelOuter = bvRaised;
    Panel1->BorderWidth = 1;
    Panel1->BevelWidth = 1;
  }
```

Delphi Examples:

```
  {
  This example uses a panel component and a button named
  CreateStatusLine on a form. The code moves the panel to the
  bottom of the form when the user clicks the button and gives
  the panel the appearance of a status line by changing the
  value of the BevelInner, BevelOuter, BevelWidth, and
  BorderWidth properties.
  }
  procedure TForm1.CreateStatusLineClick(Sender: TObject);
  begin
    with Panel1 do
    begin
      Align := alBottom;
      BevelInner := bvLowered;
      BevelOuter := bvRaised;
      BorderWidth := 1;
      BevelWidth := 1;
    end;
  end;
```

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

BorderWidth (see page 811)

3.1.27.4.5 TCustomPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth (see page 811) of the panel. The BorderWidth (see page 811) of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 811)

3.1.27.4.6 TCustomPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner (see page 807) is bvNone. If the Alignment (see page 807) property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner (see page 807) is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle (see page 810) property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
```

```
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 807)

BevelInner (see page 807)

BevelOuter (see page 808)

BorderStyle (see page 810)

Caption

3.1.27.4.7 TCustomPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.27.4.8 TCustomPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate

3.1.27.4.9 TCustomPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align

TOleContainer (see page 5969)

3.1.27.4.10 TCustomPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.27.4.11 TCustomPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.27.4.12 TCustomPanel.ShowCaption Property

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

This is ShowCaption, a member of class TCustomPanel.

3.1.27.4.13 TCustomPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment (see page 807) property to control the horizontal position of the caption.

See Also

Caption

3.1.28 TCustomRadioGroup Class

TCustomRadioGroup is the base class for radio-group components.

Class Hierarchy

Pascal

```
TCustomRadioGroup = class(TCustomGroupBox);
```

C++

```
class TCustomRadioGroup : public TCustomGroupBox;
```

File

ExtCtrls

Description

Radio groups are collections of radio buttons in a single group box. TCustomRadioGroup is the class from which radio-group components—including TRadioGroup (see page 1406) and TDBRadioGroup—descend.

Radio buttons that are contained directly in the same control component are said to be "grouped." When the user checks a radio button, all other radio buttons in its group become unchecked. Hence, two radio buttons on a form can be checked at the same time only if they are placed in separate containers, such as group boxes.

See Also

TRadioGroup (see page 1406)

TDBRadioGroup

TRadioButton

3.1.28.1 TCustomRadioGroup Members

The following tables list the members exposed by TCustomRadioGroup.

Protected Methods

	Name	Description
	CanModify (see page 817)	Determines whether the user can select a different radio button in the group.
	Loaded (see page 818)	Initializes the radio group after it is loaded into memory.
	ReadState (see page 819)	Prepares the control for having its properties assigned values from a stream.

Protected Properties

	Name	Description
	Columns (see page 820)	Specifies number of columns in the radio group.
	ItemIndex (see page 820)	Indicates which radio button in the group is currently selected.
	Items (see page 821)	Lists the radio buttons in the radio group.

Public Methods

	Name	Description
	Create (see page 816)	Creates and initializes a TCustomRadioGroup (see page 815) instance.
	Destroy (see page 816)	Destroys the TCustomRadioGroup (see page 815) instance and the radio buttons it contains.
	FlipChildren (see page 817)	Prevents the default behavior of flipping the child radio buttons.
	GetChildren (see page 818)	Prevents the streaming system from loading and saving child controls in the usual fashion.

Public Properties

	Name	Description
	Buttons (see page 819)	Access individual TRadioButton objects.
	WordWrap (see page 822)	This is WordWrap, a member of class TCustomRadioGroup.

3.1.28.2 TCustomRadioGroup.Create Constructor

Creates and initializes a TCustomRadioGroup (see page 815) instance.

Pascal

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

C++

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

Description

Call Create to instantiate a radio group at runtime. Radio group components added to forms at design time are created automatically.

AOwner is the component that is responsible for freeing the radio group instance. It becomes the value of the Owner property.

See Also

Owner

3.1.28.3 TCustomRadioGroup.Destroy Destructor

Destroys the TCustomRadioGroup (see page 815) instance and the radio buttons it contains.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRadioGroup();
```

Description

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

Destroy frees the memory for each item in the group, then calls the inherited destructor.

See Also

System::TObject::Free

3.1.28.4 TCustomRadioGroup Methods

The methods of the TCustomRadioGroup class are listed here.

Protected Methods

	Name	Description
	CanModify (see page 817)	Determines whether the user can select a different radio button in the group.
	Loaded (see page 818)	Initializes the radio group after it is loaded into memory.
	ReadState (see page 819)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	FlipChildren (see page 817)	Prevents the default behavior of flipping the child radio buttons.
	GetChildren (see page 818)	Prevents the streaming system from loading and saving child controls in the usual fashion.

3.1.28.4.1 TCustomRadioGroup.CanModify Method

Determines whether the user can select a different radio button in the group.

Pascal

```
function CanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanModify();
```

Description

Event handlers call CanModify to verify that the user is allowed to select a different radio button in the group. As defined in TCustomRadioGroup (see page 815), this method always returns true; override it to return false in situations where the user should not be able to make changes.

See Also

CanModify

3.1.28.4.2 TCustomRadioGroup.FlipChildren Method

Prevents the default behavior of flipping the child radio buttons.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren overrides the inherited FlipChildren method to disable flipping the radio buttons.

To flip the children of a radio group, use the BiDiMode property.

See Also

BiDiMode

3.1.28.4.3 TCustomRadioGroup.GetChildren Method

Prevents the streaming system from loading and saving child controls in the usual fashion.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called in the streaming sequence that loads and saves VCL components. Because radio groups load and save their buttons using the ReadState (see page 819) and WriteState methods, TCustomRadioGroup (see page 815) overrides TWinControl to disable the GetChildren method.

See Also

GetChildren

GetChildren

GetParentComponent

readstate (see page 819)

WriteState

3.1.28.4.4 TCustomRadioGroup.Loaded Method

Initializes the radio group after it is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The component streaming system calls Loaded automatically after the radio group's form is loaded into memory so that it can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method to ensure that the buttons in the group are properly arranged.

See Also

Items (see page 821)

3.1.28.4.5 TCustomRadioGroup.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

3.1.28.5 TCustomRadioGroup Properties

The properties of the TCustomRadioGroup class are listed here.

Protected Properties

	Name	Description
	Columns (see page 820)	Specifies number of columns in the radio group.
	ItemIndex (see page 820)	Indicates which radio button in the group is currently selected.
	Items (see page 821)	Lists the radio buttons in the radio group.

Public Properties

	Name	Description
R	Buttons (see page 819)	Access individual TRadioButton objects.
	WordWrap (see page 822)	This is WordWrap, a member of class TCustomRadioGroup.

3.1.28.5.1 TCustomRadioGroup.Buttons Property

Access individual TRadioButton objects.

Pascal

```
property Buttons [Index: Integer]: TRadioButton;
```

C++

```
__property TRadioButton Buttons[int Index];
```

Description

Buttons provides direct access to the individual TRadioButton objects created by the control. Buttons is a read-only property. This means you can access individual TRadioButton objects, but you cannot add or delete TRadioButton objects. To modify the button list, change the Items (see page 821) property.

Index is the zero-origin button index. If Index is equal to or greater than the number of buttons, an error is raised.

See Also

Items (see page 821)

3.1.28.5.2 TCustomRadioGroup.Columns Property

Specifies number of columns in the radio group.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Columns determines the number of columns in the radio group if Orientation is orVertical (the default), and the number of rows if Orientation is orHorizontal. Its value can range from 1 to 16. The default value is 1, which means that the radio buttons are arranged in a single line.

3.1.28.5.3 TCustomRadioGroup.ItemIndex Property

Indicates which radio button in the group is currently selected.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex holds the index of the selected radio button in the Items (see page 821) list. (The first button is 0.) The value of ItemIndex changes at runtime as the user selects radio buttons. If you want one of the buttons to appear selected when the application starts, assign that button to ItemIndex at design time; otherwise, leave ItemIndex set to the default value of -1, which means that no button is selected.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

Items (see page 821)

3.1.28.5.4 TCustomRadioGroup.Items Property

Lists the radio buttons in the radio group.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Items holds a TStrings object that lists the captions of the radio buttons in the group. In TCustomRadioGroup (see page 815) descendants, where this property is published, radio buttons can be added or removed by editing the Items list from the Object Inspector.

Note: A TRadioButton object is created automatically for each string in Items. You cannot directly add or remove TRadioButton objects, but you can access individual TButton objects through the Buttons (see page 819) property.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
```

```
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

ItemIndex (see page 820)

TRadioButton

Buttons (see page 819)

3.1.28.5.5 TCustomRadioGroup.WordWrap Property

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

This is WordWrap, a member of class TCustomRadioGroup.

3.1.29 TCustomTrayIcon Class

The base class for TTrayIcon (see page 1528).

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TCustomTrayIcon = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TCustomTrayIcon : public TComponent;
```

File

ExtCtrls

Description

TCustomTrayIcon is the base class for TTrayIcon (see page 1528), which places an icon in the system tray next to the clock.

3.1.29.1 TCustomTrayIcon Members

The following tables list the members exposed by TCustomTrayIcon.

Protected Methods

	Name	Description
	DoOnAnimate (see page 825)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 825)	This is GetAnimateInterval, a member of class TCustomTrayIcon.
	GetBalloonTimeout (see page 825)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
	Notification (see page 826)	Forwards notification messages to all owned components.
	SetAnimate (see page 827)	This is SetAnimate, a member of class TCustomTrayIcon.
	SetAnimateInterval (see page 827)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 827)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 827)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 828)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetHint (see page 828)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 828)	This is SetIcon, a member of class TCustomTrayIcon.
	SetIconIndex (see page 828)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 829)	This is SetIconList, a member of class TCustomTrayIcon.
	SetVisible (see page 829)	This is SetVisible, a member of class TCustomTrayIcon.
	WindowProc (see page 829)	This is WindowProc, a member of class TCustomTrayIcon.

Protected Properties

	Name	Description
	Data (see page 832)	This is Data, a member of class TCustomTrayIcon.

Public Events

	Name	Description
	OnAnimate (see page 834)	Occurs each time the icon changes when the tray icon is animating.
	OnBalloonClick (see page 834)	This is OnBalloonClick, a member of class TCustomTrayIcon.
	OnClick (see page 834)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 834)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 835)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 835)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 835)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Public Methods

	Name	Description
	Create (see page 824)	Creates and initializes a TCustomTrayIcon (see page 822) instance.
	Destroy (see page 824)	Frees the memory associated with the TCustomTrayIcon (see page 822) object.

	Refresh (see page 826)	This is the overview for the Refresh method overload.
	SetDefaultIcon (see page 828)	Sets the tray icon to the default application icon.
	ShowBalloonHint (see page 829)	Displays the balloon hint.

Public Properties

	Name	Description
	Animate (see page 830)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 830)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 831)	Specifies an icon for the balloon hint.
	BalloonHint (see page 831)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 831)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 831)	Specifies the text that appears as the balloon title.
	Hint (see page 832)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 832)	Specifies the default icon shown in the system tray.
	IconIndex (see page 832)	This is an index into the Icons (see page 833) image list which is the icon currently being displayed in the system tray.
	Icons (see page 833)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 833)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 833)	Determines whether the icon appears in the system tray.

3.1.29.2 TCustomTrayIcon.Create Constructor

Creates and initializes a TCustomTrayIcon (see page 822) instance.

Pascal

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

C++

```
virtual __fastcall TCustomTrayIcon(TComponent * Owner);
```

Description

Creates and initializes a TCustomTrayIcon (see page 822) instance.

3.1.29.3 TCustomTrayIcon.Destroy Destructor

Frees the memory associated with the TCustomTrayIcon (see page 822) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTrayIcon();
```

Description

Frees the memory associated with the TCustomTrayIcon (see page 822) object.

3.1.29.4 TCustomTrayIcon Methods

The methods of the TCustomTrayIcon class are listed here.

Protected Methods

	Name	Description
	DoOnAnimate (see page 825)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 825)	This is GetAnimateInterval, a member of class TCustomTrayIcon.
	GetBalloonTimeout (see page 825)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
	Notification (see page 826)	Forwards notification messages to all owned components.
	SetAnimate (see page 827)	This is SetAnimate, a member of class TCustomTrayIcon.
	SetAnimateInterval (see page 827)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 827)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 827)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 828)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetHint (see page 828)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 828)	This is SetIcon, a member of class TCustomTrayIcon.
	SetIconIndex (see page 828)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 829)	This is SetIconList, a member of class TCustomTrayIcon.
	SetVisible (see page 829)	This is SetVisible, a member of class TCustomTrayIcon.
	WindowProc (see page 829)	This is WindowProc, a member of class TCustomTrayIcon.

Public Methods

	Name	Description
	Refresh (see page 826)	This is the overview for the Refresh method overload.
	SetDefaultIcon (see page 828)	Sets the tray icon to the default application icon.
	ShowBalloonHint (see page 829)	Displays the balloon hint.

3.1.29.4.1 TCustomTrayIcon.DoOnAnimate Method

Pascal

```
procedure DoOnAnimate(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DoOnAnimate(TObject * Sender);
```

Description

This is DoOnAnimate, a member of class TCustomTrayIcon.

3.1.29.4.2 TCustomTrayIcon.GetAnimateInterval Method

Pascal

```
function GetAnimateInterval: Cardinal;
```

C++

```
__fastcall unsigned GetAnimateInterval();
```

Description

This is GetAnimateInterval, a member of class TCustomTrayIcon.

3.1.29.4.3 TCustomTrayIcon.GetBalloonTimeout Method

Pascal

```
function GetBalloonTimeout: Integer;
```

C++

```
__fastcall int GetBalloonTimeout();
```

Description

This is GetBalloonTimeout, a member of class TCustomTrayIcon.

3.1.29.4.4 TCustomTrayIcon.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

3.1.29.4.5 Refresh Method

This is the overview for the Refresh method overload.

Overload List

	Name	Description
	TCustomTrayIcon.Refresh () (see page 826)	Repaints the tray icon on the screen.
	TCustomTrayIcon.Refresh (Integer) (see page 826)	Repaints the tray icon on the screen.

3.1.29.4.5.1 TCustomTrayIcon.Refresh Method ()

Repaints the tray icon on the screen.

Pascal

```
function Refresh(Message: Integer): Boolean; overload;
procedure Refresh; overload;
```

C++

```
__fastcall Boolean Refresh(int Message);
__fastcall Refresh();
```

Description

Repaints the tray icon on the screen.

3.1.29.4.6 TCustomTrayIcon.SetAnimate Method

Pascal

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

C++

```
__fastcall SetAnimate(Boolean Value);
```

Description

This is SetAnimate, a member of class TCustomTrayIcon.

3.1.29.4.7 TCustomTrayIcon.SetAnimateInterval Method

Pascal

```
procedure SetAnimateInterval(Value: Cardinal);
```

C++

```
__fastcall SetAnimateInterval(unsigned Value);
```

Description

This is SetAnimateInterval, a member of class TCustomTrayIcon.

3.1.29.4.8 TCustomTrayIcon.SetBalloonHint Method

Pascal

```
procedure SetBalloonHint(const Value: string);
```

C++

```
__fastcall SetBalloonHint(const UnicodeString Value);
```

Description

This is SetBalloonHint, a member of class TCustomTrayIcon.

3.1.29.4.9 TCustomTrayIcon.SetBalloonTimeout Method

Pascal

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

C++

```
__fastcall SetBalloonTimeout(int Value);
```

Description

This is SetBalloonTimeout, a member of class TCustomTrayIcon.

3.1.29.4.10 TCustomTrayIcon.SetBalloonTitle Method

Pascal

```
procedure SetBalloonTitle(const Value: string);
```

C++

```
__fastcall SetBalloonTitle(const UnicodeString Value);
```

Description

This is SetBalloonTitle, a member of class TCustomTrayIcon.

3.1.29.4.11 TCustomTrayIcon.SetDefaultIcon Method

Sets the tray icon to the default application icon.

Pascal

```
procedure SetDefaultIcon;
```

C++

```
__fastcall SetDefaultIcon();
```

Description

This method sets the tray icon to the default application icon if the Icon (see page 832) property is unassigned. This method is generally not used.

3.1.29.4.12 TCustomTrayIcon.SetHint Method

Pascal

```
procedure SetHint(const Value: string);
```

C++

```
__fastcall SetHint(const UnicodeString Value);
```

Description

This is SetHint, a member of class TCustomTrayIcon.

3.1.29.4.13 TCustomTrayIcon.SetIcon Method

Pascal

```
procedure SetIcon(Value: TIcon);
```

C++

```
__fastcall SetIcon(TIcon Value);
```

Description

This is SetIcon, a member of class TCustomTrayIcon.

3.1.29.4.14 TCustomTrayIcon.SetIconIndex Method

Pascal

```
procedure SetIconIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetIconIndex(int Value);
```

Description

This is SetIconIndex, a member of class TCustomTrayIcon.

3.1.29.4.15 TCustomTrayIcon.SetIconList Method

Pascal

```
procedure SetIconList(Value: TImageList);
```

C++

```
__fastcall SetIconList(TImageList Value);
```

Description

This is SetIconList, a member of class TCustomTrayIcon.

3.1.29.4.16 TCustomTrayIcon.SetVisible Method

Pascal

```
procedure SetVisible(Value: Boolean); virtual;
```

C++

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

Description

This is SetVisible, a member of class TCustomTrayIcon.

3.1.29.4.17 TCustomTrayIcon.ShowBalloonHint Method

Displays the balloon hint.

Pascal

```
procedure ShowBalloonHint; virtual;
```

C++

```
virtual __fastcall ShowBalloonHint();
```

Description

Displays the balloon hint.

3.1.29.4.18 TCustomTrayIcon.WindowProc Method

Pascal

```
procedure WindowProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WindowProc(TMessage Message);
```

Description

This is WindowProc, a member of class TCustomTrayIcon.

3.1.29.5 TCustomTrayIcon Properties

The properties of the TCustomTrayIcon class are listed here.

Protected Properties

	Name	Description
	Data (see page 832)	This is Data, a member of class TCustomTrayIcon.

Public Properties

	Name	Description
	Animate (see page 830)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 830)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 831)	Specifies an icon for the balloon hint.
	BalloonHint (see page 831)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 831)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 831)	Specifies the text that appears as the balloon title.
	Hint (see page 832)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 832)	Specifies the default icon shown in the system tray.
	IconIndex (see page 832)	This is an index into the Icons (see page 833) image list which is the icon currently being displayed in the system tray.
	Icons (see page 833)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 833)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 833)	Determines whether the icon appears in the system tray.

3.1.29.5.1 TCustomTrayIcon.Animate Property

Specifies whether the icon in the system tray is static or animated.

Pascal

```
property Animate: Boolean;
```

C++

```
__property Boolean Animate;
```

Description

When set to True, the icon shown in the system tray displays, in sequence, the images in the image list specified by the Icons (see page 833) property. The images change only if the specified image list contains images. The images change on a rotating basis by the interval set by the AnimateInterval (see page 830) property. When set to False the animation stops.

3.1.29.5.2 TCustomTrayIcon.AnimateInterval Property

Controls the length of time between image changes for the tray icon image.

Pascal

```
property AnimateInterval: Cardinal;
```

C++

```
__property unsigned AnimateInterval;
```

Description

Use AnimateInterval to specify the length of time between image changes in the tray icon. The default interval is 1000 ms (1 second).

3.1.29.5.3 TCustomTrayIcon.BalloonFlags Property

Specifies an icon for the balloon hint.

Pascal

```
property BalloonFlags: TBalloonFlags;
```

C++

```
__property TBalloonFlags BalloonFlags;
```

Description

The following values specify an icon to be displayed in the balloon hint:

- bfNone
- bfInfo
- bfWarning
- bfError

3.1.29.5.4 TCustomTrayIcon.BalloonHint Property

Specifies the text within a balloon hint.

Pascal

```
property BalloonHint: string;
```

C++

```
__property UnicodeString BalloonHint;
```

Description

Use BalloonHint to specify the text string that appears when a balloon hint is shown.

3.1.29.5.5 TCustomTrayIcon.BalloonTimeout Property

Specifies the length of time that the balloon displays.

Pascal

```
property BalloonTimeout: Integer;
```

C++

```
__property int BalloonTimeout;
```

Description

Use BalloonTimeout to set the length of time that a balloon displays. By default, this value is set to 3000 ms (3 seconds).

3.1.29.5.6 TCustomTrayIcon.BalloonTitle Property

Specifies the text that appears as the balloon title.

Pascal

```
property BalloonTitle: string;
```

C++

```
__property UnicodeString BalloonTitle;
```

Description

Specifies the text that appears as the balloon title.

3.1.29.5.7 TCustomTrayIcon.Data Property

Pascal

```
property Data: TNotifyIconData;
```

C++

```
__property TNotifyIconData Data;
```

Description

This is Data, a member of class TCustomTrayIcon.

3.1.29.5.8 TCustomTrayIcon.Hint Property

Specifies the text that appears when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property UnicodeString Hint;
```

Description

Specifies the text that appears when the user moves the mouse over the control.

3.1.29.5.9 TCustomTrayIcon.Icon Property

Specifies the default icon shown in the system tray.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use the Icon property to set the default icon to be shown in the system tray. If you do not specify a value, the application icon is used.

3.1.29.5.10 TCustomTrayIcon.IconIndex Property

This is an index into the Icons (see page 833) image list which is the icon currently being displayed in the system tray.

Pascal

```
property IconIndex: Integer;
```

C++

```
__property int IconIndex;
```

Description

This is an index into the Icons (see page 833) image list which is the icon currently being displayed in the system tray.

3.1.29.5.11 TCustomTrayIcon.Icons Property

Specifies the list of images to be displayed for the icon in the system tray.

Pascal

```
property Icons: TImageList;
```

C++

```
__property TImageList Icons;
```

Description

Use this property to specify the list of images used for the icon in the system tray. These images are displayed when the Animate (see page 830) property is set to True.

3.1.29.5.12 TCustomTrayIcon.PopupMenu Property

Identifies the pop-up menu associated with the tray icon.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Use PopupMenuOn to specify when the menu is displayed.

3.1.29.5.13 TCustomTrayIcon.Visible Property

Determines whether the icon appears in the system tray.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Determines whether the icon appears in the system tray.

3.1.29.6 TCustomTrayIcon Events

The events of the TCustomTrayIcon class are listed here.

Public Events

	Name	Description
	OnAnimate (see page 834)	Occurs each time the icon changes when the tray icon is animating.
	OnBalloonClick (see page 834)	This is OnBalloonClick, a member of class TCustomTrayIcon.

	OnClick (see page 834)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 834)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 835)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 835)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 835)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

3.1.29.6.1 TCustomTrayIcon.OnAnimate Event

Occurs each time the icon changes when the tray icon is animating.

Pascal

```
property OnAnimate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAnimate;
```

Description

Occurs each time the icon changes when the tray icon is animating.

3.1.29.6.2 TCustomTrayIcon.OnBalloonClick Event

Pascal

```
property OnBalloonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBalloonClick;
```

Description

This is OnBalloonClick, a member of class TCustomTrayIcon.

3.1.29.6.3 TCustomTrayIcon.OnClick Event

Occurs when the user clicks the tray icon.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Occurs when the user clicks the tray icon.

3.1.29.6.4 TCustomTrayIcon.OnDblClick Event

Occurs when the user double clicks the tray icon.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Occurs when the user double clicks the tray icon.

3.1.29.6.5 TCustomTrayIcon.OnMouseDown Event

Occurs when the user presses a mouse button while the mouse pointer is over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Occurs when the user presses a mouse button while the mouse pointer is over a control.

3.1.29.6.6 TCustomTrayIcon.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

3.1.29.6.7 TCustomTrayIcon.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

3.1.30 TEditButton Class

Button in a TButtonedEdit (see page 220).

Class Hierarchy

Pascal

```
TEditButton = class(TPersistent);
```

C++

```
class TEditButton : public TPersistent;
```

File

ExtCtrls

Description

TEditButton is a button in a TButtonedEdit (see page 220) control.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";
```

```
    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);
```

```
end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

TButtonedEdit (see page 220)

LeftButton (see page 685)

RightButton (see page 688)

3.1.30.1 TEditButton Members

The following tables list the members exposed by TEditButton.

Enumerations

	Name	Description
	TButtonPosition (see page 840)	TButtonPosition specifies the position of the button in a buttoned edit control.

Protected Methods

	Name	Description
	GetOwner (see page 841)	Returns the owner of an object.
	UpdateBounds (see page 842)	Updates the positon and size of the image in this TEditButton (see page 835) instance depending on themes and properties of this TEditButton (see page 835).

Protected Properties

	Name	Description
	EditControl (see page 846)	EditControl is a reference to the parent buttoned edit control that is the parent of this TEditButton (see page 835).
	Glyph (see page 849)	This is Glyph, a member of class TEditButton.
	Images (see page 856)	Retrieves the ImageList associated with the parent buttoned edit control.
	Position (see page 856)	This is Position, a member of class TEditButton.

Public Methods

	Name	Description
	Create (see page 839)	Allocates memory and constructs a safely initialized instance of a TEditButton (see page 835).
	Destroy (see page 840)	Destroys the TPersistent instance and frees its memory.

Published Properties

	Name	Description
	DisabledImageIndex (see page 842)	Determines which image is displayed when the button is diabled.
	DropDownMenu (see page 846)	Drop down menu for button.
	Enabled (see page 846)	Specifies whether the embedded button is enabled.
	HotImageIndex (see page 850)	Determines which image is displayed when the mouse is over the button.
	ImageIndex (see page 853)	Determines which image is displayed when the button in the unselected state.
	PressedImageIndex (see page 856)	Determines which image is displayed when the button is pressed.
	Visible (see page 857)	Indicates button visibility.

3.1.30.2 TEditButton.Create Constructor

Allocates memory and constructs a safely initialized instance of a TEditButton (see page 835).

Pascal

```
constructor Create(EditControl: TCustomButtonedEdit; APosition: TButtonPosition); virtual;
reintroduce;
```

C++

```
virtual __fastcall TEditButton(TCustomButtonedEdit EditControl, TButtonPosition APosition);
```

Description

Use Create to create an instance of the TEditButton (see page 835) class.

EditControl (see page 846) specifies the buttoned edit control that is to be the parent of this TEditButton (see page 835) instance.

APosition specifies whether this TEditButton (see page 835) instance is the LeftButton or the RightButton of the buttoned edit parent.

See Also

TButtonedEdit (see page 220)

LeftButton (see page 248)

RightButton (see page 261)

3.1.30.3 TEditButton.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEditButton();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free

System::TObject::Destroy

3.1.30.4 TEditButton Enumerations

The enumerations of the TEditButton class are listed here.

Enumerations

	Name	Description
	TButtonPosition (see page 840)	TButtonPosition specifies the position of the button in a buttoned edit control.

3.1.30.4.1 ExtCtrls.TEditButton.TButtonPosition Enumeration

TButtonPosition specifies the position of the button in a buttoned edit control.

Pascal

```
TButtonPosition = (
  bpLeft,
  bpRight
);
```

C++

```
enum TButtonPosition {
```

```
    bpLeft,
    bpRight
};
```

File

ExtCtrls

Description

A TButtonedEdit (see page 220) has optional buttons on the left or on the right of it's edit field. These optional buttons are of type TEditButton (see page 835) and TButtonPosition is the enumeration that determines whether this TEditButton (see page 835) is on the left or the right. The following are possible values of TButtonPosition:

Value	Meaning
lpLeft	The button appears to the left of the edit field in a buttoned edit control.
lpRight	The button appears to the right of the edit field in a buttoned edit control.

3.1.30.5 TEditButton Methods

The methods of the TEditButton class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 841)	Returns the owner of an object.
	UpdateBounds (see page 842)	Updates the positon and size of the image in this TEditButton (see page 835) instance depending on themes and properties of this TEditButton (see page 835).

3.1.30.5.1 TEditButton.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

3.1.30.5.2 TEditButton.UpdateBounds Method

Updates the positon and size of the image in this TEditButton (see page 835) instance depending on themes and properties of this TEditButton (see page 835).

Pascal

```
procedure UpdateBounds; dynamic;
```

C++

```
__fastcall UpdateBounds();
```

Description

Adjusts the position and size of the image in this buttoned edit control. This adjustment takes into account theming, and the dimensions, border style and beveling of the buttoned edit control in an effort to display as much of the image as possible. Height and width of this buttoned edit control are zeroed if the image list is not currently populated.

See Also

TButtonedEdit (see page 220)

BevelKind (see page 234)

BevelInner (see page 233)

BevelOuter (see page 234)

BevelEdges (see page 233)

BorderStyle (see page 235)

3.1.30.6 TEditButton Properties

The properties of the TEditButton class are listed here.

Protected Properties

	Name	Description
R	EditControl (see page 846)	EditControl is a reference to the parent buttoned edit control that is the parent of this TEditButton (see page 835).
R	Glyph (see page 849)	This is Glyph, a member of class TEditButton.
R	Images (see page 856)	Retrieves the ImageList associated with the parent buttoned edit control.
R	Position (see page 856)	This is Position, a member of class TEditButton.

Published Properties

	Name	Description
	DisabledImageIndex (see page 842)	Determines which image is displayed when the button is diabled.
	DropDownMenu (see page 846)	Drop down menu for button.
	Enabled (see page 846)	Specifies whether the embedded button is enabled.
	HotImageIndex (see page 850)	Determines which image is displayed when the mouse is over the button.
	ImageIndex (see page 853)	Determines which image is displayed when the button in the unselected state.
	PressedImageIndex (see page 856)	Determines which image is displayed when the button is pressed.
	Visible (see page 857)	Indicates button visibility.

3.1.30.6.1 TEditButton.DisabledImageIndex Property

Determines which image is displayed when the button is diabled.

Pascal

```
property DisabledImageIndex: TImageIndex;
```

C++

```
__property TImageIndex DisabledImageIndex;
```

Description

Set DisabledImageIndex to associate the button with one of the images in the Images (see page 856) property of the parent buttoned edit control. The selected image will be displayed when the button is in the disabled state.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
```

```
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;
```

```
procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

TEditButton (see page 835)

Enabled (see page 846)

TButtonedEdit (see page 220)

Images (see page 244)

3.1.30.6.2 TEditButton.DropDownMenu Property

Drop down menu for button.

Pascal

```
property DropDownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropDownMenu;
```

Description

DropDownMenu is a TPopupMenu (see page 4935) that is activated when the button is pressed.

See Also

TPopupMenu (see page 4935)

3.1.30.6.3 TEditButton.EditControl Property

EditControl is a reference to the parent buttoned edit control that is the parent of this TEditButton (see page 835).

Pascal

```
property EditControl: TCustomButtonedEdit;
```

C++

```
__property TCustomButtonedEdit EditControl;
```

Description

EditControl is a reference to the parent buttoned edit control that is the parent of this TEditButton (see page 835). This property is set when this instance of TButtonedEdit (see page 220) is created. Use edit control to access the properties of the buttoned edit control.

See Also

TEditButton (see page 835)

Create (see page 839)

TButtonedEdit (see page 220)

3.1.30.6.4 TEditButton.Enabled Property

Specifies whether the embedded button is enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to enable or disable a embedded button. If Enabled is true, the button OnClick event handler is called when the user selects the item with the mouse. If Enabled is false, the embedded button appears dimmed and the user cannot select it.

However, calling the button OnClick event handler method works even when Enabled is false.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
```

```
    btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
    //resetting to default state
    btnEdit->Text = "Default";
    btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
    MessageDlg("Left button pressed with: " + btnEdit->Text,
              mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
              mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
```

```
    bmap   : TBitmap;
  begin
    //declaring an instance
      btnEdit := TButtonedEdit.Create(Self);

      //setting the parent so it will autodetele
      btnEdit.Parent := Self;

      //positioning
      btnEdit.Left := 10;
      btnEdit.Top := 10;

      //creating a TImageList and using it for the button
      imList := TImageList.Create(Self);
      bmap := TBitmap.Create();
      bmap.LoadFromFile('littleB_16.bmp');
      imList.Add(bmap,nil);
      bmap.LoadFromFile('little_logo_16.bmp');
      imList.Add(bmap,nil);
      btnEdit.Images := imList;

      //selecting the image for the left button and enabling it
      btnEdit.LeftButton.Visible := true;
      btnEdit.LeftButton.Enabled := true;
      btnEdit.LeftButton.ImageIndex := 0;
      btnEdit.LeftButton.DisabledImageIndex := 0;
      btnEdit.LeftButton.HotImageIndex := 0;

      //selecting the image for the right button
      //this will be enabled when the content changes
      btnEdit.RightButton.Enabled := true;
      btnEdit.RightButton.ImageIndex := 1;
      btnEdit.RightButton.DisabledImageIndex := 1;
      btnEdit.RightButton.HotImageIndex := 1;

      //selecting action for the buttoned edit to take
      btnEdit.OnEnter := btnEditEnter;
      btnEdit.OnRightButtonClick := btnEditRightClick;
      btnEdit.OnLeftButtonClick := btnEditLeftClick;

      //default text in the edit
      btnEdit.Text := 'Default';

      btnEdit.Constraints.MinHeight := 16;
      btnEdit.Height := 16;

      //finally displaying the button
      btnEdit.Show();

  end;
```

See Also

OnLeftButtonClick (see page 293)

3.1.30.6.5 TEditButton.Glyph Property

Pascal

```
property Glyph: TGlyph;
```

C++

```
__property TGlyph Glyph;
```

Description

This is Glyph, a member of class TEditButton.

3.1.30.6.6 TEditButton.HotImageIndex Property

Determines which image is displayed when the mouse is over the button.

Pascal

```
property HotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex HotImageIndex;
```

Description

Set HotImageIndex to associate the button with one of the images in the Images (see page 856) property of the parent buttoned edit control. The selected image will be displayed when the mouse is over the button.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;
```

```
    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
```

```
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;

    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
```

```
    btnEdit.Show();
end;
```

See Also

TEditButton (see page 835)

TButtonedEdit (see page 220)

Images (see page 244)

3.1.30.6.7 TEditButton.ImageIndex Property

Determines which image is displayed when the button in the unselected state.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to associate the button with one of the images in the Images (see page 856) property of the parent buttoned edit control. The selected image will be displayed when the button in the unselected state.

C++ Examples:

```
/*
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
*/
__fastcall TForm4::TForm4(TComponent* Owner)
    : TForm(Owner)
{
    //declaring an instance
    btnEdit = new TButtonedEdit(this);

    //setting the parent so it will autodetele
    btnEdit->Parent = this;

    //positioning
    btnEdit->Left = 10;
    btnEdit->Top = 10;

    //creating a TImageList and using it for the button
    TImageList* imList = new TImageList(this);
    Graphics::TBitmap* bmap = new Graphics::TBitmap();
    bmap->LoadFromFile("..\\littleB_16.bmp");
    imList->Add(bmap,NULL);
    bmap->LoadFromFile("..\\little_logo_16.bmp");
    imList->Add(bmap,NULL);
    btnEdit->Images = imList;

    //selecting the image for the left button and enabling it
    btnEdit->LeftButton->Visible = true;
    btnEdit->LeftButton->Enabled = true;
    btnEdit->LeftButton->ImageIndex = 0;
    btnEdit->LeftButton->DisabledImageIndex = 0;
    btnEdit->LeftButton->HotImageIndex = 0;
```

```
    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit->RightButton->Enabled = true;
    btnEdit->RightButton->ImageIndex = 1;
    btnEdit->RightButton->DisabledImageIndex = 1;
    btnEdit->RightButton->HotImageIndex = 1;

    //selecting action for the buttoned edit to take
    btnEdit->OnEnter = btnEditEnter;
    btnEdit->OnRightButtonClick = btnEditRightClick;
    btnEdit->OnLeftButtonClick = btnEditLeftClick;

    //default text in the edit
    btnEdit->Text = "Default";

    btnEdit->Constraints->MinHeight = 16;
    btnEdit->Height = 16;

    //finally displaying the button
    btnEdit->Show();

}
//---------------------------------------------------------------------------

void __fastcall TForm4::btnEditEnter(TObject* Sender){
   btnEdit->Text = "";
   btnEdit->RightButton->Visible = true;
}

void __fastcall TForm4::btnEditRightClick(TObject* Sender){
   //resetting to default state
   btnEdit->Text = "Default";
   btnEdit->RightButton->Visible = false;
}

void __fastcall TForm4::btnEditLeftClick(TObject* Sender){
   MessageDlg("Left button pressed with: " + btnEdit->Text,
             mtInformation, TMsgDlgButtons() << mbOK, 0);

}
```

Delphi Examples:

```
{
This example shows how to use the TButtonedEdit class. This example
implements a component like "TButtonedEdit search" that's to be
found in the tool pallete window.
Entering the control will enable the right button, left button will
display a MessageDlg and the right button will clear the input text.
}
type
  TForm4 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    btnEdit : TButtonedEdit;
    procedure btnEditEnter(Sender : Tobject);
    procedure btnEditLeftClick(Sender : Tobject);
    procedure btnEditRightClick(Sender : Tobject);
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation
```

```
{$R *.dfm}

procedure TForm4.btnEditEnter(Sender: Tobject);
begin
  btnEdit.Text := '';
  btnEdit.RightButton.Visible := true;
end;

procedure TForm4.btnEditLeftClick(Sender: Tobject);
begin
  MessageDlg('Left button pressed with: ' + btnEdit.Text,
             mtInformation, [mbYes], 0);

end;

procedure TForm4.btnEditRightClick(Sender: Tobject);
begin
  //resetting to default state
   btnEdit.Text := 'Default';
   btnEdit.RightButton.Visible := false;
end;

procedure TForm4.FormCreate(Sender: TObject);
var
  imList : TImageList;
  bmap   : TBitmap;
begin
  //declaring an instance
    btnEdit := TButtonedEdit.Create(Self);

    //setting the parent so it will autodetele
    btnEdit.Parent := Self;

    //positioning
    btnEdit.Left := 10;
    btnEdit.Top := 10;

    //creating a TImageList and using it for the button
    imList := TImageList.Create(Self);
    bmap := TBitmap.Create();
    bmap.LoadFromFile('littleB_16.bmp');
    imList.Add(bmap,nil);
    bmap.LoadFromFile('little_logo_16.bmp');
    imList.Add(bmap,nil);
    btnEdit.Images := imList;

    //selecting the image for the left button and enabling it
    btnEdit.LeftButton.Visible := true;
    btnEdit.LeftButton.Enabled := true;
    btnEdit.LeftButton.ImageIndex := 0;
    btnEdit.LeftButton.DisabledImageIndex := 0;
    btnEdit.LeftButton.HotImageIndex := 0;

    //selecting the image for the right button
    //this will be enabled when the content changes
    btnEdit.RightButton.Enabled := true;
    btnEdit.RightButton.ImageIndex := 1;
    btnEdit.RightButton.DisabledImageIndex := 1;
    btnEdit.RightButton.HotImageIndex := 1;

    //selecting action for the buttoned edit to take
    btnEdit.OnEnter := btnEditEnter;
    btnEdit.OnRightButtonClick := btnEditRightClick;
    btnEdit.OnLeftButtonClick := btnEditLeftClick;
```

```
    //default text in the edit
    btnEdit.Text := 'Default';

    btnEdit.Constraints.MinHeight := 16;
    btnEdit.Height := 16;

    //finally displaying the button
    btnEdit.Show();

end;
```

See Also

TEditButton (see page 835)

TButtonedEdit (see page 220)

Images (see page 244)

3.1.30.6.8 TEditButton.Images Property

Retrieves the ImageList associated with the parent buttoned edit control.

Pascal

```
property Images: TImageList;
```

C++

```
__property TImageList Images;
```

Description

Images is a read-only property that accesses the Images property of the parent buttoned edit control.

See Also

TImageList

Images (see page 682)

3.1.30.6.9 TEditButton.Position Property

Pascal

```
property Position: TButtonPosition;
```

C++

```
__property TButtonPosition Position;
```

Description

This is Position, a member of class TEditButton.

3.1.30.6.10 TEditButton.PressedImageIndex Property

Determines which image is displayed when the button is pressed.

Pascal

```
property PressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex PressedImageIndex;
```

Description

Set PressedImageIndex to associate the button with one of the images in the Images (see page 856) property of the parent buttoned edit control. The selected image will be displayed when the button is pressed.

See Also

TEditButton (see page 835)

TButtonedEdit (see page 220)

Images (see page 244)

3.1.30.6.11 TEditButton.Visible Property

Indicates button visibility.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible is true if the button is displayed and false otherwise.

See Also

Enabled (see page 846)

3.1.31 TFlowPanel Class

TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.

Class Hierarchy

Pascal

```
TFlowPanel = class(TCustomFlowPanel);
```

C++

```
class TFlowPanel : public TCustomFlowPanel;
```

File

ExtCtrls

Description

Use TFlowPanel to put an empty flow panel on a form. The major difference between a traditional panel and a flow panel is the way in which controls are placed. With a traditional panel, you place a control (such as a button) in a specific location. You can freely move that control to any location within the panel using the mouse. In a flow panel, each control is placed in a specific location, regardless of where you place it with the mouse. The automatic location is controlled by the FlowStyle (see page 884) property. For example, using the default FlowStyle (see page 884) property of LeftRightTopBottom, the first control you add to the flow panel snaps to the top left corner. The second control that you add snaps next to the first control, and so on.

C++ Examples:

```
//
//This example demonstrates the use of Flow panel to automatically
//align a set of controls and the different methods that can be used for that.
//A Stack is used to control the order of destruction of buttons.
//
void __fastcall TForm2::btAButtonClick(TObject *Sender)
{
    TButton *thisButton;
    TButton *lastButton;
    String temp;

    /* Sender is a TButton, cast it */
    thisButton = dynamic_cast<TButton*>(Sender);

    if (m_Stack->Count() > 0)
    {
        /* If there are buttons in stack, peek it */
        lastButton = dynamic_cast<TButton*>(m_Stack->Peek());

        /* Change the captions of this button and the last one */
        temp = thisButton->Caption;
        thisButton->Caption = lastButton->Caption;
        lastButton->Caption = temp;
    }
}

void __fastcall TForm2::btAddButtonClick(TObject *Sender)
{
    /* Create a new button */
    TButton *button = new TButton(m_FlowPane);

    button->Parent = m_FlowPane;
    button->Caption = IntToStr(m_Stack->Count());
    button->OnClick = btAButtonClick;

    /* Push the button to the stack */
    m_Stack->Push(button);
}

void __fastcall TForm2::btReArrangeClick(TObject *Sender)
{
    /* Change the current flow style (rotate them) */
    TFlowStyle newFlowStyle = m_FlowPane->FlowStyle;

    if (newFlowStyle == fsBottomTopRightLeft)
        newFlowStyle = fsLeftRightTopBottom;
    else
        newFlowStyle = newFlowStyle + 1;

    m_FlowPane->FlowStyle = newFlowStyle;
}

void __fastcall TForm2::btRemoveButtonClick(TObject *Sender)
{
    TButton *button;

    if (m_Stack->Count() > 0)
    {
        /* if the stack is not empty, pop the last button */
        button = dynamic_cast<TButton*>(m_Stack->Pop());
        delete button;
    }
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
```

```
    /* Create a new TFlowPanel pane and align it to client */
    m_FlowPane = new TFlowPanel(this);
    m_FlowPane->Parent = this;
    m_FlowPane->Align = alClient;

    /* Set the initial flowing style */
    m_FlowPane->FlowStyle = fsLeftRightTopBottom;

    /* Create the button stack */
    m_Stack = new TObjectStack();
}

void __fastcall TForm2::FormDestroy(TObject *Sender)
{
    /* Destroy the stack also */
    delete m_Stack;
}
```

Delphi Examples:

```
{
This example demonstrates the use of Flow panel to automatically
align a set of controls and the different methods that can be used for that.
A Stack is used to control the order of destruction of buttons.
}
procedure TForm2.btAButtonClick(Sender: TObject);
var
  ThisButton : TButton;
  LastButton : TButton;
  Temp       : String;
begin
  { Sender is a TButton, cast it }
  ThisButton := Sender as TButton;

  if FStack.Count > 0 then
  begin
    { If there are buttons in stack, peek it }
    LastButton := FStack.Peek() as TButton;

    { Change the captions of this button and the last one }
    Temp := ThisButton.Caption;
    ThisButton.Caption := LastButton.Caption;
    LastButton.Caption := Temp;
  end;
end;

procedure TForm2.btAddButtonClick(Sender: TObject);
var
  Button : TButton;
begin
  { Create a new button }
  Button := TButton.Create(FFlowPane);
  Button.Parent := FFlowPane;
  Button.Caption := IntToStr(FStack.Count);
  Button.OnClick := btAButtonClick;

  { Push the button to the stack }
  FStack.Push(Button);
end;

procedure TForm2.btReArrangeClick(Sender: TObject);
var
  NewFlowStyle : TFlowStyle;
begin
  { Chnage the flow order (rotate) }
  NewFlowStyle := FFlowPane.FlowStyle;
```

```
    if NewFlowStyle = fsBottomTopRightLeft then
       NewFlowStyle := fsLeftRightTopBottom
    else
      Inc(NewFlowStyle);

    FFlowPane.FlowStyle := NewFlowStyle;
  end;

  procedure TForm2.btRemoveButtonClick(Sender: TObject);
  var
    Button : TButton;
  begin
    if FStack.Count > 0 then
    begin
      { if the stack is not empty, pop the last button }
      Button := FStack.Pop() as TButton;
      Button.Free();
    end;
  end;

  procedure TForm2.FormCreate(Sender: TObject);
  begin
    { Create a new TFlowPanel pane and align it to client }
    FFlowPane := TFlowPanel.Create(Self);
    FFlowPane.Parent := Self;
    FFlowPane.Align := alClient;

    { Set the initial flowing style }
    FFlowPane.FlowStyle := fsLeftRightTopBottom;

    { Create the button stack }
    FStack := TObjectStack.Create();
  end;

  procedure TForm2.FormDestroy(Sender: TObject);
  begin
    { Destroy the stack also }
    FStack.Free();
  end;
```

See Also

TPanel (see page 1326)

3.1.31.1 TFlowPanel Members

The following tables list the members exposed by TFlowPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AlignControls (see page 776)	Aligns any controls for which the control is the parent within a specified area of the control.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 806)	Renders the image of the panel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.

Public Methods

	Name	Description
	Create (see page 775)	Creates and initializes an instance of TCustomPanel.

	Create (see page 803)	Creates and initializes an instance of TCustomPanel (see page 802).
	Destroy (see page 775)	Destroys an instance of TCustomControl.
	GetChildren (see page 776)	Calls a specified method for each child of the control.
	GetControlIndex (see page 777)	This is GetControlIndex, a member of class TCustomFlowPanel.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	SetControlIndex (see page 777)	This is SetControlIndex, a member of class TCustomFlowPanel.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 903)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 904)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 905)	Occurs when an attempt is made to resize the control.
	OnClick (see page 905)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 907)	Adjust resize constraints.
	OnContextPopup (see page 907)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 908)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 908)	Occurs when another control is docked to the control.
	OnDockOver (see page 910)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 913)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 916)	Occurs when the user drags an object over a control.
	OnEndDock (see page 919)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 919)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 920)	Occurs when a control receives the input focus.
	OnExit (see page 923)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 925)	Returns the control's docking information.
	OnMouseActivate (see page 928)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 929)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 931)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 931)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 931)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 934)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 936)	Occurs immediately after the control is resized.
	OnStartDock (see page 937)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 937)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 938)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 863)	Determines how the control aligns within its container (parent control).
	Alignment (see page 867)	Determines how the caption is aligned within the panel.
	Anchors (see page 867)	Specifies how the control is anchored to its parent.
	AutoSize (see page 868)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoWrap (see page 868)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	BevelEdges (see page 868)	Specifies which edges of the control are beveled.
	BevelInner (see page 869)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 870)	Specifies the control's bevel style.
	BevelOuter (see page 870)	Determines the style of the outer bevel of a panel.

	BevelWidth (see page 871)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 872)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 873)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 873)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 874)	Specifies a text string that identifies the control to the user.
	Color (see page 877)	Specifies the background color of the control.
	Constraints (see page 878)	Specifies the size constraints for the control.
	Ctl3D (see page 878)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 879)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 880)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 882)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 882)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 882)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 884)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlowStyle (see page 884)	Specifies the starting point and direction for controls added to a flow panel.
	Font (see page 888)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 889)	Determines how the panel repaints itself when it is resized.
	Locked (see page 889)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 889)	Specifies the padding of a control.
	ParentBackground (see page 890)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 890)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 890)	Determines where a control looks for its color information.
	ParentCtl3D (see page 891)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 891)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 891)	Determines where a control looks for its font information.
	ParentShowHint (see page 892)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 893)	Identifies the pop-up menu associated with the control.
	ShowCaption (see page 898)	This is ShowCaption, a member of class TFlowPanel.
	ShowHint (see page 898)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 899)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 900)	Determines if the user can tab to a control.
	UseDockManager (see page 900)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 901)	Sets the vertical position of the caption.
	Visible (see page 901)	Determines whether the component appears onscreen.

3.1.31.2 TFlowPanel Properties

The properties of the TFlowPanel class are listed here.

Published Properties

	Name	Description
	Align (see page 863)	Determines how the control aligns within its container (parent control).
	Alignment (see page 867)	Determines how the caption is aligned within the panel.
	Anchors (see page 867)	Specifies how the control is anchored to its parent.
	AutoSize (see page 868)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoWrap (see page 868)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	BevelEdges (see page 868)	Specifies which edges of the control are beveled.
	BevelInner (see page 869)	Determines the style of the inner bevel of a panel.

	BevelKind (see page 870)	Specifies the control's bevel style.
	BevelOuter (see page 870)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 871)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 872)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 873)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 873)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 874)	Specifies a text string that identifies the control to the user.
	Color (see page 877)	Specifies the background color of the control.
	Constraints (see page 878)	Specifies the size constraints for the control.
	Ctl3D (see page 878)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 879)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 880)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 882)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 882)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 882)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 884)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlowStyle (see page 884)	Specifies the starting point and direction for controls added to a flow panel.
	Font (see page 888)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 889)	Determines how the panel repaints itself when it is resized.
	Locked (see page 889)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 889)	Specifies the padding of a control.
	ParentBackground (see page 890)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 890)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 890)	Determines where a control looks for its color information.
	ParentCtl3D (see page 891)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 891)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 891)	Determines where a control looks for its font information.
	ParentShowHint (see page 892)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 893)	Identifies the pop-up menu associated with the control.
	ShowCaption (see page 898)	This is ShowCaption, a member of class TFlowPanel.
	ShowHint (see page 898)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 899)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 900)	Determines if the user can tab to a control.
	UseDockManager (see page 900)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 901)	Sets the vertical position of the caption.
	Visible (see page 901)	Determines whether the component appears onscreen.

3.1.31.2.1 TFlowPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it

continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
```

```
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
```

```
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
```

```
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.31.2.2 TFlowPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 1976)

Caption

3.1.31.2.3 TFlowPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the

length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.31.2.4 TFlowPanel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.31.2.5 TFlowPanel.AutoWrap Property

Specifies whether controls are automatically wrapped to fit on the flow panel.

Pascal

```
property AutoWrap: Boolean;
```

C++

```
__property Boolean AutoWrap;
```

Description

If set to True, a new control is automatically placed in a location on the flow panel in which it is visible. If set to False, a new control continues in the same direction as the previous control, even if it would no longer be visible on the panel. You can increase the size of the panel to view controls that are not automatically wrapped to the next line or column.

3.1.31.2.6 TFlowPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.31.2.7 TFlowPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TPanelBevel;
```

C++

```
__property TPanelBevel BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
```

```
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 808)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.31.2.8 TFlowPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.31.2.9 TFlowPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TPanelBevel;
```

C++

```
__property TPanelBevel BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
```

```
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.31.2.10 TFlowPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
```

```
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

BorderWidth (see page 811)

3.1.31.2.11 TFlowPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency,

ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.31.2.12 TFlowPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 811)

3.1.31.2.13 TFlowPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 807)

BevelInner (see page 807)

BevelOuter (see page 808)

BorderStyle (see page 810)

Caption

3.1.31.2.14 TFlowPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of

character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());
```

```
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

3.1.31.2.15 TFlowPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
```

```
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.31.2.16 TFlowPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.31.2.17 TFlowPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.31.2.18 TFlowPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
```

3.1.31.2.19 TFlowPanel.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.31.2.20 TFlowPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.31.2.21 TFlowPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.31.2.22 TFlowPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.31.2.23 TFlowPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.31.2.24 TFlowPanel.FlowStyle Property

Specifies the starting point and direction for controls added to a flow panel.

Pascal

```
property FlowStyle: TFlowStyle;
```

C++

```
__property TFlowStyle FlowStyle;
```

Description

Use FlowStyle to set the starting position for the first control added to a flow panel, as well as the subsequent position for additional controls.

Flow Style	Description
BottomTopLeftRight	The first control is at the bottom left corner of the panel. Subsequent controls are added from bottom to top and then from left to right.
BottomTopRightLeft	The first control is at the bottom right corner of the panel. Subsequent controls are added from bottom to top and then from right to left.
LeftRightBottomTop	The first control is at the bottom left corner of the panel. Subsequent controls are added from left to right and then from bottom to top.
LeftRightTopBottom	The first control is at the top left corner of the panel. Subsequent controls are added from left to right and then from top to bottom. This is the default setting.
RightLeftBottomTop	The first control is at the bottom right corner of the panel. Subsequent controls are added from right to left and then from bottom to top.
RightLeftTopBottom	The first control is at the top right corner of the panel. Subsequent controls are added from right to left and then from top to bottom.
TopBottomLeftRight	The first control is at the top left corner of the panel. Subsequent controls are added from top to bottom and then from left to right.
TopBottomRightLeft	The first control is at the top right corner of the panel. Subsequent controls are added from top to bottom and then from right to left.

C++ Examples:

```
//
//This example demonstrates the use of Flow panel to automatically
//align a set of controls and the different methods that can be used for that.
//A Stack is used to control the order of destruction of buttons.
//
void __fastcall TForm2::btAButtonClick(TObject *Sender)
{
    TButton *thisButton;
    TButton *lastButton;
    String temp;

    /* Sender is a TButton, cast it */
    thisButton = dynamic_cast<TButton*>(Sender);

    if (m_Stack->Count() > 0)
    {
        /* If there are buttons in stack, peek it */
        lastButton = dynamic_cast<TButton*>(m_Stack->Peek());

        /* Change the captions of this button and the last one */
        temp = thisButton->Caption;
        thisButton->Caption = lastButton->Caption;
        lastButton->Caption = temp;
    }
}

void __fastcall TForm2::btAddButtonClick(TObject *Sender)
{
    /* Create a new button */
    TButton *button = new TButton(m_FlowPane);

    button->Parent = m_FlowPane;
    button->Caption = IntToStr(m_Stack->Count());
    button->OnClick = btAButtonClick;

    /* Push the button to the stack */
    m_Stack->Push(button);
}
```

```
void __fastcall TForm2::btReArrangeClick(TObject *Sender)
{
    /* Change the current flow style (rotate them) */
    TFlowStyle newFlowStyle = m_FlowPane->FlowStyle;

    if (newFlowStyle == fsBottomTopRightLeft)
        newFlowStyle = fsLeftRightTopBottom;
    else
        newFlowStyle = newFlowStyle + 1;

    m_FlowPane->FlowStyle = newFlowStyle;
}

void __fastcall TForm2::btRemoveButtonClick(TObject *Sender)
{
    TButton *button;

    if (m_Stack->Count() > 0)
    {
        /* if the stack is not empty, pop the last button */
        button = dynamic_cast<TButton*>(m_Stack->Pop());
        delete button;
    }
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Create a new TFlowPanel pane and align it to client */
    m_FlowPane = new TFlowPanel(this);
    m_FlowPane->Parent = this;
    m_FlowPane->Align = alClient;

    /* Set the initial flowing style */
    m_FlowPane->FlowStyle = fsLeftRightTopBottom;

    /* Create the button stack */
    m_Stack = new TObjectStack();
}

void __fastcall TForm2::FormDestroy(TObject *Sender)
{
    /* Destroy the stack also */
    delete m_Stack;
}
```

Delphi Examples:

```
{
This example demonstrates the use of Flow panel to automatically
align a set of controls and the different methods that can be used for that.
A Stack is used to control the order of destruction of buttons.
}
procedure TForm2.btAButtonClick(Sender: TObject);
var
  ThisButton : TButton;
  LastButton : TButton;
  Temp       : String;
begin
  { Sender is a TButton, cast it }
  ThisButton := Sender as TButton;

  if FStack.Count > 0 then
  begin
    { If there are buttons in stack, peek it }
    LastButton := FStack.Peek() as TButton;
```

```
    { Change the captions of this button and the last one }
    Temp := ThisButton.Caption;
    ThisButton.Caption := LastButton.Caption;
    LastButton.Caption := Temp;
  end;
end;

procedure TForm2.btAddButtonClick(Sender: TObject);
var
  Button : TButton;
begin
  { Create a new button }
  Button := TButton.Create(FFlowPane);
  Button.Parent := FFlowPane;
  Button.Caption := IntToStr(FStack.Count);
  Button.OnClick := btAButtonClick;

  { Push the button to the stack }
  FStack.Push(Button);
end;

procedure TForm2.btReArrangeClick(Sender: TObject);
var
  NewFlowStyle : TFlowStyle;
begin
  { Chnage the flow order (rotate) }
  NewFlowStyle := FFlowPane.FlowStyle;

  if NewFlowStyle = fsBottomTopRightLeft then
     NewFlowStyle := fsLeftRightTopBottom
  else
    Inc(NewFlowStyle);

  FFlowPane.FlowStyle := NewFlowStyle;
end;

procedure TForm2.btRemoveButtonClick(Sender: TObject);
var
  Button : TButton;
begin
  if FStack.Count > 0 then
  begin
    { if the stack is not empty, pop the last button }
    Button := FStack.Pop() as TButton;
    Button.Free();
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Create a new TFlowPanel pane and align it to client }
  FFlowPane := TFlowPanel.Create(Self);
  FFlowPane.Parent := Self;
  FFlowPane.Align := alClient;

  { Set the initial flowing style }
  FFlowPane.FlowStyle := fsLeftRightTopBottom;

  { Create the button stack }
  FStack := TObjectStack.Create();
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
  { Destroy the stack also }
  FStack.Free();
```

```
end;
```

3.1.31.2.25 TFlowPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.31.2.26 TFlowPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate

3.1.31.2.27 TFlowPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align

TOleContainer (see page 5969)

3.1.31.2.28 TFlowPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

3.1.31.2.29 TFlowPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.31.2.30 TFlowPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.31.2.31 TFlowPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.31.2.32 TFlowPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.31.2.33 TFlowPanel.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.31.2.34 TFlowPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.31.2.35 TFlowPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.31.2.36 TFlowPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.31.2.37 TFlowPanel.ShowCaption Property

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

This is ShowCaption, a member of class TFlowPanel.

3.1.31.2.38 TFlowPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.31.2.39 TFlowPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.31.2.40 TFlowPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.31.2.41 TFlowPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager;
```

C++

```
__property UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager

3.1.31.2.42 TFlowPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

3.1.31.2.43 TFlowPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.31.3 TFlowPanel Events

The events of the TFlowPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 903)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 904)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 905)	Occurs when an attempt is made to resize the control.
	OnClick (see page 905)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 907)	Adjust resize constraints.
	OnContextPopup (see page 907)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 908)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 908)	Occurs when another control is docked to the control.
	OnDockOver (see page 910)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 913)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 916)	Occurs when the user drags an object over a control.
	OnEndDock (see page 919)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 919)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 920)	Occurs when a control receives the input focus.
	OnExit (see page 923)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 925)	Returns the control's docking information.
	OnMouseActivate (see page 928)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 929)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 931)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 931)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 931)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 934)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 936)	Occurs immediately after the control is resized.
	OnStartDock (see page 937)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 937)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 938)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.31.3.1 TFlowPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

3.1.31.3.2 TFlowPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

3.1.31.3.3 TFlowPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

3.1.31.3.4 TFlowPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.31.3.5 TFlowPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

3.1.31.3.6 TFlowPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.31.3.7 TFlowPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.31.3.8 TFlowPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

3.1.31.3.9 TFlowPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage(L"GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.31.3.10 TFlowPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.31.3.11 TFlowPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.31.3.12 TFlowPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.31.3.13 TFlowPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.31.3.14 TFlowPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.31.3.15 TFlowPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.31.3.16 TFlowPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
  property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.31.3.17 TFlowPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.31.3.18 TFlowPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.31.3.19 TFlowPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.31.3.20 TFlowPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.31.3.21 TFlowPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.31.3.22 TFlowPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.31.3.23 TFlowPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

3.1.31.3.24 TFlowPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.31.3.25 TFlowPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.31.3.26 TFlowPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
```

```
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
  {
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

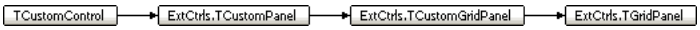
OnDockDrop

OnDockOver

3.1.32 TGridPanel Class

TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.

Class Hierarchy



Pascal

```
TGridPanel = class(TCustomGridPanel);
```

C++

```
class TGridPanel : public TCustomGridPanel;
```

File

ExtCtrls

Description

Use TGridPanel to put an empty grid panel on a form. The major difference between a traditional panel and a grid panel is the way in which controls are placed. With a traditional panel, you place a control (such as a button) in a specific location. You can freely move that control to any location within the panel using the mouse. In a grid panel, you specify the number of rows and columns on the panel, then each component is placed into the next available cell in the grid, regardless of where you place it with the mouse. If each grid contains a control and you add more controls, the location for the new controls is determined by the ExpandStyle (see page 968) property. For example, using the default ExpandStyle (see page 968) property of AddRows, a new row is added to the grid, and each cell in the row can contain a new control.

See Also

TPanel (see page 1326)

3.1.32.1 TGridPanel Members

The following tables list the members exposed by TGridPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AlignControls (see page 784)	Aligns any controls for which the control is the parent within a specified area of the control.
	AutoAddColumn (see page 785)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 785)	This is AutoAddRow, a member of class TCustomGridPanel.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CellIndexToCell (see page 785)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 785)	This is CellToCellIndex, a member of class TCustomGridPanel.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Loaded (see page 786)	Initializes the control after it is loaded from a stream.
	Paint (see page 787)	Renders the image of the panel.
	Paint (see page 806)	Renders the image of the panel.
	RemoveEmptyAutoAddColumns (see page 787)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 787)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.

	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.
	UpdateControlOriginalParentSize (see page 788)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

Public Methods

	Name	Description
	Create (see page 783)	Creates and initializes an instance of TCustomPanel.
	Create (see page 803)	Creates and initializes an instance of TCustomPanel (see page 802).
	Destroy (see page 783)	Destroys an instance of TCustomControl.
	GetChildren (see page 785)	Calls a specified method for each child of the control.
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.
	IsColumnEmpty (see page 786)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsRowEmpty (see page 786)	This is IsRowEmpty, a member of class TCustomGridPanel.

Public Properties

	Name	Description
	CellCount (see page 788)	This is CellCount, a member of class TCustomGridPanel.
	CellRect (see page 788)	Returns the screen coordinates of a cell in the grid.
	CellSize (see page 789)	This is CellSize, a member of class TCustomGridPanel.
	ColumnSpanIndex (see page 789)	This is ColumnSpanIndex, a member of class TCustomGridPanel.
	DockManager (see page 960)	Specifies the control's docking manager interface.
	RowSpanIndex (see page 790)	This is RowSpanIndex, a member of class TCustomGridPanel.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 984)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 984)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 985)	Occurs when an attempt is made to resize the control.
	OnClick (see page 986)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 987)	Adjust resize constraints.
	OnContextPopup (see page 988)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 989)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 989)	Occurs when another control is docked to the control.
	OnDockOver (see page 991)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 993)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 996)	Occurs when the user drags an object over a control.
	OnEndDock (see page 999)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1000)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1001)	Occurs when a control receives the input focus.
	OnExit (see page 1003)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 1006)	Returns the control's docking information.
	OnMouseActivate (see page 1008)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1009)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1011)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1011)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1012)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1014)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 1016)	Occurs immediately after the control is resized.
	OnStartDock (see page 1017)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnStartDrag (see page 1018)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 1018)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 945)	Determines how the control aligns within its container (parent control).
	Alignment (see page 948)	Determines how the caption is aligned within the panel.
	Anchors (see page 949)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 949)	Specifies which edges of the control are beveled.
	BevelInner (see page 950)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 951)	Specifies the control's bevel style.
	BevelOuter (see page 951)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 952)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 953)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 954)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 954)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 955)	Specifies a text string that identifies the control to the user.
	Color (see page 958)	Specifies the background color of the control.
	ColumnCollection (see page 959)	Sets the number and size of the columns on the grid panel.
	Constraints (see page 959)	Specifies the size constraints for the control.
	ControlCollection (see page 959)	This is ControlCollection, a member of class TGridPanel.
	Ctl3D (see page 960)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 962)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 963)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 965)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 965)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 966)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 967)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandStyle (see page 968)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	Font (see page 968)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 969)	Determines how the panel repaints itself when it is resized.
	Locked (see page 969)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 970)	Specifies the padding of a control.
	ParentBackground (see page 970)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 970)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 971)	Determines where a control looks for its color information.
	ParentCtl3D (see page 971)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 972)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 972)	Determines where a control looks for its font information.
	ParentShowHint (see page 972)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 973)	Identifies the pop-up menu associated with the control.
	RowCollection (see page 978)	Sets the number and size of the rows on the grid panel.
	ShowCaption (see page 978)	This is ShowCaption, a member of class TGridPanel.
	ShowHint (see page 978)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 980)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 980)	Determines if the user can tab to a control.
	UseDockManager (see page 981)	Specifies whether the docking manager is used in drag-and-dock operations.

	VerticalAlignment (see page 981)	Sets the vertical position of the caption.
	Visible (see page 982)	Determines whether the component appears onscreen.

3.1.32.2 TGridPanel Properties

The properties of the TGridPanel class are listed here.

Public Properties

	Name	Description
	DockManager (see page 960)	Specifies the control's docking manager interface.

Published Properties

	Name	Description
	Align (see page 945)	Determines how the control aligns within its container (parent control).
	Alignment (see page 948)	Determines how the caption is aligned within the panel.
	Anchors (see page 949)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 949)	Specifies which edges of the control are beveled.
	BevelInner (see page 950)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 951)	Specifies the control's bevel style.
	BevelOuter (see page 951)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 952)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 953)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 954)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 954)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 955)	Specifies a text string that identifies the control to the user.
	Color (see page 958)	Specifies the background color of the control.
	ColumnCollection (see page 959)	Sets the number and size of the columns on the grid panel.
	Constraints (see page 959)	Specifies the size constraints for the control.
	ControlCollection (see page 959)	This is ControlCollection, a member of class TGridPanel.
	Ctl3D (see page 960)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 962)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 963)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 965)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 965)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 966)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 967)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandStyle (see page 968)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	Font (see page 968)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 969)	Determines how the panel repaints itself when it is resized.
	Locked (see page 969)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 970)	Specifies the padding of a control.
	ParentBackground (see page 970)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 970)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 971)	Determines where a control looks for its color information.
	ParentCtl3D (see page 971)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 972)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 972)	Determines where a control looks for its font information.
	ParentShowHint (see page 972)	Determines where a control looks to find out if its Help Hint should be shown.

	PopupMenu (see page 973)	Identifies the pop-up menu associated with the control.
	RowCollection (see page 978)	Sets the number and size of the rows on the grid panel.
	ShowCaption (see page 978)	This is ShowCaption, a member of class TGridPanel.
	ShowHint (see page 978)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 980)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 980)	Determines if the user can tab to a control.
	UseDockManager (see page 981)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 981)	Sets the vertical position of the caption.
	Visible (see page 982)	Determines whether the component appears onscreen.

3.1.32.2.1 TGridPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
```

```
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
```

```
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
```

```
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.32.2.2 TGridPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 1976)

Caption

3.1.32.2.3 TGridPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.32.2.4 TGridPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.32.2.5 TGridPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TPanelBevel;
```

C++

```
__property TPanelBevel BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 808)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.32.2.6 TGridPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.32.2.7 TGridPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TPanelBevel;
```

C++

```
__property TPanelBevel BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.32.2.8 TGridPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
```

```
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

BorderWidth (see page 811)

3.1.32.2.9 TGridPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.32.2.10 TGridPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 811)

3.1.32.2.11 TGridPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
```

```
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 807)

BevelInner (see page 807)

BevelOuter (see page 808)

BorderStyle (see page 810)

Caption

3.1.32.2.12 TGridPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
```

```
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

3.1.32.2.13 TGridPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.32.2.14 TGridPanel.ColumnCollection Property

Sets the number and size of the columns on the grid panel.

Pascal

```
property ColumnCollection: TColumnCollection;
```

C++

```
__property TColumnCollection ColumnCollection;
```

Description

Use the ColumnCollection property to set the number of columns as well as the size of each column. You can use the SizeStyle property to specify whether column sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

3.1.32.2.15 TGridPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.32.2.16 TGridPanel.ControlCollection Property

Pascal

```
property ControlCollection: TControlCollection;
```

C++

```
__property TControlCollection ControlCollection;
```

Description

This is ControlCollection, a member of class TGridPanel.

3.1.32.2.17 TGridPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.32.2.18 TGridPanel.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager;
```

C++

```
__property DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/
```

```
void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;
```

```
procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

UseDockManager

DefaultDockTreeClass

3.1.32.2.19 TGridPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
```

```
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.32.2.20 TGridPanel.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.32.2.21 TGridPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.32.2.22 TGridPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.32.2.23 TGridPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.32.2.24 TGridPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.32.2.25 TGridPanel.ExpandStyle Property

Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.

Pascal

```
property ExpandStyle: TExpandStyle;
```

C++

```
__property TExpandStyle ExpandStyle;
```

Description

Use the ExpandStyle property to specify how a grid panel is expanded when each cell of the grid contains a control, and a new control is added.

3.1.32.2.26 TGridPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.32.2.27 TGridPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate

3.1.32.2.28 TGridPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being

replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align

TOleContainer (see page 5969)

3.1.32.2.29 TGridPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

3.1.32.2.30 TGridPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.32.2.31 TGridPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern

locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.32.2.32 TGridPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.32.2.33 TGridPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.32.2.34 TGridPanel.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.32.2.35 TGridPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.32.2.36 TGridPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.32.2.37 TGridPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.32.2.38 TGridPanel.RowCollection Property

Sets the number and size of the rows on the grid panel.

Pascal

```
property RowCollection: TRowCollection;
```

C++

```
__property TRowCollection RowCollection;
```

Description

Use the RowCollection property to set the number of rows as well as the size of each row. You can use the SizeStyle property to specify whether row sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

3.1.32.2.39 TGridPanel.ShowCaption Property

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

This is ShowCaption, a member of class TGridPanel.

3.1.32.2.40 TGridPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.32.2.41 TGridPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.32.2.42 TGridPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.32.2.43 TGridPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager;
```

C++

```
__property UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager

3.1.32.2.44 TGridPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.

taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

3.1.32.2.45 TGridPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
```

```
  form OnCreate event handler populates each sheet with an
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.32.3 TGridPanel Events

The events of the TGridPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 984)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 984)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 985)	Occurs when an attempt is made to resize the control.
	OnClick (see page 986)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 987)	Adjust resize constraints.
	OnContextPopup (see page 988)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 989)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 989)	Occurs when another control is docked to the control.
	OnDockOver (see page 991)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 993)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 996)	Occurs when the user drags an object over a control.
	OnEndDock (see page 999)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1000)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1001)	Occurs when a control receives the input focus.
	OnExit (see page 1003)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 1006)	Returns the control's docking information.
	OnMouseActivate (see page 1008)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1009)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1011)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1011)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1012)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1014)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 1016)	Occurs immediately after the control is resized.
	OnStartDock (see page 1017)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1018)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 1018)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.32.3.1 TGridPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

3.1.32.3.2 TGridPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

3.1.32.3.3 TGridPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

3.1.32.3.4 TGridPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.32.3.5 TGridPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

3.1.32.3.6 TGridPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.32.3.7 TGridPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.32.3.8 TGridPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
```

```
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
```

```
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockOver

3.1.32.3.9 TGridPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
```

```
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
```

```
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.32.3.10 TGridPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
```

```
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
```

```
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.32.3.11 TGridPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
```

```
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.32.3.12 TGridPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.32.3.13 TGridPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.32.3.14 TGridPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
```

```
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
```

```
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.32.3.15 TGridPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.32.3.16 TGridPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.32.3.17 TGridPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.32.3.18 TGridPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.32.3.19 TGridPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.32.3.20 TGridPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.32.3.21 TGridPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.32.3.22 TGridPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.32.3.23 TGridPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
```

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

3.1.32.3.24 TGridPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.32.3.25 TGridPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.32.3.26 TGridPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
```

```
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
```

```
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

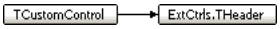
OnDockDrop

OnDockOver

3.1.33 THeader Class

THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.

Class Hierarchy



Pascal

```
THeader = class(TCustomControl);
```

C++

```
class THeader : public TCustomControl;
```

File

ExtCtrls

Description

THeader is provided for backward compatibility. New applications should use THeaderControl instead.

3.1.33.1 THeader Members

The following tables list the members exposed by THeader.

Protected Methods

	Name	Description
	CreateParams (see page 1023)	Initializes a window-creation parameter data structure.
	MouseDown (see page 1024)	OnMouseDown event dispatcher.
	MouseMove (see page 1024)	Respond to mouse moving over control area..
	MouseUp (see page 1025)	OnMouseUp event dispatcher.
	Paint (see page 1025)	Renders the image of a custom control.
	Sized (see page 1026)	This is Sized, a member of class THeader.
	Sizing (see page 1026)	This is Sizing, a member of class THeader.

Public Methods

	Name	Description
	Create (see page 1022)	Creates a new THeader (see page 1021) object.
	Destroy (see page 1023)	Frees the memory associated with the THeader (see page 1021) object.

Public Properties

	Name	Description
	SectionWidth (see page 1042)	Determines the width, in pixels, of the sections of a header.

Published Events

	Name	Description
	OnContextPopup (see page 1046)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnSized (see page 1047)	Occurs after the header has been resized.
	OnSizing (see page 1047)	Occurs before the header is resized.

Published Properties

	Name	Description
	Align (see page 1027)	Determines how the control aligns within its container (parent control).
	AllowResize (see page 1030)	Determines if the user can modify the size of the header at runtime with the mouse.
	Anchors (see page 1030)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 1031)	Determines whether the header control has a border.
	Constraints (see page 1031)	Specifies the size constraints for the control.
	DoubleBuffered (see page 1032)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 1034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1034)	Controls the attributes of text written on or in the control.
	ParentDoubleBuffered (see page 1035)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1036)	Determines where a control looks for its font information.
	ParentShowHint (see page 1036)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1037)	Identifies the pop-up menu associated with the control.
	Sections (see page 1042)	Lists the strings that contain the text for the sections of a header.
	ShowHint (see page 1042)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1044)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1044)	Determines if the user can tab to a control.
	Visible (see page 1045)	Determines whether the component appears onscreen.

3.1.33.2 THeader.Create Constructor

Creates a new THeader (see page 1021) object.

Pascal

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

C++

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

Description

Call Create to instantiate a header at runtime. Headers added to forms at design time are created automatically.

The AOwner parameter initializes the Owner property of the header.

See Also

Owner

3.1.33.3 THeader.Destroy Destructor

Frees the memory associated with the THeader (see page 1021) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THeader();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the THeader (see page 1021) reference is nil before calling Destroy.

See Also

System::TObject::Free

3.1.33.4 THeader Methods

The methods of the THeader class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 1023)	Initializes a window-creation parameter data structure.
	MouseDown (see page 1024)	OnMouseDown event dispatcher.
	MouseMove (see page 1024)	Respond to mouse moving over control area..
	MouseUp (see page 1025)	OnMouseUp event dispatcher.
	Paint (see page 1025)	Renders the image of a custom control.
	Sized (see page 1026)	This is Sized, a member of class THeader.
	Sizing (see page 1026)	This is Sizing, a member of class THeader.

3.1.33.4.1 THeader.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call

to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

3.1.33.4.2 THeader.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

3.1.33.4.3 THeader.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

3.1.33.4.4 THeader.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

3.1.33.4.5 THeader.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

3.1.33.4.6 THeader.Sized Method

Pascal

```
procedure Sized(ASection: Integer; AWidth: Integer); dynamic;
```

C++

```
__fastcall Sized(int ASection, int AWidth);
```

Description

This is Sized, a member of class THeader.

3.1.33.4.7 THeader.Sizing Method

Pascal

```
procedure Sizing(ASection: Integer; AWidth: Integer); dynamic;
```

C++

```
__fastcall Sizing(int ASection, int AWidth);
```

Description

This is Sizing, a member of class THeader.

3.1.33.5 THeader Properties

The properties of the THeader class are listed here.

Public Properties

	Name	Description
	SectionWidth (see page 1042)	Determines the width, in pixels, of the sections of a header.

Published Properties

	Name	Description
	Align (see page 1027)	Determines how the control aligns within its container (parent control).
	AllowResize (see page 1030)	Determines if the user can modify the size of the header at runtime with the mouse.
	Anchors (see page 1030)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 1031)	Determines whether the header control has a border.
	Constraints (see page 1031)	Specifies the size constraints for the control.
	DoubleBuffered (see page 1032)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 1034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1034)	Controls the attributes of text written on or in the control.
	ParentDoubleBuffered (see page 1035)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

	ParentFont (see page 1036)	Determines where a control looks for its font information.
	ParentShowHint (see page 1036)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1037)	Identifies the pop-up menu associated with the control.
	Sections (see page 1042)	Lists the strings that contain the text for the sections of a header.
	ShowHint (see page 1042)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1044)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1044)	Determines if the user can tab to a control.
	Visible (see page 1045)	Determines whether the component appears onscreen.

3.1.33.5.1 THeader.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
```

```
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );
```

```
procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.33.5.2 THeader.AllowResize Property

Determines if the user can modify the size of the header at runtime with the mouse.

Pascal

```
property AllowResize: Boolean;
```

C++

```
__property Boolean AllowResize;
```

Description

If AllowResize is false, the sections within a header can't be resized. If AllowResize is true, clicking a border of a header section and dragging it left or right changes the width of the section. The default value is true.

3.1.33.5.3 THeader.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.33.5.4 THeader.BorderStyle Property

Determines whether the header control has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

3.1.33.5.5 THeader.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.33.5.6 THeader.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
```

```
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.33.5.7 THeader.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.33.5.8 THeader.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
```

```
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    for (int i = 0; i < Screen->Fonts->Count; i++)
     ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
  //  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
  }

  void __fastcall TForm1::ComboBox1Change(TObject *Sender)
  {
    RichEdit1->Font->Name = ComboBox1->Text;
  }
```

Delphi Examples:

```
  {
  To use this example, place a TComboBox and TRichEdit or
  TMemo on a form.  During form creation, a list of font names
  is loaded into the combo box. When the combo box is clicked,
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.33.5.9 THeader.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.33.5.10 THeader.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.33.5.11 THeader.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.33.5.12 THeader.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.33.5.13 THeader.Sections Property

Lists the strings that contain the text for the sections of a header.

Pascal

```
property Sections: TStrings;
```

C++

```
__property TStrings Sections;
```

Description

The number of lines of the string list determines the number of sections of the header. If the string list is empty, the header has one blank section. If this string list contains one or more lines, the text of each line is in its own section. The first line is in the leftmost section, the second line is in the next section to the right, and so on.

3.1.33.5.14 THeader.SectionWidth Property

Determines the width, in pixels, of the sections of a header.

Pascal

```
property SectionWidth [X: Integer]: Integer;
```

C++

```
__property int SectionWidth[int X];
```

Description

Use SectionWidth to specify the width of each header section. X is an index into the sections, from 0 to the number of sections - 1. For example, the index of the first section is 0, the second section is 1, and so on.

3.1.33.5.15 THeader.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.33.5.16 THeader.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.33.5.17 THeader.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.33.5.18 THeader.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.33.6 THeader Events

The events of the THeader class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 1046)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnSized (see page 1047)	Occurs after the header has been resized.
	OnSizing (see page 1047)	Occurs before the header is resized.

3.1.33.6.1 THeader.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.33.6.2 THeader.OnSized Event

Occurs after the header has been resized.

Pascal

```
property OnSized: TSectionEvent;
```

C++

```
__property TSectionEvent OnSized;
```

Description

Write an OnSized event handler to perform special processing after the header has been resized. In this event, the AWidth parameter is the width of the section before it was resized. Use the SectionWidth (see page 1042) property to determine the new size of the section.

See Also

OnSizing (see page 1047)

SectionWidth (see page 1042)

3.1.33.6.3 THeader.OnSizing Event

Occurs before the header is resized.

Pascal

```
property OnSizing: TSectionEvent;
```

C++

```
__property TSectionEvent OnSizing;
```

Description

Write an OnSizing event handler to perform special processing immediately before the header is resized. In this event, AWidth is the width that the section will have after it is resized. Use the SectionWidth (see page 1042) property to determine the current size of the section.

See Also

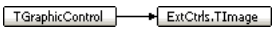
OnSized (see page 1047)

SectionWidth (see page 1042)

3.1.34 TImage Class

TImage displays a graphical image.

Class Hierarchy



Pascal

```
TImage = class(TGraphicControl);
```

C++

```
class TImage : public TGraphicControl;
```

File

ExtCtrls

Description

Use TImage to display a graphical image on a form. Use the TPicture (see page 2812) object in the Picture (see page 1066) property to specify the actual bitmap, con, metafile, or other graphic object displayed by TImage. Properties and methods of TPicture (see page 2812) can be used for such things as loading an image from file, clearing the image in the TImage, and assigning an image for another control. TImage introduces several properties to determine how the image is displayed within the boundaries of the TImage object.

To add an image to a form or data module so that it is available for display by other controls, such as action lists and bitmap buttons, use a TImageList control instead.

A TImage may hold a bitmap, icon, PNG, GIF or JPEG image. A TImageList may hold any image type a TImage holds.

See Also

TImageList

TPicture (see page 2812)

TBitmap (see page 2628)

TIcon (see page 2768)

TMetafile (see page 2780)

Adding Graphical Objects to a String List

Adding Images to an Application

Graphic Controls

Images

3.1.34.1 TImage Members

The following tables list the members exposed by TImage.

Protected Methods

	Name	Description
	CanAutoSize (see page 1051)	Indicates whether the control can resize itself automatically to accommodate its contents.
	DestRect (see page 1052)	This is DestRect, a member of class TImage.
	DoPaletteChange (see page 1052)	This is DoPaletteChange, a member of class TImage.
	GetPalette (see page 1052)	Returns the handle to a palette for use by the control.
	Paint (see page 1052)	Render control surface.
	Progress (see page 1053)	This is Progress, a member of class TImage.

Public Methods

	Name	Description
	Create (see page 1050)	Creates and initializes an instance of TImage (see page 1048).
	Destroy (see page 1050)	Destroys an instance of TImage (see page 1048).

Public Properties

	Name	Description
	Canvas (see page 1058)	Provides a drawing surface for embellishing bitmap images.

Published Events

	Name	Description
	OnClick (see page 1079)	Occurs when the user clicks the control.
	OnContextPopup (see page 1081)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1082)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1082)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1089)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1090)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1091)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1093)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1093)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1093)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1096)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProgress (see page 1098)	Occurs periodically during slow operations that affect the image.
	OnStartDock (see page 1098)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1099)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1054)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1057)	Specifies how the control is anchored to its parent.

	AutoSize (see page 1058)	Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.
	Center (see page 1061)	Indicates whether the image is centered in the image control.
	Constraints (see page 1062)	Specifies the size constraints for the control.
	DragCursor (see page 1062)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1063)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1063)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1065)	Controls whether the control responds to mouse, keyboard, and timer events.
	IncrementalDisplay (see page 1065)	Specifies whether successive approximations of the image should be drawn during slow operations.
	ParentShowHint (see page 1066)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 1066)	Specifies the image that appears on the image control.
	PopupMenu (see page 1068)	Identifies the pop-up menu associated with the control.
	Proportional (see page 1073)	Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.
	ShowHint (see page 1073)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 1075)	Indicates whether the image should be changed so that it exactly fits the bounds of the image control.
	Transparent (see page 1077)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 1077)	Determines whether the component appears onscreen.

3.1.34.2 TImage.Create Constructor

Creates and initializes an instance of TImage (see page 1048).

Pascal

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

C++

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

Description

Call Create to instantiate a TImage (see page 1048) object at runtime. Images placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the TImage (see page 1048) instance. It becomes the value of the Owner property.

See Also

Owner

3.1.34.3 TImage.Destroy Destructor

Destroys an instance of TImage (see page 1048).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TImage (see page 1048) reference is not nil, and only then calls Destroy.

As the control is destroyed, it frees the TPicture (see page 2812) object in the Picture (see page 1066) property.

See Also

System::TObject::Free

3.1.34.4 TImage Methods

The methods of the TImage class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 1051)	Indicates whether the control can resize itself automatically to accommodate its contents.
	DestRect (see page 1052)	This is DestRect, a member of class TImage.
	DoPaletteChange (see page 1052)	This is DoPaletteChange, a member of class TImage.
	GetPalette (see page 1052)	Returns the handle to a palette for use by the control.
	Paint (see page 1052)	Render control surface.
	Progress (see page 1053)	This is Progress, a member of class TImage.

3.1.34.4.1 TImage.CanAutoSize Method

Indicates whether the control can resize itself automatically to accommodate its contents.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns true, and the resize sequence is aborted.

As implemented in TControl, CanAutoSize always returns true without changing NewWidth or NewHeight. Override CanAutoSize to implement the AutoSize property for a particular control class's contents.

See Also

AutoSize

OnResize

OnCanResize

OnConstrainedResize

Width

Height

3.1.34.4.2 TImage.DestRect Method

Pascal

```
function DestRect: TRect;
```

C++

```
__fastcall TRect DestRect();
```

Description

This is DestRect, a member of class TImage.

3.1.34.4.3 TImage.DoPaletteChange Method

Pascal

```
function DoPaletteChange: Boolean;
```

C++

```
__fastcall Boolean DoPaletteChange();
```

Description

This is DoPaletteChange, a member of class TImage.

3.1.34.4.4 TImage.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged

3.1.34.4.5 TImage.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

3.1.34.4.6 TImage.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const UnicodeString Msg);
```

Description

This is Progress, a member of class TImage.

3.1.34.5 TImage Properties

The properties of the TImage class are listed here.

Public Properties

	Name	Description
R	Canvas (see page 1058)	Provides a drawing surface for embellishing bitmap images.

Published Properties

	Name	Description
	Align (see page 1054)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1057)	Specifies how the control is anchored to its parent.
	AutoSize (see page 1058)	Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.
	Center (see page 1061)	Indicates whether the image is centered in the image control.
	Constraints (see page 1062)	Specifies the size constraints for the control.
	DragCursor (see page 1062)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1063)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1063)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1065)	Controls whether the control responds to mouse, keyboard, and timer events.
	IncrementalDisplay (see page 1065)	Specifies whether successive approximations of the image should be drawn during slow operations.
	ParentShowHint (see page 1066)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 1066)	Specifies the image that appears on the image control.

	PopupMenu (see page 1068)	Identifies the pop-up menu associated with the control.
	Proportional (see page 1073)	Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.
	ShowHint (see page 1073)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 1075)	Indicates whether the image should be changed so that it exactly fits the bounds of the image control.
	Transparent (see page 1077)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 1077)	Determines whether the component appears onscreen.

3.1.34.5.1 TImage.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
```

```
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );
```

```
procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.34.5.2 TImage.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.34.5.3 TImage.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the TImage (see page 1048) control adjusts its own dimensions to match the dimensions of the image that the Picture (see page 1066) property specifies. When AutoSize is true, the TImage (see page 1048) control forces its own height and width to match those of the picture it represents. When AutoSize is false (the default), the Center (see page 1061) and Stretch (see page 1075) properties determine how the image control deals with discrepancies between the size of the picture and the size of the image control.

See Also

Constraints

Picture (see page 1066)

Width (see page 2829)

Height (see page 2828)

Height

Width

Center (see page 1061)

Stretch (see page 1075)

3.1.34.5.4 TImage.Canvas Property

Provides a drawing surface for embellishing bitmap images.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas is the TCanvas (see page 2673) object on which the image for the TImage (see page 1048) component is drawn. The painting of the image specified in the Picture (see page 1066) property is automatic and does not require manual intercession for the picture to appear.

You can use properties and methods of TCanvas (see page 2673) to do such things as draw lines, draw images, and write text to the TImage (see page 1048) control's canvas.

Note: Canvas is only available if the Picture (see page 1066) property represents a TBitmap (see page 2628) object. Trying to read Canvas when Picture (see page 1066) represents another type of graphic image causes TImage (see page 1048) to raise an EInvalidOperation exception.

C++ Examples:

```
/*
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    HRGN MyRgn;

    MyRgn = ::CreateRectRgn(100,100,200,200);
    ::SelectClipRgn(Image1->Canvas->Handle,MyRgn);
    Image1->Canvas->Brush->Color = clRed;
    Image1->Canvas->FillRect(Image1->Canvas->ClipRect);
    Image1->Invalidate();
    ::SelectClipRgn(Image1->Canvas->Handle,NULL);
    ::DeleteObject(MyRgn);
}
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    std::auto_ptr<TIcon> pi(new Graphics::TIcon);
    AnsiString as;
    pi->LoadFromFile(OpenDialog1->FileName);
    as = ChangeFileExt(OpenDialog1->FileName,".BMP");
    Image1->Width = pi->Width;
    Image1->Height = pi->Height;
    Image1->Canvas->Draw(0, 0, pi.get());
    Image1->Picture->SaveToFile(as);
    ShowMessage(OpenDialog1->FileName + " Saved to " + as);
  }
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
```

```
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TImage *myImage;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TImage")
  {
    myImage = dynamic_cast<TImage *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myImage->Canvas->MoveTo(myImage->ClientRect.Left, myImage->ClientRect.Top);
    myImage->Canvas->LineTo(myImage->ClientRect.Right, myImage->ClientRect.Bottom);
    myImage = NULL;
  }
}
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
    MyRgn: HRGN;
begin
    MyRgn := CreateRectRgn(100,100,200,200);
    SelectClipRgn(Image1.Canvas.Handle,MyRgn);
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
    Image1.Invalidate;
    SelectClipRgn(Image1.Canvas.Handle, HRGN(nil));
    DeleteObject(MyRgn);
end;
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
```

```
      try
        Icon.Loadfromfile(OpenDialog1.FileName);
        s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
        Image1.Width := Icon.Width;
        Image1.Height := Icon.Height;
        Image1.Canvas.Draw(0,0,Icon);
        Image1.Picture.SaveToFile(s);
        ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
      finally
        Icon.Free;
      end;
    end;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TImage) then
    begin
      myImage:= (ListBox1.Items.Objects[Index] as TImage);
      with myImage.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myImage.Canvas.MoveTo(Left, Top);
        myImage.Canvas.LineTo(Right, Bottom);
      end;
    myImage:= nil;
    end;
  {
  The following example paints a cross-hatched ellipse onto
  Image1 which is a TImage on the form when the user clicks on
  Button1.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    with Image1 do begin
      Canvas.Brush.Color := clRed;
      Canvas.Brush.Style := bsDiagCross;
      Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
    end;
  end;
```

See Also

EInvalidOperation

Picture (see page 1066)

TCanvas (see page 2673)

3.1.34.5.5 TImage.Center Property

Indicates whether the image is centered in the image control.

Pascal

```
property Center: Boolean;
```

C++

```
__property Boolean Center;
```

Description

When the image does not fit perfectly within the image control, use Center to specify how the image is positioned. When Center

is true, the image is centered in the control. When Center is false, the upper left corner of the image is positioned at the upper left corner of the control.

The default value for Center is false.

Note: Center has no effect if the AutoSize (see page 1058) property is true or if the Stretch (see page 1075) property is true and Picture (see page 1066) does not specify an icon.

See Also

AutoSize

Picture (see page 1066)

Stretch (see page 1075)

Proportional (see page 1073)

3.1.34.5.6 TImage.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.34.5.7 TImage.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.34.5.8 TImage.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.34.5.9 TImage.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
```

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.34.5.10 TImage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.34.5.11 TImage.IncrementalDisplay Property

Specifies whether successive approximations of the image should be drawn during slow operations.

Pascal

```
property IncrementalDisplay: Boolean;
```

C++

```
__property Boolean IncrementalDisplay;
```

Description

Set IncrementalDisplay to true on large compressed images that take a long time to load or on images that may be altered by a computationally slow process. When IncrementalDisplay is true, partial renderings of the image are drawn periodically during slow operations such as loading. Set IncrementalDisplay to false to wait until the entire operation is complete before drawing the image.

For some classes of graphic objects, an OnProgress (see page 1098) event occurs at various points during slow operations. If IncrementalDisplay is true, the OnProgress (see page 1098) event occurs immediately after the approximate image is drawn. If IncrementalDisplay is false, OnProgress (see page 1098) occurs but the image is not updated until the entire image has been loaded.

The default value for IncrementalDisplay is false.

See Also

OnProgress (see page 1098)

3.1.34.5.12 TImage.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.34.5.13 TImage.Picture Property

Specifies the image that appears on the image control.

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

Use Picture to specify the image for the TImage (see page 1048) component. Picture is a reference to a TPicture (see page 2812) object. Use properties and methods of TPicture (see page 2812) to specify a bitmap, icon, metafile, or user-defined graphic to be displayed by the image control.

Setting Picture at design time brings up the Picture Editor, which can be used to specify the file that contains the image.

C++ Examples:

```
/*
```

```
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

See Also

Graphic (see page 2827)

3.1.34.5.14 TImage.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
```

```
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
```

```
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.34.5.15 TImage.Proportional Property

Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.

Pascal

```
property Proportional: Boolean;
```

C++

```
__property Boolean Proportional;
```

Description

Set Proportional to true to ensure that the image can be fully displayed in the image control without any distortion such as occurs with the Stretch (see page 1075) property. When Proportional is true, images that are too large to fit in the image control are scaled down (while maintaining the same aspect ratio) until they fit in the image control. Images that are too small are displayed normally. That is, Proportional can reduce the magnification of the image, but does not increase it.

When the image control resizes, the image resizes also.

To resize the image so that it fits exactly in the image control, even if that causes distortion, use the Stretch (see page 1075) property instead.

To resize the control to the image rather than resizing the image to the control, use the AutoSize (see page 1058) property instead.

The default value for Proportional is false.

Note: Proportional has no effect if the Picture (see page 1066) property contains an icon.

See Also

AutoSize (see page 1058)

Picture (see page 1066)

Stretch (see page 1075)

3.1.34.5.16 TImage.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.34.5.17 TImage.Stretch Property

Indicates whether the image should be changed so that it exactly fits the bounds of the image control.

Pascal

```
property Stretch: Boolean;
```

C++

```
__property Boolean Stretch;
```

Description

Set Stretch to true to cause the image to assume the size and shape of the image control. When the image control resizes, the image resizes also. Stretch resizes the height and width of the image independently. Thus, unlike a simple change in magnification, Stretch can distort the image if the image control is not the same shape as the image.

To resize the image without any distortion, use the Proportional (see page 1073) property instead.

To resize the control to the image rather than resizing the image to the control, use the AutoSize (see page 1058) property instead.

The default value for Stretch is false.

Note: Stretch has no effect if the Picture (see page 1066) property contains an icon.

C++ Examples:

```
/*
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ImageList1->GetBitmap(0, Image1->Picture->Bitmap);
```

```
  Image1->Stretch = True;
}
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
*/

#include <Jpeg.hpp>
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
  std::auto_ptr<TJPEGImage> jp(new TJPEGImage());
  jp->Assign(Image1->Picture->Bitmap);
  jp->SaveToFile("..\\factory.jpg");
  Image2->Stretch = True;
  Image2->Picture->LoadFromFile("..\\factory.jpg");
}
```

Delphi Examples:

```
{
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   ImageList1.GetBitmap(2,Image1.Picture.Bitmap);
   Image1.Stretch := True;
end;
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

See Also

AutoSize

Picture (see page 1066)

Proportional (see page 1073)

3.1.34.5.18 TImage.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TImage (see page 1048) object to show through the background of the image. Set Transparent to false to make the background of the image opaque. Setting Transparent sets the Transparent property of the Picture (see page 1066).

Note: Transparent has no effect unless the Picture (see page 1066) property specifies a TBitmap (see page 2628) object.

See Also

Picture (see page 1066)

Transparent (see page 2762)

Graphic (see page 2827)

3.1.34.5.19 TImage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.34.6 TImage Events

The events of the TImage class are listed here.

Published Events

	Name	Description
	OnClick (see page 1079)	Occurs when the user clicks the control.
	OnContextPopup (see page 1081)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1082)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1082)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1089)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1090)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1091)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1093)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1093)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1093)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1096)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProgress (see page 1098)	Occurs periodically during slow operations that affect the image.
	OnStartDock (see page 1098)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1099)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.34.6.1 TImage.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check

box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.34.6.2 TImage.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.34.6.3 TImage.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.34.6.4 TImage.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
```

```
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
```

```
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.34.6.5 TImage.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
```

```
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.34.6.6 TImage.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.34.6.7 TImage.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.34.6.8 TImage.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.34.6.9 TImage.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.34.6.10 TImage.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.34.6.11 TImage.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.34.6.12 TImage.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.34.6.13 TImage.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.34.6.14 TImage.OnProgress Event

Occurs periodically during slow operations that affect the image.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

Write an OnProgress event handler to provide the user with feedback during slow operations such as loading large compressed images.

Note: The PercentDone parameter of the event handler is only an approximation. With some image formats, the value of PercentDone may actually decrease from the value in previous events, as the graphic object discovers there is more work to do.

See Also

IncrementalDisplay (see page 1065)

OnProgress (see page 2830)

TProgressBar

3.1.34.6.15 TImage.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the

appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.34.6.16 TImage.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.35 TLabeledEdit Class

TLabeledEdit is an edit control that has an associated label.

Class Hierarchy

Pascal

```
TLabeledEdit = class(TCustomLabeledEdit);
```

C++

```
class TLabeledEdit : public TCustomLabeledEdit;
```

File

ExtCtrls

Description

Use TLabeledEdit to add an edit control with a label to a form. TLabeledEdit works with an associated label control, which is the value of its EditLabel (see page 1115) property. The position of that label relative to the edit control is determined by the LabelPosition (see page 1118) and LabelSpacing (see page 1118) properties.

Use the Text (see page 1131) property to work with the text in the control, and the label's Caption property to work with the text in the label.

See Also

TEdit

TBoundLabel (see page 174)

TMaskEdit (see page 4610)

3.1.35.1 TLabeledEdit Members

The following tables list the members exposed by TLabeledEdit.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 792)	Responds when the BiDiMode property changes.
	CMEnabledchanged (see page 793)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 793)	Responds when the Visible property changes.
	Notification (see page 793)	Responds to notifications when objects are created or destroyed.
	SetName (see page 794)	Sets the name of the control.
	SetParent (see page 795)	Sets the value of the Parent property.

Public Methods

	Name	Description
	Create (see page 791)	Creates an instance of TCustomLabeledEdit (see page 790).
	SetBounds (see page 794)	Sets the windowed control's boundary properties all at once.
	SetupInternalLabel (see page 795)	Initializes the control that implements the EditLabel (see page 796) property.

Published Events

	Name	Description
	OnChange (see page 1134)	Occurs when the text for the edit control may have changed.
	OnClick (see page 1135)	Occurs when the user clicks the control.
	OnContextPopup (see page 1137)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1137)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1138)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1141)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1144)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1144)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1145)	Occurs when a control receives the input focus.
	OnExit (see page 1148)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1151)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1154)	Occurs when a key is pressed.
	OnKeyUp (see page 1155)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1157)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1158)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1160)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1160)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1161)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1163)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1165)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1166)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Alignment (see page 1103)	Determines how the text is aligned within the text edit control.
	Anchors (see page 1104)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 1104)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 1105)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 1106)	Specifies which edges of the control are beveled.
	BevelInner (see page 1106)	Specifies the cut of the inner bevel.
	BevelKind (see page 1107)	Specifies the control's bevel style.
	BevelOuter (see page 1107)	Specifies the cut of the outer bevel.
	BiDiMode (see page 1107)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 1108)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 1108)	Determines the case of the text within the edit control.
	Color (see page 1109)	Specifies the background color of the control.
	Constraints (see page 1109)	Specifies the size constraints for the control.
	Ctl3D (see page 1110)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1110)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1112)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1113)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 1113)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditLabel (see page 1115)	Specifies the control that implements the label.
	Enabled (see page 1115)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1115)	Controls the attributes of text written on or in the control.
	HideSelection (see page 1116)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 1117)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1117)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LabelPosition (see page 1118)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 1118)	Specifies the distance, in pixels, between the label and the edit region.
	MaxLength (see page 1119)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 1119)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 1120)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 1120)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1120)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1121)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1121)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1121)	Determines where a control looks for its font information.
	ParentShowHint (see page 1122)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 1123)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 1123)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 1128)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 1128)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1130)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1130)	Determines if the user can tab to a control.
	Text (see page 1131)	Contains a text string associated with the control.
	TextHint (see page 1132)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 1132)	Determines whether the component appears onscreen.

3.1.35.2 TLabeledEdit Properties

The properties of the TLabeledEdit class are listed here.

Published Properties

	Name	Description
	Alignment (see page 1103)	Determines how the text is aligned within the text edit control.
	Anchors (see page 1104)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 1104)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 1105)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 1106)	Specifies which edges of the control are beveled.
	BevelInner (see page 1106)	Specifies the cut of the inner bevel.
	BevelKind (see page 1107)	Specifies the control's bevel style.
	BevelOuter (see page 1107)	Specifies the cut of the outer bevel.
	BiDiMode (see page 1107)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 1108)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 1108)	Determines the case of the text within the edit control.
	Color (see page 1109)	Specifies the background color of the control.

	Constraints (see page 1109)	Specifies the size constraints for the control.
	Ctl3D (see page 1110)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1110)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1112)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1113)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1113)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditLabel (see page 1115)	Specifies the control that implements the label.
	Enabled (see page 1115)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1115)	Controls the attributes of text written on or in the control.
	HideSelection (see page 1116)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 1117)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1117)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LabelPosition (see page 1118)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 1118)	Specifies the distance, in pixels, between the label and the edit region.
	MaxLength (see page 1119)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 1119)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 1120)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 1120)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1120)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1121)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1121)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1121)	Determines where a control looks for its font information.
	ParentShowHint (see page 1122)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 1123)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 1123)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 1128)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 1128)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1130)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1130)	Determines if the user can tab to a control.
	Text (see page 1131)	Contains a text string associated with the control.
	TextHint (see page 1132)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 1132)	Determines whether the component appears onscreen.

3.1.35.2.1 TLabeledEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment;
```

C++

```
__property Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment

Text

Displaying and Editing Fields in an Edit Box

3.1.35.2.2 TLabeledEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.35.2.3 TLabeledEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect;
```

C++

```
__property AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll

SelLength

SelStart

SelText

SetSelTextBuf

3.1.35.2.4 TLabeledEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle

ClientHeight

ControlStyle

Height

3.1.35.2.5 TLabeledEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.35.2.6 TLabeledEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.35.2.7 TLabeledEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.35.2.8 TLabeledEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.35.2.9 TLabeledEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.35.2.10 TLabeledEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

3.1.35.2.11 TLabeledEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase;
```

C++

```
__property CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

3.1.35.2.12 TLabeledEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.35.2.13 TLabeledEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.35.2.14 TLabeledEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.35.2.15 TLabeledEdit.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.35.2.16 TLabeledEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.35.2.17 TLabeledEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.35.2.18 TLabeledEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.35.2.19 TLabeledEdit.EditLabel Property

Specifies the control that implements the label.

Pascal

```
property EditLabel: TBoundLabel;
```

C++

```
__property TBoundLabel EditLabel;
```

Description

Use EditLabel to work with the label that is associated with this labeled edit control. Use this label's properties to specify the caption that appears on the label, the position of the caption within the label, the font used to write that caption, and so on. At design time, expand the EditLabel property in the Object Inspector to set its properties.

See Also

LabelPosition (see page 796)

LabelSpacing (see page 796)

3.1.35.2.20 TLabeledEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.35.2.21 TLabeledEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.35.2.22 TLabeledEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection;
```

C++

```
__property HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength

SelStart

SelText

SetSelTextBuf

3.1.35.2.23 TLabeledEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

3.1.35.2.24 TLabeledEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

3.1.35.2.25 TLabeledEdit.LabelPosition Property

Specifies the position of the label relative to the edit control.

Pascal

```
property LabelPosition: TLabelPosition;
```

C++

```
__property TLabelPosition LabelPosition;
```

Description

Set LabelPosition to indicate where the label sits relative to the edit control.

See Also

EditLabel (see page 796)

LabelSpacing (see page 796)

3.1.35.2.26 TLabeledEdit.LabelSpacing Property

Specifies the distance, in pixels, between the label and the edit region.

Pascal

```
property LabelSpacing: Integer;
```

C++

```
__property int LabelSpacing;
```

Description

Set LabelSpacing to indicate the distance, in pixels, between the label and the edit control.

See Also

EditLabel (see page 796)

LabelPosition (see page 796)

3.1.35.2.27 TLabeledEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength;
```

C++

```
__property MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

3.1.35.2.28 TLabeledEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly;
```

C++

```
__property NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

3.1.35.2.29 TLabeledEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert;
```

C++

```
__property OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

3.1.35.2.30 TLabeledEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.35.2.31 TLabeledEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.35.2.32 TLabeledEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.35.2.33 TLabeledEdit.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.35.2.34 TLabeledEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.35.2.35 TLabeledEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.35.2.36 TLabeledEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar;
```

C++

```
__property PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

3.1.35.2.37 TLabeledEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
```

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
```

```
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
  makes the pop-up menu available to both edit boxes:
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.35.2.38 TLabeledEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard

GetSelTextBuf

3.1.35.2.39 TLabeledEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
```

```
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.35.2.40 TLabeledEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.35.2.41 TLabeledEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.35.2.42 TLabeledEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines

3.1.35.2.43 TLabeledEdit.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint;
```

C++

```
__property TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

3.1.35.2.44 TLabeledEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.35.3 TLabeledEdit Events

The events of the TLabeledEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 1134)	Occurs when the text for the edit control may have changed.
	OnClick (see page 1135)	Occurs when the user clicks the control.
	OnContextPopup (see page 1137)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1137)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1138)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1141)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1144)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1144)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1145)	Occurs when a control receives the input focus.
	OnExit (see page 1148)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1151)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1154)	Occurs when a key is pressed.
	OnKeyUp (see page 1155)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1157)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1158)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1160)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1160)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1161)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1163)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1165)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1166)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.35.3.1 TLabeledEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change

Text

3.1.35.3.2 TLabeledEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.35.3.3 TLabeledEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.35.3.4 TLabeledEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.35.3.5 TLabeledEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.35.3.6 TLabeledEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.35.3.7 TLabeledEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.35.3.8 TLabeledEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.35.3.9 TLabeledEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
  property OnEnter;
```

C++

```
  __property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.35.3.10 TLabeledEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));
```

```
    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.35.3.11 TLabeledEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

3.1.35.3.12 TLabeledEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

3.1.35.3.13 TLabeledEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

3.1.35.3.14 TLabeledEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control.

However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.35.3.15 TLabeledEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
```

```
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
```

```
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.35.3.16 TLabeledEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.35.3.17 TLabeledEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.35.3.18 TLabeledEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.35.3.19 TLabeledEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.35.3.20 TLabeledEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.35.3.21 TLabeledEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.36 TLinkLabel Class

Label containing an HTML link.

Class Hierarchy

Pascal

```
TLinkLabel = class(TCustomLinkLabel);
```

C++

```
class TLinkLabel : public TCustomLinkLabel;
```

File

ExtCtrls

Description

TLinkLabel is a TCustomLinkLabel (see page 797) that can contain HTML tags, including a link. The link appears as underlined text. The OnLinkClick (see page 1202) event is triggered by clicking the link.

See Also

TCustomLinkLabel (see page 797)

OnLinkClick (see page 802)

TSysLinkEvent (see page 1555)

TSysLinkType (see page 1546)

3.1.36.1 TLinkLabel Members

The following tables list the members exposed by TLinkLabel.

Protected Methods

	Name	Description
	CreateParams (see page 799)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 799)	Creates the underlying window.
	SetAutoSize (see page 800)	This is SetAutoSize, a member of class TCustomLinkLabel.
	UseThemes (see page 800)	This is UseThemes, a member of class TCustomLinkLabel.

Public Methods

	Name	Description
	Create (see page 798)	Creates an instance of TWinControl.

Public Nested Types

Name	Description
TLinkAlignment (see page 801)	This is nested type ExtCtrls.TCustomLinkLabel.TLinkAlignment.

Published Events

	Name	Description
	OnClick (see page 1191)	Occurs when the user clicks the control.

	OnContextPopup (see page 1193)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1194)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1194)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1197)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1200)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1201)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnLinkClick (see page 1202)	Event for click on TCustomLinkLabel HTML link.
	OnMouseActivate (see page 1202)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1203)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1205)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1205)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1206)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1208)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1210)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1211)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Alignment (see page 1169)	This is Alignment, a member of class TLinkLabel.
	Anchors (see page 1169)	Specifies how the control is anchored to its parent.
	AutoSize (see page 1170)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 1170)	Specifies which edges of the control are beveled.
	BevelInner (see page 1171)	Specifies the cut of the inner bevel.
	BevelKind (see page 1171)	Specifies the control's bevel style.
	BevelOuter (see page 1172)	Specifies the cut of the outer bevel.
	Caption (see page 1172)	Specifies a text string that identifies the control to the user.
	Color (see page 1175)	Specifies the background color of the control.
	Constraints (see page 1175)	Specifies the size constraints for the control.
	DragCursor (see page 1176)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1176)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1177)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1178)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1179)	Controls the attributes of text written on or in the control.
	ParentColor (see page 1180)	Determines where a control looks for its color information.
	ParentFont (see page 1180)	Determines where a control looks for its font information.
	ParentShowHint (see page 1181)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1181)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1186)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1188)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1188)	Determines if the user can tab to a control.
	UseVisualStyle (see page 1189)	This is UseVisualStyle, a member of class TLinkLabel.
	Visible (see page 1189)	Determines whether the component appears onscreen.

3.1.36.2 TLinkLabel Properties

The properties of the TLinkLabel class are listed here.

Published Properties

	Name	Description
	Alignment (see page 1169)	This is Alignment, a member of class TLinkLabel.
	Anchors (see page 1169)	Specifies how the control is anchored to its parent.
	AutoSize (see page 1170)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 1170)	Specifies which edges of the control are beveled.
	BevelInner (see page 1171)	Specifies the cut of the inner bevel.
	BevelKind (see page 1171)	Specifies the control's bevel style.
	BevelOuter (see page 1172)	Specifies the cut of the outer bevel.
	Caption (see page 1172)	Specifies a text string that identifies the control to the user.
	Color (see page 1175)	Specifies the background color of the control.
	Constraints (see page 1175)	Specifies the size constraints for the control.
	DragCursor (see page 1176)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1176)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1177)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1178)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1179)	Controls the attributes of text written on or in the control.
	ParentColor (see page 1180)	Determines where a control looks for its color information.
	ParentFont (see page 1180)	Determines where a control looks for its font information.
	ParentShowHint (see page 1181)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1181)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1186)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1188)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1188)	Determines if the user can tab to a control.
	UseVisualStyle (see page 1189)	This is UseVisualStyle, a member of class TLinkLabel.
	Visible (see page 1189)	Determines whether the component appears onscreen.

3.1.36.2.1 TLinkLabel.Alignment Property

Pascal

```
property Alignment: TLinkAlignment;
```

C++

```
__property TLinkAlignment Alignment;
```

Description

This is Alignment, a member of class TLinkLabel.

3.1.36.2.2 TLinkLabel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.36.2.3 TLinkLabel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.36.2.4 TLinkLabel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.36.2.5 TLinkLabel.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

3.1.36.2.6 TLinkLabel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.36.2.7 TLinkLabel.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

3.1.36.2.8 TLinkLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
```

```
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
```

```
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long to be the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
  This example creates a ToolBar and adds ToolButtons to it.
  Include ComCtlrs in the uses clause.
  }
  procedure AddButtons(ToolBar: TToolBar;
    Const ButtonCaptions: array of String);
  var
    I: Integer;
  begin
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
```

```
  ToolBar1.Free;
end;
```

3.1.36.2.9 TLinkLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.36.2.10 TLinkLabel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.36.2.11 TLinkLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.36.2.12 TLinkLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.36.2.13 TLinkLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.36.2.14 TLinkLabel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls

appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.36.2.15 TLinkLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.36.2.16 TLinkLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.36.2.17 TLinkLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform

appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.36.2.18 TLinkLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.36.2.19 TLinkLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse

button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
```

```
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
```

```
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.36.2.20 TLinkLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine

whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
```

```
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.36.2.21 TLinkLabel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.36.2.22 TLinkLabel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.36.2.23 TLinkLabel.UseVisualStyle Property

Pascal

```
property UseVisualStyle: Boolean;
```

C++

```
__property Boolean UseVisualStyle;
```

Description

This is UseVisualStyle, a member of class TLinkLabel.

3.1.36.2.24 TLinkLabel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.36.3 TLinkLabel Events

The events of the TLinkLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 1191)	Occurs when the user clicks the control.
	OnContextPopup (see page 1193)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1194)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1194)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1197)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1200)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1201)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnLinkClick (see page 1202)	Event for click on TCustomLinkLabel HTML link.
	OnMouseActivate (see page 1202)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1203)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1205)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1205)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1206)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1208)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1210)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1211)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.36.3.1 TLinkLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check

box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.36.3.2 TLinkLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.36.3.3 TLinkLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.36.3.4 TLinkLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
```

```
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
```

```
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.36.3.5 TLinkLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
```

```
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.36.3.6 TLinkLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.36.3.7 TLinkLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.36.3.8 TLinkLabel.OnLinkClick Event

Event for click on TCustomLinkLabel HTML link.

Pascal

```
property OnLinkClick: TSysLinkEvent;
```

C++

```
__property TSysLinkEvent OnLinkClick;
```

Description

OnLinkClick is triggered by a click on a TCustomLinkLabel's HTML link. The event identifies the link URI and link type.

See Also

TCustomLinkLabel (see page 797)

TSysLinkEvent (see page 1555)

TSysLinkType (see page 1546)

3.1.36.3.9 TLinkLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.36.3.10 TLinkLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.36.3.11 TLinkLabel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.36.3.12 TLinkLabel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.36.3.13 TLinkLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.36.3.14 TLinkLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.36.3.15 TLinkLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.36.3.16 TLinkLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

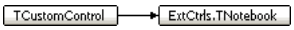
OnDragOver

OnDragDrop

3.1.37 TNotebook Class

TNotebook displays multiple pages, each with its own set of controls.

Class Hierarchy



Pascal

```
TNotebook = class(TCustomControl);
```

C++

```
class TNotebook : public TCustomControl;
```

File

ExtCtrls

Description

Notebook components are frequently used with tab set controls (TTabSet) to let the user select pages in the notebook by clicking a tab. TNotebook is provided for backward compatibility. New applications should use TPageControl instead.

3.1.37.1 TNotebook Members

The following tables list the members exposed by TNotebook.

Protected Methods

	Name	Description
	CreateParams (see page 1214)	Initializes a window-creation parameter data structure.
	GetChildOwner (see page 1215)	Returns the owner of a child component being read from a stream.
	ReadState (see page 1216)	Prepares the control for having its properties assigned values from a stream.
	ShowControl (see page 1216)	Ensures that a specified child control is visible.

Public Methods

	Name	Description
	Create (see page 1213)	Constructs a TNotebook (see page 1211) component.
	Destroy (see page 1214)	Destroys the TNotebook (see page 1211) component.
	GetChildren (see page 1215)	Calls a specified method for each child of the control.

Published Events

	Name	Description
	OnClick (see page 1242)	Occurs when the user clicks the control.
	OnContextPopup (see page 1244)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1244)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1245)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1248)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1251)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1251)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1252)	Occurs when a control receives the input focus.
	OnExit (see page 1255)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 1257)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1258)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1261)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1261)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 1261)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1263)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPageChanged (see page 1266)	Occurs when the user selects a new page in the notebook.
	OnStartDock (see page 1266)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1267)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	ActivePage (see page 1217)	Determines which page displays in the notebook.
	Align (see page 1218)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1221)	Specifies how the control is anchored to its parent.
	Color (see page 1222)	Specifies the background color of the control.
	Constraints (see page 1222)	Specifies the size constraints for the control.
	Ctl3D (see page 1223)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1223)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1225)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1226)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1226)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1228)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1228)	Controls the attributes of text written on or in the control.
	PageIndex (see page 1229)	Determines which page displays in the notebook control.
	Pages (see page 1230)	Contains the strings that identify the individual pages of the notebook control.
	ParentColor (see page 1230)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1230)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1231)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1231)	Determines where a control looks for its font information.
	ParentShowHint (see page 1232)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1232)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1237)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1239)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1239)	Determines if the user can tab to a control.
	Visible (see page 1240)	Determines whether the component appears onscreen.

3.1.37.2 TNotebook.Create Constructor

Constructs a TNotebook (see page 1211) component.

Pascal

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

C++

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

Description

Call Create to instantiate a notebook at runtime. Notebook components placed in forms at design time are created automatically.

The AOwner parameter specifies the component (usually the form) that is responsible for freeing the notebook.

3.1.37.3 TNotebook.Destroy Destructor

Destroys the TNotebook (see page 1211) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNotebook();
```

Description

Do not call Destroy directly in an application. Instead, use Free, which checks that the TNotebook (see page 1211) reference is not nil before calling Destroy.

See Also

System::TObject::Free

3.1.37.4 TNotebook Methods

The methods of the TNotebook class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 1214)	Initializes a window-creation parameter data structure.
	GetChildOwner (see page 1215)	Returns the owner of a child component being read from a stream.
	ReadState (see page 1216)	Prepares the control for having its properties assigned values from a stream.
	ShowControl (see page 1216)	Ensures that a specified child control is visible.

Public Methods

	Name	Description
	GetChildren (see page 1215)	Calls a specified method for each child of the control.

3.1.37.4.1 TNotebook.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard

controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

3.1.37.4.2 TNotebook.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent, GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner of a component is responsible for destroying it.

See Also

Create

Owner

3.1.37.4.3 TNotebook.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

3.1.37.4.4 TNotebook.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

3.1.37.4.5 TNotebook.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl, ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show

Parent

3.1.37.5 TNotebook Properties

The properties of the TNotebook class are listed here.

Published Properties

	Name	Description
	ActivePage (see page 1217)	Determines which page displays in the notebook.
	Align (see page 1218)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1221)	Specifies how the control is anchored to its parent.
	Color (see page 1222)	Specifies the background color of the control.
	Constraints (see page 1222)	Specifies the size constraints for the control.
	Ctl3D (see page 1223)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1223)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1225)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1226)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1226)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1228)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1228)	Controls the attributes of text written on or in the control.
	PageIndex (see page 1229)	Determines which page displays in the notebook control.
	Pages (see page 1230)	Contains the strings that identify the individual pages of the notebook control.
	ParentColor (see page 1230)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1230)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1231)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1231)	Determines where a control looks for its font information.
	ParentShowHint (see page 1232)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1232)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1237)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1239)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1239)	Determines if the user can tab to a control.
	Visible (see page 1240)	Determines whether the component appears onscreen.

3.1.37.5.1 TNotebook.ActivePage Property

Determines which page displays in the notebook.

Pascal

```
property ActivePage: string;
```

C++

```
__property UnicodeString ActivePage;
```

Description

ActivePage determines which page displays in the notebook. The value of ActivePage must be one of the strings contained in the Pages (see page 1230) property.

To get or set the active page by index rather than string, use the PageIndex (see page 1229) property instead.

See Also

PageIndex (see page 1229)

Pages (see page 1230)

3.1.37.5.2 TNotebook.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
```

```
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.37.5.3 TNotebook.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.37.5.4 TNotebook.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.37.5.5 TNotebook.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.37.5.6 TNotebook.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.37.5.7 TNotebook.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.37.5.8 TNotebook.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.37.5.9 TNotebook.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.37.5.10 TNotebook.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.37.5.11 TNotebook.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.37.5.12 TNotebook.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
```

```
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.37.5.13 TNotebook.PageIndex Property

Determines which page displays in the notebook control.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

The value of PageIndex determines which page displays in the notebook. Changing the PageIndex value changes the page in the control.

Each string in the Pages (see page 1230) property is automatically assigned a PageIndex value when the page is created. The first page receives a value of 0, the second has a value of 1, and so on. If you delete a string from the Pages (see page 1230) property, the PageIndex values are reassigned so that the values always begin with 0 and continue to increase without any gaps between values.

To get or set the current page using its string rather than its index, use the ActivePage (see page 1217) property instead.

See Also

ActivePage (see page 1217)

Pages (see page 1230)

3.1.37.5.14 TNotebook.Pages Property

Contains the strings that identify the individual pages of the notebook control.

Pascal

```
property Pages: TStrings;
```

C++

```
__property TStrings Pages;
```

Description

The Pages property contains the strings that identify the individual pages of the notebook control. Both these controls create a separate page for each string in the Pages property. For example, if Pages contains three strings, First, Second, and Third, the control has three separate pages.

You can access the various pages in a notebook or tabbed notebook control with either the ActivePage (see page 1217) or PageIndex (see page 1229) property.

See Also

ActivePage (see page 1217)

PageIndex (see page 1229)

3.1.37.5.15 TNotebook.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.37.5.16 TNotebook.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.37.5.17 TNotebook.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.37.5.18 TNotebook.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.37.5.19 TNotebook.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.37.5.20 TNotebook.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
```

```
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.37.5.21 TNotebook.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.37.5.22 TNotebook.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.37.5.23 TNotebook.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.37.5.24 TNotebook.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.37.6 TNotebook Events

The events of the TNotebook class are listed here.

Published Events

	Name	Description
	OnClick (see page 1242)	Occurs when the user clicks the control.
	OnContextPopup (see page 1244)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1244)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1245)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1248)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1251)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1251)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1252)	Occurs when a control receives the input focus.
	OnExit (see page 1255)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 1257)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1258)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1261)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1261)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1261)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 1263)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPageChanged (see page 1266)	Occurs when the user selects a new page in the notebook.
	OnStartDock (see page 1266)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1267)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.37.6.1 TNotebook.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.37.6.2 TNotebook.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.37.6.3 TNotebook.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.37.6.4 TNotebook.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.37.6.5 TNotebook.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.37.6.6 TNotebook.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.37.6.7 TNotebook.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.37.6.8 TNotebook.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.37.6.9 TNotebook.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.37.6.10 TNotebook.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
  property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.37.6.11 TNotebook.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.37.6.12 TNotebook.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.37.6.13 TNotebook.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.37.6.14 TNotebook.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.37.6.15 TNotebook.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.37.6.16 TNotebook.OnPageChanged Event

Occurs when the user selects a new page in the notebook.

Pascal

```
property OnPageChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPageChanged;
```

Description

Use the OnPageChanged event when you want special processing to occur when the active page changes.

3.1.37.6.17 TNotebook.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.37.6.18 TNotebook.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

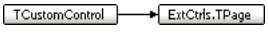
DragKind

OnDragOver

OnDragDrop

3.1.38 TPage Class

Class Hierarchy



Pascal

```
TPage = class(TCustomControl);
```

C++

```
class TPage : public TCustomControl;
```

File

ExtCtrls

Description

This is class ExtCtrls.TPage.

3.1.38.1 TPage Members

The following tables list the members exposed by TPage.

Protected Methods

	Name	Description
	Paint (see page 1269)	Renders the image of a custom control.
	ReadState (see page 1269)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	Create (see page 1268)	Creates an instance of TCustomControl.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 1278)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 1278)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	Caption (see page 1270)	Specifies a text string that identifies the control to the user.
	Height (see page 1273)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 1274)	Indicates the position of the control in its parent's tab order.
	Visible (see page 1274)	Determines whether the component appears onscreen.
	Width (see page 1276)	Specifies the horizontal size of the control or form in pixels.

3.1.38.2 TPage.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

3.1.38.3 TPage Methods

The methods of the TPage class are listed here.

Protected Methods

	Name	Description
	Paint (see page 1269)	Renders the image of a custom control.
	ReadState (see page 1269)	Prepares the control for having its properties assigned values from a stream.

3.1.38.3.1 TPage.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

3.1.38.3.2 TPage.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

3.1.38.4 TPage Properties

The properties of the TPage class are listed here.

Published Properties

	Name	Description
	Caption (see page 1270)	Specifies a text string that identifies the control to the user.
	Height (see page 1273)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 1274)	Indicates the position of the control in its parent's tab order.
	Visible (see page 1274)	Determines whether the component appears onscreen.
	Width (see page 1276)	Specifies the horizontal size of the control or form in pixels.

3.1.38.4.1 TPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
```

```
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
```

```
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long to be the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
  This example creates a ToolBar and adds ToolButtons to it.
  Include ComCtlrs in the uses clause.
  }
  procedure AddButtons(ToolBar: TToolBar;
    Const ButtonCaptions: array of String);
  var
    I: Integer;
  begin
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
```

```
  ToolBar1.Free;
end;
```

3.1.38.4.2 TPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
```

```
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

3.1.38.4.3 TPage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.38.4.4 TPage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.38.4.5 TPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
```

```
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

3.1.38.5 TPage Events

The events of the TPage class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 1278)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 1278)	Occurs when an object with custom alignment is aligned.

3.1.38.5.1 TPage.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

3.1.38.5.2 TPage.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

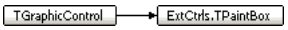
TAlignInfo

TAlignPositionEvent

3.1.39 TPaintBox Class

TPaintBox provides a canvas that applications can use for rendering an image.

Class Hierarchy



Pascal

```
TPaintBox = class(TGraphicControl);
```

C++

```
class TPaintBox : public TGraphicControl;
```

File

ExtCtrls

Description

Use TPaintBox to add custom images to a form. Unlike TImage (see page 1048), which displays an image that is stored in a bitmap, icon, or metafile, TPaintBox requires an application to draw the image directly on a canvas. Use the OnPaint (see page 1322) event handler to draw on the paint box's Canvas (see page 1286), the drawing surface of the paint box.

See Also

TCanvas (see page 2673)

TImage (see page 1048)

Graphic Controls

Paint Boxes

3.1.39.1 TPaintBox Members

The following tables list the members exposed by TPaintBox.

Protected Methods

	Name	Description
	Paint (see page 1281)	Generates an OnPaint (see page 1322) event.

Public Methods

	Name	Description
	Create (see page 1281)	Creates an instance of TPaintBox (see page 1279).

Public Properties

	Name	Description
	Canvas (see page 1286)	Provides the drawing surface used by the graphic control.

Published Events

	Name	Description
	OnClick (see page 1304)	Occurs when the user clicks the control.
	OnContextPopup (see page 1305)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1306)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1306)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1309)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1313)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1313)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1314)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1315)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1317)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1318)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1318)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1320)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 1322)	OnPaint occurs when the paintbox receives a Windows paint message.
	OnStartDock (see page 1324)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1325)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1282)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1286)	Specifies how the control is anchored to its parent.
	Color (see page 1288)	Specifies the background color of the control.
	Constraints (see page 1289)	Specifies the size constraints for the control.
	DragCursor (see page 1290)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1290)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1290)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1292)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1292)	Controls the attributes of text written on or in the control.

	ParentColor (see page 1293)	Determines where a control looks for its color information.
	ParentFont (see page 1294)	Determines where a control looks for its font information.
	ParentShowHint (see page 1294)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1295)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1300)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 1301)	Determines whether the component appears onscreen.

3.1.39.2 TPaintBox.Create Constructor

Creates an instance of TPaintBox (see page 1279).

Pascal

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

C++

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

Description

Call Create to instantiate a TPaintBox (see page 1279) object at runtime. TPaintBox (see page 1279) objects added to forms at design time are created automatically.

Call Create to instantiate a TImage (see page 1048) object at runtime. Images placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the paintbox instance. It becomes the value of the Owner property.

See Also

Owner

3.1.39.3 TPaintBox Methods

The methods of the TPaintBox class are listed here.

Protected Methods

	Name	Description
	Paint (see page 1281)	Generates an OnPaint (see page 1322) event.

3.1.39.3.1 TPaintBox.Paint Method

Generates an OnPaint (see page 1322) event.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint responds to WM_PAINT (see page 5198) messages from Windows by initializing the canvas and calling the OnPaint (see page 1322) event handler if it is assigned.

See Also

OnPaint (see page 1322)

3.1.39.4 TPaintBox Properties

The properties of the TPaintBox class are listed here.

Public Properties

	Name	Description
	Canvas (see page 1286)	Provides the drawing surface used by the graphic control.

Published Properties

	Name	Description
	Align (see page 1282)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1286)	Specifies how the control is anchored to its parent.
	Color (see page 1288)	Specifies the background color of the control.
	Constraints (see page 1289)	Specifies the size constraints for the control.
	DragCursor (see page 1290)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1290)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1290)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1292)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1292)	Controls the attributes of text written on or in the control.
	ParentColor (see page 1293)	Determines where a control looks for its color information.
	ParentFont (see page 1294)	Determines where a control looks for its font information.
	ParentShowHint (see page 1294)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1295)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1300)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 1301)	Determines whether the component appears onscreen.

3.1.39.4.1 TPaintBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.39.4.2 TPaintBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.39.4.3 TPaintBox.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
```

```
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = dynamic_cast<TLabel *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
```

```
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

3.1.39.4.4 TPaintBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.39.4.5 TPaintBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.39.4.6 TPaintBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.39.4.7 TPaintBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.39.4.8 TPaintBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
```

```
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.39.4.9 TPaintBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.39.4.10 TPaintBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.39.4.11 TPaintBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.39.4.12 TPaintBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.39.4.13 TPaintBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.39.4.14 TPaintBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
```

```
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  */
  void __fastcall TForm1::Copy1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
```

```
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
```

```
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.39.4.15 TPaintBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.39.4.16 TPaintBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.39.5 TPaintBox Events

The events of the TPaintBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 1304)	Occurs when the user clicks the control.
	OnContextPopup (see page 1305)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1306)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1306)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1309)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1313)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1313)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1314)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1315)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1317)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1318)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1318)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1320)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 1322)	OnPaint occurs when the paintbox receives a Windows paint message.
	OnStartDock (see page 1324)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1325)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.39.5.1 TPaintBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.39.5.2 TPaintBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.39.5.3 TPaintBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.39.5.4 TPaintBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.39.5.5 TPaintBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.39.5.6 TPaintBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.39.5.7 TPaintBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.39.5.8 TPaintBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level

form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.39.5.9 TPaintBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.39.5.10 TPaintBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.39.5.11 TPaintBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.39.5.12 TPaintBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.39.5.13 TPaintBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.39.5.14 TPaintBox.OnPaint Event

OnPaint occurs when the paintbox receives a Windows paint message.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to draw the image of the paint box. The OnPaint event fires when the system repaints the TPaintBox (see page 1279) control. If there is a handler for this event, its code is executed when the OnPaint event fires.

Use properties and methods of the canvas (such as LineTo, Draw, and TextOut) to programmatically construct the image that the paintbox presents on a form. This can consist of a number of operations including drawing lines, images, and shapes. A TPaintBox (see page 1279) itself has no visible manifestation, so if an OnPaint event handler is not provided or no drawing is done in the event handler, the paintbox can not be seen on the form at runtime.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = dynamic_cast<TPaintBox *>(Sender);
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
```

```
  the Button's OnClick event handler below, this drawing will
  not be persistent and will be "erased" the first time the
  PaintBox component needs to repaint.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     PaintBox1.Canvas.Brush.Color := clRed;
     PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
     PaintBox1.Canvas.Brush.Color := clBlue;
     PaintBox1.Canvas.Ellipse(0,0,100,50);
  end;
  {
  In order for the drawing to be persistent, you need to also
  include an OnPaint event handler that tells the PaintBox how
  to redraw itself when it needs to. In the code below, an
  ellipse will be drawn everytime the PaintBox renders itself,
  but the PaintBox will only paint the client area of the
  control red after the Button is clicked. This client area
  will only stay red until it is invalidated, but the ellipse
  will persist. This behavior is different from the TImage
  component, because the TImage component maintains an
  internal bitmap that stores this drawing information for
  you.
  }
  procedure TForm1.PaintBox1Paint(Sender: TObject);
  begin
     PaintBox1.Canvas.Brush.Color := clBlue;
     PaintBox1.Canvas.Ellipse(0,0,100,50);
  end;
  {
  This example paints a white five-pointed star in a paintbox
  control:
  }
  procedure TForm1.PaintBox1Paint(Sender: TObject);
  begin
    with Sender as TPaintBox do
    begin
      Canvas.Pen.Color := clWhite;
      Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
      Point(10, 30), Point(60, 60), Point(40, 10)]);
    end;
  end;
```

See Also

Canvas

TCanvas (see page 2673)

3.1.39.5.15 TPaintBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the

appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.39.5.16 TPaintBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.40 TPanel Class

TPanel implements a generic panel control.

Class Hierarchy

Pascal

```
TPanel = class(TCustomPanel);
```

C++

```
class TPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TPanel to put an empty panel on a form. Panels have properties for providing a beveled border around the control, as well as methods to help manage the placement of child controls embedded in the panel.

You can also use panels to group controls together, similar to the way you can use a group box, but with a beveled border (or no border) rather than the group box outline. Panels are typically used for groups of controls within a single form. If you intend to use the same grouping in other forms, you may want to use a frame instead.

Panels with no borders are useful as docking sites when writing applications that use drag-and-dock.

Although you can use a panel to implement a status bar or tool bar, it is recommended that you use the TToolBar and TStatusBar classes instead.

See Also

TToolBar

TStatusBar

TGroupBox

Grouping Controls

Panels

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using a Panel Component

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Designing Toolbars and Cool Bars

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

3.1.40.1 TPanel Members

The following tables list the members exposed by TPanel.

Protected Methods

	Name	Description
	AdjustClientRect (see page 804)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	CanAutoSize (see page 804)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 805)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 806)	Renders the image of the panel.
	SetParentBackground (see page 806)	This is SetParentBackground, a member of class TCustomPanel.

Public Methods

	Name	Description
	Create (see page 803)	Creates and initializes an instance of TCustomPanel (see page 802).
	GetControlsAlignment (see page 805)	Returns the alignment of text in the panel.

Public Properties

	Name	Description
	DockManager (see page 1345)	Specifies the control's docking manager interface.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 1368)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 1369)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 1370)	Occurs when an attempt is made to resize the control.
	OnClick (see page 1370)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 1372)	Adjust resize constraints.
	OnContextPopup (see page 1372)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1373)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 1373)	Occurs when another control is docked to the control.
	OnDockOver (see page 1375)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 1378)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1381)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1384)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1384)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 1385)	Occurs when a control receives the input focus.
	OnExit (see page 1388)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 1390)	Returns the control's docking information.
	OnMouseActivate (see page 1393)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1394)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1396)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1396)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1396)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1399)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 1401)	Occurs immediately after the control is resized.
	OnStartDock (see page 1402)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1402)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 1403)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 1330)	Determines how the control aligns within its container (parent control).
	Alignment (see page 1333)	Determines how the caption is aligned within the panel.
	Anchors (see page 1334)	Specifies how the control is anchored to its parent.
	AutoSize (see page 1334)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 1335)	Specifies which edges of the control are beveled.
	BevelInner (see page 1335)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 1336)	Specifies the control's bevel style.
	BevelOuter (see page 1336)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 1337)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 1339)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 1339)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 1339)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 1341)	Specifies a text string that identifies the control to the user.
	Color (see page 1343)	Specifies the background color of the control.
	Constraints (see page 1344)	Specifies the size constraints for the control.
	Ctl3D (see page 1345)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 1347)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 1348)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1350)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1350)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1351)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1352)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1353)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 1354)	Determines how the panel repaints itself when it is resized.
	Locked (see page 1354)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 1354)	Specifies the padding of a control.
	ParentBackground (see page 1355)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 1355)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1355)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1356)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 1356)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1356)	Determines where a control looks for its font information.
	ParentShowHint (see page 1357)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1358)	Identifies the pop-up menu associated with the control.
	ShowCaption (see page 1363)	This is ShowCaption, a member of class TPanel.
	ShowHint (see page 1363)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1364)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1365)	Determines if the user can tab to a control.
	UseDockManager (see page 1365)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 1366)	Sets the vertical position of the caption.
	Visible (see page 1366)	Determines whether the component appears onscreen.

3.1.40.2 TPanel Properties

The properties of the TPanel class are listed here.

Public Properties

	Name	Description
	DockManager (see page 1345)	Specifies the control's docking manager interface.

Published Properties

	Name	Description
	Align (see page 1330)	Determines how the control aligns within its container (parent control).
	Alignment (see page 1333)	Determines how the caption is aligned within the panel.
	Anchors (see page 1334)	Specifies how the control is anchored to its parent.
	AutoSize (see page 1334)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 1335)	Specifies which edges of the control are beveled.
	BevelInner (see page 1335)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 1336)	Specifies the control's bevel style.
	BevelOuter (see page 1336)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 1337)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 1339)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 1339)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 1339)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 1341)	Specifies a text string that identifies the control to the user.
	Color (see page 1343)	Specifies the background color of the control.
	Constraints (see page 1344)	Specifies the size constraints for the control.
	Ctl3D (see page 1345)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 1347)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 1348)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1350)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1350)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1351)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1352)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1353)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 1354)	Determines how the panel repaints itself when it is resized.
	Locked (see page 1354)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 1354)	Specifies the padding of a control.

	ParentBackground (see page 1355)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 1355)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1355)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1356)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1356)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1356)	Determines where a control looks for its font information.
	ParentShowHint (see page 1357)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1358)	Identifies the pop-up menu associated with the control.
	ShowCaption (see page 1363)	This is ShowCaption, a member of class TPanel.
	ShowHint (see page 1363)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1364)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1365)	Determines if the user can tab to a control.
	UseDockManager (see page 1365)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 1366)	Sets the vertical position of the caption.
	Visible (see page 1366)	Determines whether the component appears onscreen.

3.1.40.2.1 TPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
```

```
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
```

```
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
```

```
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.40.2.2 TPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to

use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 1976)

Caption

3.1.40.2.3 TPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.40.2.4 TPanel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

3.1.40.2.5 TPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

3.1.40.2.6 TPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TPanelBevel;
```

C++

```
__property TPanelBevel BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
```

```
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 808)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.40.2.7 TPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

3.1.40.2.8 TPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TPanelBevel;
```

C++

```
__property TPanelBevel BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelWidth (see page 809)

BorderWidth (see page 811)

3.1.40.2.9 TPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 807)

BevelOuter (see page 808)

BorderWidth (see page 811)

3.1.40.2.10 TPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.40.2.11 TPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 811)

3.1.40.2.12 TPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 807)

BevelInner (see page 807)

BevelOuter (see page 808)

BorderStyle (see page 810)

Caption

3.1.40.2.13 TPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>
```

```
void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
```

```
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

3.1.40.2.14 TPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
```

```
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.40.2.15 TPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.40.2.16 TPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.40.2.17 TPanel.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager;
```

C++

```
__property DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
```

```
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

UseDockManager

DefaultDockTreeClass

3.1.40.2.18 TPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
```

```
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

3.1.40.2.19 TPanel.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.40.2.20 TPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.40.2.21 TPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.40.2.22 TPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.40.2.23 TPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.40.2.24 TPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.40.2.25 TPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate

3.1.40.2.26 TPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align

TOleContainer (see page 5969)

3.1.40.2.27 TPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

3.1.40.2.28 TPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.40.2.29 TPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.40.2.30 TPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.40.2.31 TPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.40.2.32 TPanel.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.40.2.33 TPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.40.2.34 TPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.40.2.35 TPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.40.2.36 TPanel.ShowCaption Property

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

This is ShowCaption, a member of class TPanel.

3.1.40.2.37 TPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.40.2.38 TPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.40.2.39 TPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.40.2.40 TPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager;
```

C++

```
__property UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager

3.1.40.2.41 TPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

3.1.40.2.42 TPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.40.3 TPanel Events

The events of the TPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 1368)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 1369)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 1370)	Occurs when an attempt is made to resize the control.
	OnClick (see page 1370)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 1372)	Adjust resize constraints.
	OnContextPopup (see page 1372)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1373)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 1373)	Occurs when another control is docked to the control.
	OnDockOver (see page 1375)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 1378)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1381)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1384)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1384)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1385)	Occurs when a control receives the input focus.
	OnExit (see page 1388)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 1390)	Returns the control's docking information.
	OnMouseActivate (see page 1393)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1394)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1396)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1396)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1396)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1399)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 1401)	Occurs immediately after the control is resized.
	OnStartDock (see page 1402)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1402)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 1403)	Occurs when the application tries to undock a control that is docked to the windowed control.

3.1.40.3.1 TPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

3.1.40.3.2 TPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

3.1.40.3.3 TPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

3.1.40.3.4 TPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.40.3.5 TPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

3.1.40.3.6 TPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.40.3.7 TPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

3.1.40.3.8 TPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

3.1.40.3.9 TPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage(L"GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

3.1.40.3.10 TPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.40.3.11 TPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.40.3.12 TPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.40.3.13 TPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.40.3.14 TPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.40.3.15 TPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.40.3.16 TPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
  property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

3.1.40.3.17 TPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.40.3.18 TPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.40.3.19 TPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.40.3.20 TPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.40.3.21 TPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.40.3.22 TPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.40.3.23 TPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

3.1.40.3.24 TPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.40.3.25 TPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.40.3.26 TPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
```

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
  {
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnDockOver

3.1.41 TRadioGroup Class

TRadioGroup represents a group of radio buttons that function together.

Class Hierarchy

Pascal

```
TRadioGroup = class(TCustomRadioGroup);
```

C++

```
class TRadioGroup : public TCustomRadioGroup;
```

File

ExtCtrls

Description

A TRadioGroup object is a special group box that contains only radio buttons. Radio buttons that are placed directly in the same control component are said to be "grouped." When the user checks a radio button, all other radio buttons in its group become unchecked. Hence, two radio buttons on a form can be checked at the same time only if they are placed in separate containers, such as group boxes.

To add radio buttons to a TRadioGroup, edit the Items (see page 1424) property in the Object Inspector. Each string in Items (see page 1424) makes a radio button appear in the group box with the string as its caption. The value of the ItemIndex (see page 1423) property determines which radio button is currently selected.

Display the radio buttons in a single column or in multiple columns by setting the Columns (see page 1416) property.

Note: Setting the radio group's BiDiMode (see page 1413) to bdRightToLeft automatically flips its radio buttons. The FlipChildren (see page 817) method has no effect.

See Also

TDBRadioGroup

TRadioButton

Group Boxes and Radio Groups

Grouping Controls

3.1.41.1 TRadioGroup Members

The following tables list the members exposed by TRadioGroup.

Protected Methods

	Name	Description
	CanModify (see page 817)	Determines whether the user can select a different radio button in the group.
	Loaded (see page 818)	Initializes the radio group after it is loaded into memory.
	ReadState (see page 819)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	Create (see page 816)	Creates and initializes a TCustomRadioGroup (see page 815) instance.
	Destroy (see page 816)	Destroys the TCustomRadioGroup (see page 815) instance and the radio buttons it contains.
	FlipChildren (see page 817)	Prevents the default behavior of flipping the child radio buttons.
	GetChildren (see page 818)	Prevents the streaming system from loading and saving child controls in the usual fashion.

Public Properties

	Name	Description
	Buttons (see page 819)	Access individual TRadioButton objects.

Published Events

	Name	Description
	OnClick (see page 1438)	Occurs when the user clicks the control.
	OnContextPopup (see page 1440)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 1440)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1443)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1447)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1447)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1448)	Occurs when a control receives the input focus.
	OnExit (see page 1451)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 1453)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1454)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1409)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1412)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 1413)	Specifies the bi-directional mode for the control.
	Caption (see page 1413)	Specifies a text string that identifies the control to the user.
	Color (see page 1416)	Specifies the background color of the control.
	Columns (see page 1416)	Specifies number of columns in the radio group.
	Constraints (see page 1417)	Specifies the size constraints for the control.
	Ctl3D (see page 1417)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1418)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1420)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1420)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1420)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1422)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1422)	Controls the attributes of text written on or in the control.
	ItemIndex (see page 1423)	Indicates which radio button in the group is currently selected.
	Items (see page 1424)	Lists the radio buttons in the radio group.
	ParentBackground (see page 1425)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 1426)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1426)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1427)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 1427)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1427)	Determines where a control looks for its font information.
	ParentShowHint (see page 1428)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1428)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1433)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1435)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1435)	Determines if the user can tab to a control.
	Visible (see page 1436)	Determines whether the component appears onscreen.
	WordWrap (see page 1437)	This is WordWrap, a member of class TRadioGroup.

3.1.41.2 TRadioGroup Properties

The properties of the TRadioGroup class are listed here.

Published Properties

	Name	Description
	Align (see page 1409)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1412)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 1413)	Specifies the bi-directional mode for the control.
	Caption (see page 1413)	Specifies a text string that identifies the control to the user.
	Color (see page 1416)	Specifies the background color of the control.
	Columns (see page 1416)	Specifies number of columns in the radio group.
	Constraints (see page 1417)	Specifies the size constraints for the control.
	Ctl3D (see page 1417)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1418)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1420)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1420)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1420)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1422)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1422)	Controls the attributes of text written on or in the control.
	ItemIndex (see page 1423)	Indicates which radio button in the group is currently selected.
	Items (see page 1424)	Lists the radio buttons in the radio group.
	ParentBackground (see page 1425)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 1426)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 1426)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1427)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1427)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1427)	Determines where a control looks for its font information.
	ParentShowHint (see page 1428)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1428)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1433)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1435)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1435)	Determines if the user can tab to a control.
	Visible (see page 1436)	Determines whether the component appears onscreen.
	WordWrap (see page 1437)	This is WordWrap, a member of class TRadioGroup.

3.1.41.2.1 TRadioGroup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
```

```
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.41.2.2 TRadioGroup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.41.2.3 TRadioGroup.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

3.1.41.2.4 TRadioGroup.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
```

```
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
```

```
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
```

```
end;
```

3.1.41.2.5 TRadioGroup.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

3.1.41.2.6 TRadioGroup.Columns Property

Specifies number of columns in the radio group.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Columns determines the number of columns in the radio group if Orientation is orVertical (the default), and the number of rows if Orientation is orHorizontal. Its value can range from 1 to 16. The default value is 1, which means that the radio buttons are arranged in a single line.

3.1.41.2.7 TRadioGroup.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.41.2.8 TRadioGroup.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

3.1.41.2.9 TRadioGroup.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
```

```
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

3.1.41.2.10 TRadioGroup.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.41.2.11 TRadioGroup.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.41.2.12 TRadioGroup.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.41.2.13 TRadioGroup.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.41.2.14 TRadioGroup.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

3.1.41.2.15 TRadioGroup.ItemIndex Property

Indicates which radio button in the group is currently selected.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex holds the index of the selected radio button in the Items list. (The first button is 0.) The value of ItemIndex changes at runtime as the user selects radio buttons. If you want one of the buttons to appear selected when the application starts, assign

that button to ItemIndex at design time; otherwise, leave ItemIndex set to the default value of -1, which means that no button is selected.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

Items (see page 821)

3.1.41.2.16 TRadioGroup.Items Property

Lists the radio buttons in the radio group.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Items holds a TStrings object that lists the captions of the radio buttons in the group. In TCustomRadioGroup descendants, where this property is published, radio buttons can be added or removed by editing the Items list from the Object Inspector.

Note: A TRadioButton object is created automatically for each string in Items. You cannot directly add or remove TRadioButton objects, but you can access individual TButton objects through the Buttons property.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

ItemIndex (see page 820)

TRadioButton

Buttons (see page 819)

3.1.41.2.17 TRadioGroup.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

3.1.41.2.18 TRadioGroup.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

3.1.41.2.19 TRadioGroup.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.41.2.20 TRadioGroup.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

3.1.41.2.21 TRadioGroup.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

3.1.41.2.22 TRadioGroup.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

3.1.41.2.23 TRadioGroup.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.41.2.24 TRadioGroup.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
```

```
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

3.1.41.2.25 TRadioGroup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.41.2.26 TRadioGroup.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

3.1.41.2.27 TRadioGroup.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

3.1.41.2.28 TRadioGroup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.41.2.29 TRadioGroup.WordWrap Property

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

This is WordWrap, a member of class TRadioGroup.

3.1.41.3 TRadioGroup Events

The events of the TRadioGroup class are listed here.

Published Events

	Name	Description
	OnClick (see page 1438)	Occurs when the user clicks the control.
	OnContextPopup (see page 1440)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 1440)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1443)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1447)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1447)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1448)	Occurs when a control receives the input focus.
	OnExit (see page 1451)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 1453)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1454)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.41.3.1 TRadioGroup.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

3.1.41.3.2 TRadioGroup.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.41.3.3 TRadioGroup.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.41.3.4 TRadioGroup.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.41.3.5 TRadioGroup.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.41.3.6 TRadioGroup.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.41.3.7 TRadioGroup.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
  applied to the active control. Clicking the button sets the
  ActiveControl to the button.  That is why we need to save
  the ActiveControl in a shared OnEnter procedure.  In the rich
  edit, only the text selected converts.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
     FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

3.1.41.3.8 TRadioGroup.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
```

```
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

3.1.41.3.9 TRadioGroup.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.41.3.10 TRadioGroup.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

3.1.42 TRowCollection Class

Controls the **Collection Editor** for rows within a grid panel.

Class Hierarchy

Pascal

```
TRowCollection = class(TCellCollection);
```

C++

```
class TRowCollection : public TCellCollection;
```

File

ExtCtrls

Description

TRowCollection controls the **Collection Editor** for rows within a grid panel.

See Also

Collection Editor

3.1.42.1 TRowCollection Members

The following tables list the members exposed by TRowCollection.

Protected Methods

	Name	Description
	GetAttr (see page 459)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 459)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 460)	Returns a specified item in the collection.
	GetItemAttr (see page 1456)	Returns the value of a custom attribute assigned to one of the collection's items.
	Notify (see page 1457)	Responds when items are added to or removed from the collection.
	SetItem (see page 461)	This is SetItem, a member of class TCellCollection.
	Update (see page 461)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 1456)	Adds a row to the grid panel.
	Create (see page 1456)	This is Create, a member of class TRowCollection.
	Owner (see page 460)	Returns the Owner of the cell collection.

Public Properties

	Name	Description
	Items (see page 461)	Lists the items of the cell collection.

3.1.42.2 TRowCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TRowCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TRowCollection.

3.1.42.3 TRowCollection Methods

The methods of the TRowCollection class are listed here.

Protected Methods

	Name	Description
	GetItemAttr (see page 1456)	Returns the value of a custom attribute assigned to one of the collection's items.
	Notify (see page 1457)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 1456)	Adds a row to the grid panel.

3.1.42.3.1 TRowCollection.Add Method

Adds a row to the grid panel.

Pascal

```
function Add: TRowItem;
```

C++

```
__fastcall TRowItem Add();
```

Description

Use the Add method to add a row to the grid panel.

3.1.42.3.2 TRowCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

3.1.42.3.3 TRowCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added

Deleting

3.1.43 TRowItem Class

Class Hierarchy

Pascal

```
TRowItem = class(TCellItem);
```

C++

```
class TRowItem : public TCellItem;
```

File

ExtCtrls

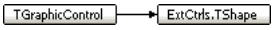
Description

This is class ExtCtrls.TRowItem.

3.1.44 TShape Class

TShape represents a geometric shape that can be drawn on a form.

Class Hierarchy



Pascal

```
TShape = class(TGraphicControl);
```

C++

```
class TShape : public TGraphicControl;
```

File

ExtCtrls

Description

Add a TShape object to a form to draw a simple geometric shape on the form. TShape introduces properties to describe the pen used to outline the shape and the brush used to fill it.

If the shape is only part of the image of a custom control, use the methods of the control's canvas instead.

See Also

TBrush (see page 2664)

TCanvas (see page 2673)

TPen (see page 2801)

TBevel (see page 163)

Graphic Controls

Shapes

3.1.44.1 TShape Members

The following tables list the members exposed by TShape.

Protected Methods

	Name	Description
	Paint (see page 1460)	Render control surface.

Public Methods

	Name	Description
	Create (see page 1459)	Creates an instance of TShape (see page 1458).
	Destroy (see page 1460)	Destroys of an instance of TShape (see page 1458).

Published Events

	Name	Description
	OnContextPopup (see page 1474)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 1475)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 1478)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1481)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1481)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1482)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1483)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1486)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1486)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1486)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1488)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1491)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1491)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Methods

	Name	Description
	StyleChanged (see page 1461)	This is StyleChanged, a member of class TShape.

Published Properties

	Name	Description
	Align (see page 1461)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1465)	Specifies how the control is anchored to its parent.
	Brush (see page 1465)	Specifies the color and pattern used for filling the shape control.
	Constraints (see page 1466)	Specifies the size constraints for the control.
	DragCursor (see page 1466)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1467)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1467)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1469)	Controls whether the control responds to mouse, keyboard, and timer events.
	ParentShowHint (see page 1469)	Determines where a control looks to find out if its Help Hint should be shown.
	Pen (see page 1470)	Specifies the pen used to outline the shape control.
	Shape (see page 1470)	Specifies the shape of the control.
	ShowHint (see page 1471)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 1472)	Determines whether the component appears onscreen.

3.1.44.2 TShape.Create Constructor

Creates an instance of TShape (see page 1458).

Pascal

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

C++

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

Description

Call TShape (see page 1458) indirectly, using the new keyword, to instantiate a TShape (see page 1458) object at runtime. Shapes added to a form at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the shape object.

See Also

Owner

3.1.44.3 TShape.Destroy Destructor

Destroys of an instance of TShape (see page 1458).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShape();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the object is not nil before it calls Destroy.

See Also

System::TObject::Free

3.1.44.4 TShape Methods

The methods of the TShape class are listed here.

Protected Methods

	Name	Description
	Paint (see page 1460)	Render control surface.

Published Methods

	Name	Description
	StyleChanged (see page 1461)	This is StyleChanged, a member of class TShape.

3.1.44.4.1 TShape.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

3.1.44.4.2 TShape.StyleChanged Method

Pascal

```
procedure StyleChanged(Sender: TObject);
```

C++

```
__fastcall StyleChanged(TObject * Sender);
```

Description

This is StyleChanged, a member of class TShape.

3.1.44.5 TShape Properties

The properties of the TShape class are listed here.

Published Properties

	Name	Description
	Align (see page 1461)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1465)	Specifies how the control is anchored to its parent.
	Brush (see page 1465)	Specifies the color and pattern used for filling the shape control.
	Constraints (see page 1466)	Specifies the size constraints for the control.
	DragCursor (see page 1466)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1467)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1467)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1469)	Controls whether the control responds to mouse, keyboard, and timer events.
	ParentShowHint (see page 1469)	Determines where a control looks to find out if its Help Hint should be shown.
	Pen (see page 1470)	Specifies the pen used to outline the shape control.
	Shape (see page 1470)	Specifies the shape of the control.
	ShowHint (see page 1471)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 1472)	Determines whether the component appears onscreen.

3.1.44.5.1 TShape.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
```

```
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

3.1.44.5.2 TShape.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.44.5.3 TShape.Brush Property

Specifies the color and pattern used for filling the shape control.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

Brush is a TBrush (see page 2664) object that represents the color and pattern that fill the interior of the shape. Use the properties and methods of TBrush (see page 2664) to change the color or pattern of the shape object:

```
Shape1.Brush.Color := clWhite;
Shape1->Brush->Color = clWhite;
```

Note: Setting the Brush property assigns the properties of another TBrush (see page 2664) object to the shape's brush. The shape object does not take ownership of the assigned object.

See Also

Pen (see page 1470)

TBrush (see page 2664)

3.1.44.5.4 TShape.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.44.5.5 TShape.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

3.1.44.5.6 TShape.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

3.1.44.5.7 TShape.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
```

```
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

3.1.44.5.8 TShape.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

3.1.44.5.9 TShape.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

3.1.44.5.10 TShape.Pen Property

Specifies the pen used to outline the shape control.

Pascal

```
property Pen: TPen;
```

C++

```
__property TPen Pen;
```

Description

Read Pen to obtain the TPen (see page 2801) object that represents the color and style of the line used to outline the shape. Use the properties of Pen to change the color or style.

Pen is a TPen (see page 2801) object that represents the color and style of the line used to outline the shape. Use the properties and methods of TPen (see page 2801) to change the color or pattern of the shape object:

Shape1.Pen.Width := 1;

Shape1.Pen.Color := Red;

Shape1->Pen->Width = 1;

Shape1->Pen->Color = clRed (see page 2916);

Note: Setting the Pen property assigns the properties of another TPen (see page 2801) object to the shape's pen. The shape object does not take ownership of the assigned object.

See Also

Brush (see page 1465)

TPen (see page 2801)

3.1.44.5.11 TShape.Shape Property

Specifies the shape of the control.

Pascal

```
property Shape: TShapeType;
```

C++

```
__property TShapeType Shape;
```

Description

Set Shape to the geometric shape that should be drawn on the form. The Shape property has these possible values:

Value	Meaning
stCircle	The shape is a circle.
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle. (Default)
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.
stSquare	The shape is a square.

3.1.44.5.12 TShape.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

3.1.44.5.13 TShape.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.44.6 TShape Events

The events of the TShape class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 1474)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 1475)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1478)	Occurs when the user drags an object over a control.
	OnEndDock (see page 1481)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 1481)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 1482)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1483)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1486)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1486)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1486)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1488)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 1491)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 1491)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

3.1.44.6.1 TShape.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

3.1.44.6.2 TShape.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

3.1.44.6.3 TShape.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
```

```
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

3.1.44.6.4 TShape.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

3.1.44.6.5 TShape.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

3.1.44.6.6 TShape.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
```

```
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.44.6.7 TShape.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

3.1.44.6.8 TShape.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

3.1.44.6.9 TShape.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

3.1.44.6.10 TShape.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

3.1.44.6.11 TShape.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

3.1.44.6.12 TShape.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

3.1.44.6.13 TShape.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

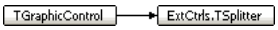
OnDragOver

OnDragDrop

3.1.45 TSplitter Class

TSplitter divides the client area of a form into resizable panes.

Class Hierarchy



Pascal

```
TSplitter = class(TGraphicControl);
```

C++

```
class TSplitter : public TGraphicControl;
```

File

ExtCtrls

Description

Add a splitter to a form between two aligned controls to allow users to resize the controls at runtime. The splitter sits between a control aligned to one edge of the form and the controls that fill up the rest of the client area. Give the splitter the same alignment as the control that is anchored to the edge of the form. When the user moves the splitter, it resizes the anchored control. This, in turn, changes the client area of the form, and the controls that fill up the rest of the client area resize accordingly.

Use each control on the form as a separate pane. After each pane is placed, place a splitter with the same alignment to allow that pane to be resized. The last pane to be placed on the form should be client-aligned, so that it resizes automatically to fill up the remaining space after all other panes are resized.

See Also

TForm (see page 2241)

Splitter Controls

3.1.45.1 TSplitter Members

The following tables list the members exposed by TSplitter.

Protected Methods

	Name	Description
	CanResize (see page 1496)	Determines how the splitter responds to mouse moves.
	DoCanResize (see page 1496)	Generates an OnCanResize (see page 1517) event and enforces the MinSize (see page 1511) property.
	MouseDown (see page 1497)	Dynamically locates the control that can be resized by the splitter.
	MouseMove (see page 1497)	Provides visual feedback of how the neighboring control would be resized.
	MouseUp (see page 1497)	Resizes the neighboring control when the user releases the mouse.
	Paint (see page 1498)	Draws the image of the splitter.
	RequestAlign (see page 1498)	Changes the cursor when the alignment of the splitter changes.
	StopSizing (see page 1499)	Cleans up after the splitter has resized its neighboring control.

Public Methods

	Name	Description
	Create (see page 1493)	Creates an instance of TSplitter (see page 1492).
	Destroy (see page 1495)	Frees the memory associated with the splitter.

Public Properties

	Name	Description
	Canvas (see page 1504)	Provides the drawing surface used by the graphic control.

Published Events

	Name	Description
	OnCanResize (see page 1517)	Occurs when the user tries to move the splitter with the mouse.
	OnMoved (see page 1517)	Occurs after the user has retiled the form using the splitter.
	OnPaint (see page 1520)	Occurs when the splitter needs to draw itself.

Published Properties

	Name	Description
	Align (see page 1499)	Determines how the control aligns within its container (parent control).
	AutoSnap (see page 1503)	Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 1511).
	Beveled (see page 1503)	Determines whether the splitter looks beveled along the edge that moves.
	Color (see page 1506)	Specifies the background color of the control.
	Constraints (see page 1507)	Specifies the size constraints for the control.
	Cursor (see page 1507)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	MinSize (see page 1511)	Specifies the minimum size, in pixels, of the panes on either side of the splitter.
	ParentColor (see page 1512)	Determines where a control looks for its color information.
	ResizeStyle (see page 1513)	Specifies the effect of moving the splitter.
	Visible (see page 1514)	Determines whether the component appears onscreen.
	Width (see page 1515)	Specifies the horizontal size of the control or form in pixels.

3.1.45.2 TSplitter.Create Constructor

Creates an instance of TSplitter (see page 1492).

Pascal

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

C++

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

Description

Call Create to instantiate a splitter at runtime. TSplitter (see page 1492) objects placed on forms at design time are created automatically.

AOwner specifies the form whose surface the splitter divides.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
```

```
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

Owner

3.1.45.3 TSplitter.Destroy Destructor

Frees the memory associated with the splitter.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSplitter();
```

Description

Do not call Destroy directly. Instead,call Free, which checks that the splitter is not nil before calling Destroy.

See Also

System::TObject::Free

3.1.45.4 TSplitter Methods

The methods of the TSplitter class are listed here.

Protected Methods

	Name	Description
	CanResize (see page 1496)	Determines how the splitter responds to mouse moves.
	DoCanResize (see page 1496)	Generates an OnCanResize (see page 1517) event and enforces the MinSize (see page 1511) property.
	MouseDown (see page 1497)	Dynamically locates the control that can be resized by the splitter.
	MouseMove (see page 1497)	Provides visual feedback of how the neighboring control would be resized.
	MouseUp (see page 1497)	Resizes the neighboring control when the user releases the mouse.
	Paint (see page 1498)	Draws the image of the splitter.
	RequestAlign (see page 1498)	Changes the cursor when the alignment of the splitter changes.
	StopSizing (see page 1499)	Cleans up after the splitter has resized its neighboring control.

3.1.45.4.1 TSplitter.CanResize Method

Determines how the splitter responds to mouse moves.

Pascal

```
function CanResize(var NewSize: Integer): Boolean; virtual; reintroduce;
```

C++

```
virtual __fastcall Boolean CanResize(int NewSize);
```

Description

CanResize is called by the DoCanResize (see page 1496) method to generate an OnCanResize (see page 1517) event. The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. CanResize returns a boolean value that indicates whether the value specified by NewSize should be applied by redrawing the line (when ResizeStyle (see page 1513) is rsLine) or resizing the associated control (when ResizeStyle (see page 1513) is rsUpdate);

Unless an OnCanResize (see page 1517) event handler sets the value of NewSize and the return value, CanResize returns true without changing the NewSize parameter.

Note: CanResize is not called if the value of ResizeStyle (see page 1513) is rsNone when the user presses the mouse button.

See Also

OnCanResize (see page 1517)

ResizeStyle (see page 1513)

DoCanResize (see page 1496)

3.1.45.4.2 TSplitter.DoCanResize Method

Generates an OnCanResize (see page 1517) event and enforces the MinSize (see page 1511) property.

Pascal

```
function DoCanResize(var NewSize: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCanResize(int NewSize);
```

Description

DoCanResize is called automatically when the user moves the mouse after pressing the mouse button on the splitter. The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. DoCanResize returns a boolean value that indicates whether the value specified by NewSize should be applied by redrawing the line (when ResizeStyle (see page 1513) is rsLine) or resizing the associated control (when ResizeStyle (see page 1513) is rsUpdate).

DoCanResize calls CanResize (see page 1496) to generate an OnCanResize (see page 1517) event. If, after calling the event handler, NewSize is less than the value of MinSize (see page 1511), NewSize is changed to zero.

Note: DoCanResize is not called if the value of ResizeStyle (see page 1513) is rsNone when the user presses the mouse button.

3.1.45.4.3 TSplitter.MouseDown Method

Dynamically locates the control that can be resized by the splitter.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the parent class's method, MouseDown locates the control that it is about to resize. Because this control is located dynamically, an application can delete or add aligned controls beside a splitter at runtime and the splitter will resize whichever control is currently beside it.

See Also

MouseDown

MouseMove (see page 1497)

MouseUp (see page 1497)

3.1.45.4.4 TSplitter.MouseMove Method

Provides visual feedback of how the neighboring control would be resized.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method, MouseMove computes the new boundaries of the neighboring control and draws a line where the new edge would be. The control is not actually resized until the mouse is released.

See Also

OnCanResize (see page 1517)

ResizeStyle (see page 1513)

CanResize (see page 1496)

MinSize (see page 1511)

MouseMove

MouseDown (see page 1497)

MouseUp (see page 1497)

3.1.45.4.5 TSplitter.MouseUp Method

Resizes the neighboring control when the user releases the mouse.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method, MouseUp resizes the neighboring control that was located when the mouse went down. After the control is resized, MouseUp generates an OnMoved (see page 1517) event.

See Also

MouseUp

MouseDown (see page 1497)

MouseMove (see page 1497)

3.1.45.4.6 TSplitter.Paint Method

Draws the image of the splitter.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when the splitter receives WM_PAINT (see page 5198) messages from its Parent. Paint draws the background color of the splitter, and if Beveled (see page 1503) is true, draws a bevel along the movable edges. After drawing the background and bevel, Paint generates an OnPaint (see page 1520) event.

See Also

Beveled (see page 1503)

Color

Parent

OnPaint (see page 1520)

3.1.45.4.7 TSplitter.RequestAlign Method

Changes the cursor when the alignment of the splitter changes.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

RequestAlign changes the cursor from crHSplit to crVSplit when the Align (see page 1499) property changes to alBottom or alTop. It changes the cursor from crVSplit to crHSplit when the Align (see page 1499) property changes to any other value.

If the Cursor (see page 1507) property is not crHSplit or crVSplit, RequestAlign has no effect.

Override RequestAlign to suppress the automatic cursor change or to implement a different association between alignment and the splitter's cursor.

See Also

Align

Cursor

3.1.45.4.8 TSplitter.StopSizing Method

Cleans up after the splitter has resized its neighboring control.

Pascal

```
procedure StopSizing; dynamic;
```

C++

```
__fastcall StopSizing();
```

Description

This protected procedure is called when the user releases the mouse or hits the Escape key while using the splitter to resize a control. StopSizing makes sure that the resize line is no longer visible and generates an OnMoved (see page 1517) event.

See Also

OnMoved (see page 1517)

3.1.45.5 TSplitter Properties

The properties of the TSplitter class are listed here.

Public Properties

	Name	Description
	Canvas (see page 1504)	Provides the drawing surface used by the graphic control.

Published Properties

	Name	Description
	Align (see page 1499)	Determines how the control aligns within its container (parent control).
	AutoSnap (see page 1503)	Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 1511).
	Beveled (see page 1503)	Determines whether the splitter looks beveled along the edge that moves.
	Color (see page 1506)	Specifies the background color of the control.
	Constraints (see page 1507)	Specifies the size constraints for the control.
	Cursor (see page 1507)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	MinSize (see page 1511)	Specifies the minimum size, in pixels, of the panes on either side of the splitter.
	ParentColor (see page 1512)	Determines where a control looks for its color information.
	ResizeStyle (see page 1513)	Specifies the effect of moving the splitter.
	Visible (see page 1514)	Determines whether the component appears onscreen.
	Width (see page 1515)	Specifies the horizontal size of the control or form in pixels.

3.1.45.5.1 TSplitter.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
```

```
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
```

```
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.45.5.2 TSplitter.AutoSnap Property

Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 1511).

Pascal

```
property AutoSnap: Boolean;
```

C++

```
__property Boolean AutoSnap;
```

Description

Set AutoSnap to false to prevent the splitter from resizing neighboring objects to zero when the user tries to make them smaller than MinSize (see page 1511). When AutoSnap is false, such resize attempts are simply blocked, leaving the neighboring object with a size of MinSize (see page 1511). The default value of AutoSnap is true.

See Also

MinSize (see page 1511)

3.1.45.5.3 TSplitter.Beveled Property

Determines whether the splitter looks beveled along the edge that moves.

Pascal

```
property Beveled: Boolean;
```

C++

```
__property Boolean Beveled;
```

Description

Set Beveled to false to remove the beveled edges from the splitter. Set Beveled to true to give the edge a 3D beveled look.

The bevel on a splitter runs in a single direction only: along the edges the user moves to resize neighboring objects. Thus, if the

Align (see page 1499) property is alLeft or alRight, the vertical edges of the splitter are beveled. If the Align (see page 1499) property is alTop or alBottom, the horizontal edges of the splitter are beveled. The default value of Beveled is false.

See Also

Align

3.1.45.5.4 TSplitter.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
```

```
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = dynamic_cast<TLabel *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
```

```
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

3.1.45.5.5 TSplitter.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
```

```
end;
```

See Also

ParentColor

3.1.45.5.6 TSplitter.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

3.1.45.5.7 TSplitter.Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Pascal

```
property Cursor;
```

C++

```
__property Cursor;
```

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

C++ Examples:

```
/*
```

```
  This example uses a splitter and two image controls to
  create a "slide show" effect.  The example displays an image
  (the current "slide"), which can be hidden behind a screen
  that looks like a venetian blind. When the blind is pulled
  down to completely cover the current slide, the slide changes
  to a different image.  When the blind is raised, the new
  image appears.
  To prepare this example, follow these steps:
  1. Place an image control on the form.
  Set the picture property to a bitmap of a venetian blind,
  Set the Stretch property to True.
  Set the Align property to alTop.
  2. Add a splitter control to the form.
  Set the Align property to alTop.
  Set the Cursor property to crVSplit.
  Set the OnMoved event handler to the code that appears below.
  Set the MinSize property to 1.
  3. Add a second image control to the form.
  Set the Align property to alClient.
  Set the Stretch property to True.
  4. Select the form and add the OnCreate and OnDestroy event
     handlers that appear below.
  5. Add the global variables below to the unit.
  */
  int CurImage;
  Graphics::TBitmap *BMPs[5];

  void __fastcall TForm1::Splitter1Moved(TObject *Sender)
  {
    // check if the blind is closed
    if (Image2->Height <= 1)
    {
      // the blind is closed, change the current slide
      CurImage++;
      if (CurImage == 5)
         CurImage = 0;
      Image2->Picture->Bitmap = BMPs[CurImage];
    }
  }

  #include <memory>       //for STL auto_ptr class

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    // Initialize the array of slides.
    // These must be freed in the OnDestroy event handler.

    static std::auto_ptr<Graphics::TBitmap> _MyBMP0Cleaner(BMPs[0] = new Graphics::TBitmap);
    BMPs[0]->LoadFromFile("..\\factory.bmp");

    static std::auto_ptr<Graphics::TBitmap> _MyBMP1Cleaner(BMPs[1] = new Graphics::TBitmap);
    BMPs[1]->LoadFromFile("..\\Rhododendron.bmp");

    static std::auto_ptr<Graphics::TBitmap> _MyBMP2Cleaner(BMPs[2] = new Graphics::TBitmap);
    BMPs[2]->LoadFromFile("..\\littleB_64.bmp");

    static std::auto_ptr<Graphics::TBitmap> _MyBMP3Cleaner(BMPs[3] = new Graphics::TBitmap);
    BMPs[3]->LoadFromFile("..\\Soap Bubbles.bmp");

    static std::auto_ptr<Graphics::TBitmap> _MyBMP4Cleaner(BMPs[4] = new Graphics::TBitmap);
    BMPs[4]->LoadFromFile("..\\Gone Fishing.bmp");

    CurImage = 0;

    Image2->Picture->Bitmap = BMPs[0];
  }
```

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/

#include <memory>       //for STL auto_ptr class

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bmpMask(new Graphics::TBitmap);
  std::auto_ptr<Graphics::TBitmap> bmpColor(new Graphics::TBitmap);
  std::auto_ptr<TIconInfo> iconInfo(new TIconInfo);

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo.get());

  Screen->Cursor = crMyCursor;
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
```

```
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize the array of slides.
    These must be freed in the OnDestroy event handler. }
  BMPs[1] := TBitmap.Create;
  BMPs[1].LoadFromFile('Factory.bmp');
  BMPs[2] := TBitmap.Create;
  BMPs[2].LoadFromFile('Rhododendron.bmp');
  BMPs[3] := TBitmap.Create;
  BMPs[3].LoadFromFile('littleB_64.bmp');
  BMPs[4] := TBitmap.Create;
  BMPs[4].LoadFromFile('Soap bubbles.bmp');
  BMPs[5] := TBitmap.Create;
  BMPs[5].LoadFromFile('Gone Fishing.bmp');
  CurImage := 1;
  Image2.Picture.Bitmap := BMPs[1];
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  { clean up }
  for I := 1 to 5 do
    BMPs[I].Free;
end;
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);
```

```
    Screen.Cursor := crMyCursor;

    bmpMask.Free;
    bmpColor.Free;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    DestroyIcon(Screen.Cursors[crMyCursor]);
  end;
```

See Also

DragCursor

Cursors (see page 2458)

Screen (see page 2615)

ProcessMessages (see page 1954)

3.1.45.5.8 TSplitter.MinSize Property

Specifies the minimum size, in pixels, of the panes on either side of the splitter.

Pascal

```
property MinSize: NaturalNumber;
```

C++

```
__property NaturalNumber MinSize;
```

Description

Set MinSize to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align (see page 1499) property is alLeft or alRight, the splitter cannot resize the regions to its left or right any smaller than MinSize pixels. If the Align (see page 1499) property is alTop or alBottom, the splitter cannot resize the regions above or below it any smaller than MinSize pixels. The default value of MinSize is 30.

Note: Always set MinSize to a value less than half the client width of its parent. When MinSize is half the client width of the splitter's parent, the splitter cannot move because to do so would be to resize one of the panes less than MinSize pixels.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

Align

3.1.45.5.9 TSplitter.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

3.1.45.5.10 TSplitter.ResizeStyle Property

Specifies the effect of moving the splitter.

Pascal

```
property ResizeStyle: TResizeStyle;
```

C++

```
__property TResizeStyle ResizeStyle;
```

Description

Use ResizeStyle to determine what happens when the user moves the splitter with the mouse.

ResizeStyle can have one of the values in the following table:

Value	Meaning
rsNone	Moving the splitter with the mouse has no effect.
rsLine	A line is drawn indicating the new position of the splitter that would result if the user released the mouse at the current position. The splitter does not actually move and no controls are resized.
rsUpdate	The control against which the splitter is aligned is resized based on the current mouse position, and the splitter is moved to that position.
rsPattern	A pattern is drawn indicating the new position of the splitter that would result if the user released the mouse at the current position. The splitter does not actually move and no controls are resized. This is the default value for ResizeStyle.

See Also

OnCanResize (see page 1517)

OnMoved (see page 1517)

3.1.45.5.11 TSplitter.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.45.5.12 TSplitter.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

3.1.45.6 TSplitter Events

The events of the TSplitter class are listed here.

Published Events

	Name	Description
	OnCanResize (see page 1517)	Occurs when the user tries to move the splitter with the mouse.
	OnMoved (see page 1517)	Occurs after the user has retiled the form using the splitter.
	OnPaint (see page 1520)	Occurs when the splitter needs to draw itself.

3.1.45.6.1 TSplitter.OnCanResize Event

Occurs when the user tries to move the splitter with the mouse.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Write an OnCanResize event handler to conditionally prevent the user from resizing neighboring components using the splitter.

The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. This value can be changed if the attempted resize can be partially allowed. The Accept parameter indicates whether the resize should be permitted at all. Set Accept to false to disallow any movement of the splitter.

See Also

OnMoved (see page 1517)

ResizeStyle (see page 1513)

3.1.45.6.2 TSplitter.OnMoved Event

Occurs after the user has retiled the form using the splitter.

Pascal

```
property OnMoved: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMoved;
```

Description

Write an OnMoved event handler to take specific action when the user finishes dragging the edge of the neighboring control using the splitter. For example, after resizing a control on the edge of the form, use an OnMoved event handler to update the layout of the new client area of the form. Use the ClientRect property of the form to determine the new size remaining.

C++ Examples:

```
/*
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
*/
int CurImage;
Graphics::TBitmap *BMPs[5];

void __fastcall TForm1::Splitter1Moved(TObject *Sender)
{
  // check if the blind is closed
  if (Image2->Height <= 1)
  {
    // the blind is closed, change the current slide
    CurImage++;
    if (CurImage == 5)
       CurImage = 0;
    Image2->Picture->Bitmap = BMPs[CurImage];
  }
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Initialize the array of slides.
  // These must be freed in the OnDestroy event handler.

  static std::auto_ptr<Graphics::TBitmap> _MyBMP0Cleaner(BMPs[0] = new Graphics::TBitmap);
  BMPs[0]->LoadFromFile("..\\factory.bmp");

  static std::auto_ptr<Graphics::TBitmap> _MyBMP1Cleaner(BMPs[1] = new Graphics::TBitmap);
  BMPs[1]->LoadFromFile("..\\Rhododendron.bmp");

  static std::auto_ptr<Graphics::TBitmap> _MyBMP2Cleaner(BMPs[2] = new Graphics::TBitmap);
  BMPs[2]->LoadFromFile("..\\littleB_64.bmp");

  static std::auto_ptr<Graphics::TBitmap> _MyBMP3Cleaner(BMPs[3] = new Graphics::TBitmap);
  BMPs[3]->LoadFromFile("..\\Soap Bubbles.bmp");

  static std::auto_ptr<Graphics::TBitmap> _MyBMP4Cleaner(BMPs[4] = new Graphics::TBitmap);
  BMPs[4]->LoadFromFile("..\\Gone Fishing.bmp");

  CurImage = 0;
```

```
  Image2->Picture->Bitmap = BMPs[0];
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize the array of slides.
    These must be freed in the OnDestroy event handler. }
  BMPs[1] := TBitmap.Create;
  BMPs[1].LoadFromFile('Factory.bmp');
  BMPs[2] := TBitmap.Create;
  BMPs[2].LoadFromFile('Rhododendron.bmp');
  BMPs[3] := TBitmap.Create;
  BMPs[3].LoadFromFile('littleB_64.bmp');
  BMPs[4] := TBitmap.Create;
  BMPs[4].LoadFromFile('Soap bubbles.bmp');
  BMPs[5] := TBitmap.Create;
  BMPs[5].LoadFromFile('Gone Fishing.bmp');
  CurImage := 1;
  Image2.Picture.Bitmap := BMPs[1];
end;

procedure TForm1.FormDestroy(Sender: TObject);
```

```
var
  I: Integer;
begin
  { clean up }
  for I := 1 to 5 do
    BMPs[I].Free;
end;
```

See Also

ClientRect

3.1.45.6.3 TSplitter.OnPaint Event

Occurs when the splitter needs to draw itself.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to customize the appearance of the splitter. For example, use an OnPaint event handler to give a splitter the appearance of the splitters that appear in cool bar objects. Use the Canvas (see page 1504) property as a drawing surface for painting the splitter. Use the ClientRect property to determine the coordinates of the canvas on which to paint the splitter.

When a splitter object receives a WM_PAINT (see page 5198) message, it paints its background before generating the OnPaint event. If the Beveled (see page 1503) property is true, the beveled appearance will also be added before the OnPaint event. Most applications set Beveled (see page 1503) to false when using the OnPaint event to customize the appearance of the splitter.

See Also

ClientRect

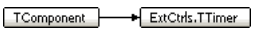
Canvas

Beveled (see page 1503)

3.1.46 TTimer Class

TTimer encapsulates the Windows API timer functions.

Class Hierarchy



Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TTimer = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TTimer : public TComponent;
```

File

ExtCtrls

Description

TTimer is used to simplify calling the Windows API timer functions SetTimer and KillTimer, and to simplify processing the WM_TIMER (see page 5215) messages. Use one timer component for each timer in the application.

The execution of the timer occurs through its OnTimer (see page 1527) event. TTimer has an Interval (see page 1525) property that determines how often the timer's OnTimer (see page 1527) event occurs. Interval (see page 1525) corresponds to the parameter for the Windows API SetTimer function.

Warning: Caution:Limitations on the total number of timers system-wide are system dependent.

3.1.46.1 TTimer Members

The following tables list the members exposed by TTimer.

Protected Methods

	Name	Description
	Timer (see page 1522)	This is Timer, a member of class TTimer.

Public Methods

	Name	Description
	Create (see page 1521)	Instantiates a timer object.
	Destroy (see page 1522)	Disposes of a timer object.

Published Events

	Name	Description
	OnTimer (see page 1527)	Occurs when a specified amount of time, determined by the Interval (see page 1525) property, has passed.

Published Properties

	Name	Description
	Enabled (see page 1523)	Controls whether the timer generates OnTimer (see page 1527) events periodically.
	Interval (see page 1525)	Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 1527) event.

3.1.46.2 TTimer.Create Constructor

Instantiates a timer object.

Pascal

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

C++

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

Description

Call Create to instantiate a timer at runtime. Timers added to forms or data modules at design time are created automatically.

AOwner specifies the component, typically a form or data module, that is responsible for freeing the timer.

See Also

Owner

3.1.46.3 TTimer.Destroy Destructor

Disposes of a timer object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTimer();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the timer is not nil before it calls Destroy.

Destroy deactivates the timer by setting Enabled (see page 1523) to False before freeing the resources required by the timer.

See Also

Create (see page 1521)

3.1.46.4 TTimer Methods

The methods of the TTimer class are listed here.

Protected Methods

	Name	Description
	Timer (see page 1522)	This is Timer, a member of class TTimer.

3.1.46.4.1 TTimer.Timer Method

Pascal

```
procedure Timer; dynamic;
```

C++

```
__fastcall Timer();
```

Description

This is Timer, a member of class TTimer.

3.1.46.5 TTimer Properties

The properties of the TTimer class are listed here.

Published Properties

	Name	Description
	Enabled (see page 1523)	Controls whether the timer generates OnTimer (see page 1527) events periodically.

	Interval (see page 1525)	Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 1527) event.

3.1.46.5.1 TTimer.Enabled Property

Controls whether the timer generates OnTimer (see page 1527) events periodically.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to enable or disable the timer. If Enabled is true, the timer responds normally. If Enabled is false, the timer does not generate OnTimer (see page 1527) events. The default is true.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
```

```
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
  TimerOver: Word;

procedure TForm1.FormClick(Sender: TObject);
begin
  MediaPlayer1.DeviceType := dtCDAudio;
  MediaPlayer1.Open;
  MediaPlayer1.Play;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if TimeOver = 10 then
  begin
    MediaPlayer1.Eject;
    MediaPlayer1.Close;
    { disable the timer, as we are done }
    Timer1.Enabled := False;
```

```
    end
    else
      if (MediaPlayer1.Mode = mpOpen) then
       Inc(TimeOver);
end;
```

See Also

OnTimer (see page 1527)

3.1.46.5.2 TTimer.Interval Property

Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 1527) event.

Pascal

```
property Interval: Cardinal;
```

C++

```
__property unsigned Interval;
```

Description

Interval determines how frequently the OnTimer (see page 1527) event occurs. Each time the specified interval passes, the OnTimer (see page 1527) event occurs.

Use Interval to specify any cardinal value as the interval between OnTimer (see page 1527) events. The default value is 1000 (one second).

Note: A 0 value is valid, however the timer will not call an OnTimer (see page 1527) event for a value of 0.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
/*
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Timer1->Interval = StrToInt(MaskEdit1->Text);
  Timer1->Enabled = true;
  Hide();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
  Visible = true;
  Timer1->Enabled = false;
}
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
{
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Interval := StrtoInt(MaskEdit1.Text);
  Timer1.Enabled := True;
  Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Visible := True;
  Timer1.Enabled := False;
end;
{
This example uses a timer on a form. When the application
```

```
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

OnTimer (see page 1527)

3.1.46.6 TTimer Events

The events of the TTimer class are listed here.

Published Events

	Name	Description
	OnTimer (see page 1527)	Occurs when a specified amount of time, determined by the Interval (see page 1525) property, has passed.

3.1.46.6.1 TTimer.OnTimer Event

Occurs when a specified amount of time, determined by the Interval (see page 1525) property, has passed.

Pascal

```
property OnTimer: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTimer;
```

Description

Write an OnTimer event handler to execute an action at regular intervals.

The Interval (see page 1525) property of a timer determines how frequently the OnTimer event occurs. Each time the specified interval passes, the OnTimer event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
```

```
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Interval (see page 1525)

3.1.47 TTrayIcon Class

Creates an icon in the system tray next to the clock.

Class Hierarchy

Pascal

```
TTrayIcon = class(TCustomTrayIcon);
```

C++

```
class TTrayIcon : public TCustomTrayIcon;
```

File

ExtCtrls

Description

Use the TTrayIcon component to create an icon in the system tray next to the clock. TTrayIcon has events for mouse clicks, properties for menus and animation of the icon, and methods for showing balloon hints.

3.1.47.1 TTrayIcon Members

The following tables list the members exposed by TTrayIcon.

Protected Methods

	Name	Description
	DoOnAnimate (see page 825)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 825)	This is GetAnimateInterval, a member of class TCustomTrayIcon.

	GetBalloonTimeout (see page 825)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
V	Notification (see page 826)	Forwards notification messages to all owned components.
	SetAnimate (see page 827)	This is SetAnimate, a member of class TCustomTrayIcon.
	SetAnimateInterval (see page 827)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 827)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 827)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 828)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetHint (see page 828)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 828)	This is SetIcon, a member of class TCustomTrayIcon.
V	SetIconIndex (see page 828)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 829)	This is SetIconList, a member of class TCustomTrayIcon.
V	SetVisible (see page 829)	This is SetVisible, a member of class TCustomTrayIcon.
V	WindowProc (see page 829)	This is WindowProc, a member of class TCustomTrayIcon.

Protected Properties

	Name	Description
R	Data (see page 832)	This is Data, a member of class TCustomTrayIcon.

Public Methods

	Name	Description
V	Create (see page 824)	Creates and initializes a TCustomTrayIcon (see page 822) instance.
V	Destroy (see page 824)	Frees the memory associated with the TCustomTrayIcon (see page 822) object.
	Refresh (see page 826)	This is the overview for the Refresh method overload.
	SetDefaultIcon (see page 828)	Sets the tray icon to the default application icon.
V	ShowBalloonHint (see page 829)	Displays the balloon hint.

Published Events

	Name	Description
	OnAnimate (see page 1533)	Occurs each time the icon changes when the tray icon is animating.
	OnBalloonClick (see page 1534)	This is OnBalloonClick, a member of class TTrayIcon.
	OnClick (see page 1534)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 1534)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 1534)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 1535)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 1535)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	Animate (see page 1530)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 1530)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 1530)	Specifies an icon for the balloon hint.
	BalloonHint (see page 1531)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 1531)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 1531)	Specifies the text that appears as the balloon title.
	Hint (see page 1532)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 1532)	Specifies the default icon shown in the system tray.
	IconIndex (see page 1532)	This is an index into the Icons image list which is the icon currently being displayed in the system tray.
	Icons (see page 1532)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 1533)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 1533)	Determines whether the icon appears in the system tray.

3.1.47.2 TTrayIcon Properties

The properties of the TTrayIcon class are listed here.

Published Properties

	Name	Description
	Animate (see page 1530)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 1530)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 1530)	Specifies an icon for the balloon hint.
	BalloonHint (see page 1531)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 1531)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 1531)	Specifies the text that appears as the balloon title.
	Hint (see page 1532)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 1532)	Specifies the default icon shown in the system tray.
	IconIndex (see page 1532)	This is an index into the Icons image list which is the icon currently being displayed in the system tray.
	Icons (see page 1532)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 1533)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 1533)	Determines whether the icon appears in the system tray.

3.1.47.2.1 TTrayIcon.Animate Property

Specifies whether the icon in the system tray is static or animated.

Pascal

```
property Animate: Boolean;
```

C++

```
__property Boolean Animate;
```

Description

When set to True, the icon shown in the system tray displays, in sequence, the images in the image list specified by the Icons property. The images change only if the specified image list contains images. The images change on a rotating basis by the interval set by the AnimateInterval property. When set to False the animation stops.

3.1.47.2.2 TTrayIcon.AnimateInterval Property

Controls the length of time between image changes for the tray icon image.

Pascal

```
property AnimateInterval: Cardinal;
```

C++

```
__property unsigned AnimateInterval;
```

Description

Use AnimateInterval to specify the length of time between image changes in the tray icon. The default interval is 1000 ms (1 second).

3.1.47.2.3 TTrayIcon.BalloonFlags Property

Specifies an icon for the balloon hint.

Pascal

```
property BalloonFlags: TBalloonFlags;
```

C++

```
__property TBalloonFlags BalloonFlags;
```

Description

The following values specify an icon to be displayed in the balloon hint:

- bfNone
- bfInfo
- bfWarning
- bfError

3.1.47.2.4 TTrayIcon.BalloonHint Property

Specifies the text within a balloon hint.

Pascal

```
property BalloonHint: string;
```

C++

```
__property UnicodeString BalloonHint;
```

Description

Use BalloonHint to specify the text string that appears when a balloon hint is shown.

3.1.47.2.5 TTrayIcon.BalloonTimeout Property

Specifies the length of time that the balloon displays.

Pascal

```
property BalloonTimeout: Integer;
```

C++

```
__property int BalloonTimeout;
```

Description

Use BalloonTimeout to set the length of time that a balloon displays. By default, this value is set to 3000 ms (3 seconds).

3.1.47.2.6 TTrayIcon.BalloonTitle Property

Specifies the text that appears as the balloon title.

Pascal

```
property BalloonTitle: string;
```

C++

```
__property UnicodeString BalloonTitle;
```

Description

Specifies the text that appears as the balloon title.

3.1.47.2.7 TTrayIcon.Hint Property

Specifies the text that appears when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property UnicodeString Hint;
```

Description

Specifies the text that appears when the user moves the mouse over the control.

3.1.47.2.8 TTrayIcon.Icon Property

Specifies the default icon shown in the system tray.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use the Icon property to set the default icon to be shown in the system tray. If you do not specify a value, the application icon is used.

3.1.47.2.9 TTrayIcon.IconIndex Property

This is an index into the Icons image list which is the icon currently being displayed in the system tray.

Pascal

```
property IconIndex: Integer;
```

C++

```
__property int IconIndex;
```

Description

This is an index into the Icons image list which is the icon currently being displayed in the system tray.

3.1.47.2.10 TTrayIcon.Icons Property

Specifies the list of images to be displayed for the icon in the system tray.

Pascal

```
property Icons: TImageList;
```

C++

```
__property TImageList Icons;
```

Description

Use this property to specify the list of images used for the icon in the system tray. These images are displayed when the Animate property is set to True.

3.1.47.2.11 TTrayIcon.PopupMenu Property

Identifies the pop-up menu associated with the tray icon.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Use PopupMenuOn to specify when the menu is displayed.

3.1.47.2.12 TTrayIcon.Visible Property

Determines whether the icon appears in the system tray.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Determines whether the icon appears in the system tray.

3.1.47.3 TTrayIcon Events

The events of the TTrayIcon class are listed here.

Published Events

	Name	Description
	OnAnimate (see page 1533)	Occurs each time the icon changes when the tray icon is animating.
	OnBalloonClick (see page 1534)	This is OnBalloonClick, a member of class TTrayIcon.
	OnClick (see page 1534)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 1534)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 1534)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 1535)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 1535)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

3.1.47.3.1 TTrayIcon.OnAnimate Event

Occurs each time the icon changes when the tray icon is animating.

Pascal

```
property OnAnimate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAnimate;
```

Description

Occurs each time the icon changes when the tray icon is animating.

3.1.47.3.2 TTrayIcon.OnBalloonClick Event

Pascal

```
property OnBalloonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBalloonClick;
```

Description

This is OnBalloonClick, a member of class TTrayIcon.

3.1.47.3.3 TTrayIcon.OnClick Event

Occurs when the user clicks the tray icon.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Occurs when the user clicks the tray icon.

3.1.47.3.4 TTrayIcon.OnDblClick Event

Occurs when the user double clicks the tray icon.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Occurs when the user double clicks the tray icon.

3.1.47.3.5 TTrayIcon.OnMouseDown Event

Occurs when the user presses a mouse button while the mouse pointer is over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Occurs when the user presses a mouse button while the mouse pointer is over a control.

3.1.47.3.6 TTrayIcon.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

3.1.47.3.7 TTrayIcon.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

3.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Frame3D (see page 1536)	Draws a three dimensional frame on a canvas.
	NotebookHandlesNeeded (see page 1537)	This is function ExtCtrls.NotebookHandlesNeeded.

3.2.1 ExtCtrls.Frame3D Function

Draws a three dimensional frame on a canvas.

Pascal

```
procedure Frame3D(Canvas: TCanvas; var Rect: TRect; TopColor: TColor; BottomColor: TColor;
Width: Integer);
```

C++

```
Frame3D(TCanvas Canvas, TRect Rect, TColor TopColor, TColor BottomColor, int Width);
```

File

ExtCtrls

Description

Use Frame3D to create three-dimensional effects when drawing directly on a canvas.

The Canvas parameter is the canvas to draw on.

The Rect parameter specifies the position on the canvas where the three-dimensional rectangle should appear. It returns the dimensions of the rectangle that lies inside the frame.

The TopColor and BottomColor parameters indicate the colors that appear on the left and top or the right and bottom of the rectangle respectively.

The Width parameter specifies the width of the frame.

To create a raised effect, the TopColor should be a light color and the BottomColor should be a shadow color. To create a depressed effect, the TopColor should be the shadow color and the BottomColor should be the light color. To create a beveled effect, draw a frame immediately inside another frame with the colors reversed.

See Also

TWinControl

TWinControl

TWincontrol

TWinControl

TWInControl

TBevel

TFrameControl

3.2.2 ExtCtrls.NotebookHandlesNeeded Function

Pascal

```
procedure NotebookHandlesNeeded(Notebook: TNotebook);
```

C++

```
NotebookHandlesNeeded(TNotebook Notebook);
```

File

ExtCtrls

Description

This is function ExtCtrls.NotebookHandlesNeeded.

3.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBalloonFlags (see page 1538)	A type specified when showing a balloon hint that puts one of the default icons on the balloon message.
	TBandDrawingStyle (see page 1539)	This is record ExtCtrls.TBandDrawingStyle.
	TBandPaintOption (see page 1539)	TBandPaintOption and TBandPaintOptions (see page 1549) specify which parts of a band are painted.
	TBevelShape (see page 1540)	TBevelShape determines the overall shape of a bevel control.
	TBevelStyle (see page 1541)	TBevelStyle indicates whether a bevel appears raised or lowered.
	TColorBoxStyles (see page 1541)	This is record ExtCtrls.TColorBoxStyles.
	TCornerEdge (see page 1542)	The size of the cornering for the edge of the bands in a TControlBar (see page 590).
	TExpandStyle (see page 1542)	Specifies how a grid panel expands.
	TFlowStyle (see page 1543)	This is record ExtCtrls.TFlowStyle.
	TLabelPosition (see page 1543)	TLabelPosition specifies the position of the label in a labeled edit control.
	TResizeStyle (see page 1544)	This is record ExtCtrls.TResizeStyle.
	TShapeType (see page 1544)	TShapeType determines the shape of a TShape (see page 1458) control.
	TSizeStyle (see page 1545)	This is record ExtCtrls.TSizeStyle.
	TSysLinkType (see page 1546)	TSysLinkType describes an HTML link type.

3.3.1 ExtCtrls.TBalloonFlags Enumeration

A type specified when showing a balloon hint that puts one of the default icons on the balloon message.

Pascal

```
TBalloonFlags = (
  bfNone = NIIF_NONE,
  bfInfo = NIIF_INFO,
  bfWarning = NIIF_WARNING,
  bfError = NIIF_ERROR
);
```

C++

```
enum TBalloonFlags {
  bfNone = NIIF_NONE,
  bfInfo = NIIF_INFO,
  bfWarning = NIIF_WARNING,
  bfError = NIIF_ERROR
};
```

File

ExtCtrls

Description

You can set TBalloonFlags to one of the following values:

- bfNone

- bfInfo
- bfWarning
- bfError

3.3.2 ExtCtrls.TBandDrawingStyle Enumeration

Pascal

```
TBandDrawingStyle = (
  dsNormal,
  dsGradient
);
```

C++

```
enum TBandDrawingStyle {
  dsNormal,
  dsGradient
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TBandDrawingStyle.

3.3.3 ExtCtrls.TBandPaintOption Enumeration

TBandPaintOption and TBandPaintOptions (see page 1549) specify which parts of a band are painted.

Pascal

```
TBandPaintOption = (
  bpoGrabber,
  bpoFrame,
  bpoGradient,
  bpoRoundRect
);
```

C++

```
enum TBandPaintOption {
  bpoGrabber,
  bpoFrame,
  bpoGradient,
  bpoRoundRect
};
```

File

ExtCtrls

Description

TBandPaintOptions (see page 1549) has one or more of the following values:

Value	Meaning
bpoGrabber	The grabber is painted.
bpoFrame	The frame is painted.
bpoGradient	The gradient is painted.
bpoRoundRect	The round rectangle is painted.

3.3.4 ExtCtrls.TBevelShape Enumeration

TBevelShape determines the overall shape of a bevel control.

Pascal

```
TBevelShape = (
  bsBox,
  bsFrame,
  bsTopLine,
  bsBottomLine,
  bsLeftLine,
  bsRightLine,
  bsSpacer
);
```

C++

```
enum TBevelShape {
  bsBox,
  bsFrame,
  bsTopLine,
  bsBottomLine,
  bsLeftLine,
  bsRightLine,
  bsSpacer
};
```

File

ExtCtrls

Description

These are the possible values of the TBevelShape type:

Value	Meaning
bsBox	The entire client area appears raised or lowered.
bsFrame	The client area is outlined by a raised or lowered frame.
bsTopLine	The bevel displays a line at the top of the client area.
bsBottomLine	The bevel displays a line at the bottom of the client area.
bsLeftLine	The bevel displays a line at the left side of the client area.
bsRightLine	The bevel displays a line at the right side of the client area.
bsSpacer	The bevel is an empty space.

3.3.5 ExtCtrls.TBevelStyle Enumeration

TBevelStyle indicates whether a bevel appears raised or lowered.

Pascal

```
TBevelStyle = (
  bsLowered,
  bsRaised
);
```

C++

```
enum TBevelStyle {
  bsLowered,
  bsRaised
};
```

File

ExtCtrls

Description

These are the possible values of the TBevelStyle type:

Value	Meaning
bsLowered	The bevel is lowered.
bsRaised	The bevel is raised.

3.3.6 ExtCtrls.TColorBoxStyles Enumeration

Pascal

```
TColorBoxStyles = (
  cbStandardColors,
  cbExtendedColors,
  cbSystemColors,
  cbIncludeNone,
  cbIncludeDefault,
  cbCustomColor,
  cbPrettyNames,
  cbCustomColors
);
```

C++

```
enum TColorBoxStyles {
  cbStandardColors,
  cbExtendedColors,
  cbSystemColors,
  cbIncludeNone,
  cbIncludeDefault,
  cbCustomColor,
  cbPrettyNames,
  cbCustomColors
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TColorBoxStyles.

3.3.7 ExtCtrls.TCornerEdge Enumeration

The size of the cornering for the edge of the bands in a TControlBar (see page 590).

Pascal

```
TCornerEdge = (
  ceNone,
  ceSmall,
  ceMedium,
  ceLarge
);
```

C++

```
enum TCornerEdge {
  ceNone,
  ceSmall,
  ceMedium,
  ceLarge
};
```

File

ExtCtrls

Description

This type contains the size of the cornering for the edge of the bands in a TControlBar (see page 590).

3.3.8 ExtCtrls.TExpandStyle Enumeration

Specifies how a grid panel expands.

Pascal

```
TExpandStyle = (
  emAddRows,
  emAddColumns,
  emFixedSize
);
```

C++

```
enum TExpandStyle {
  emAddRows,
  emAddColumns,
  emFixedSize
};
```

File

ExtCtrls

Description

Use TExpandStyle to specify whether a grid panel adds rows or columns when new controls are added.

3.3.9 ExtCtrls.TFlowStyle Enumeration

Pascal

```
TFlowStyle = (
  fsLeftRightTopBottom,
  fsRightLeftTopBottom,
  fsLeftRightBottomTop,
  fsRightLeftBottomTop,
  fsTopBottomLeftRight,
  fsBottomTopLeftRight,
  fsTopBottomRightLeft,
  fsBottomTopRightLeft
);
```

C++

```
enum TFlowStyle {
  fsLeftRightTopBottom,
  fsRightLeftTopBottom,
  fsLeftRightBottomTop,
  fsRightLeftBottomTop,
  fsTopBottomLeftRight,
  fsBottomTopLeftRight,
  fsTopBottomRightLeft,
  fsBottomTopRightLeft
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TFlowStyle.

3.3.10 ExtCtrls.TLabelPosition Enumeration

TLabelPosition specifies the position of the label in a labeled edit control.

Pascal

```
TLabelPosition = (
  lpAbove,
  lpBelow,
  lpLeft,
  lpRight
);
```

C++

```
enum TLabelPosition {
  lpAbove,
  lpBelow,
  lpLeft,
  lpRight
```

```
};
```

File

ExtCtrls

Description

The following are possible values of TLabelPosition:

Value	Meaning
lpAbove	The label appears above the edit control.
lpBelow	The label appears below the edit control.
lpLeft	The label appears to the left of the edit control.
lpRight	The label appears to the right of the edit control.

3.3.11 ExtCtrls.TResizeStyle Enumeration

Pascal

```
TResizeStyle = (
  rsNone,
  rsLine,
  rsUpdate,
  rsPattern
);
```

C++

```
enum TResizeStyle {
  rsNone,
  rsLine,
  rsUpdate,
  rsPattern
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TResizeStyle.

3.3.12 ExtCtrls.TShapeType Enumeration

TShapeType determines the shape of a TShape (see page 1458) control.

Pascal

```
TShapeType = (
  stRectangle,
  stSquare,
  stRoundRect,
  stRoundSquare,
  stEllipse,
  stCircle
```

```
);
```

C++

```
enum TShapeType {
  stRectangle,
  stSquare,
  stRoundRect,
  stRoundSquare,
  stEllipse,
  stCircle
};
```

File

ExtCtrls

Description

These are the possible values of the TShapeType type:

Value	Meaning
stCircle	The shape is a circle.
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle. (Default)
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.
stSquare	The shape is a square.

3.3.13 ExtCtrls.TSizeStyle Enumeration

Pascal

```
TSizeStyle = (
  ssAbsolute,
  ssPercent,
  ssAuto
);
```

C++

```
enum TSizeStyle {
  ssAbsolute,
  ssPercent,
  ssAuto
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TSizeStyle.

3.3.14 ExtCtrls.TSysLinkType Enumeration

TSysLinkType describes an HTML link type.

Pascal

```
TSysLinkType = (
  sltURL,
  sltID
);
```

C++

```
enum TSysLinkType {
  sltURL,
  sltID
};
```

File

ExtCtrls

Description

TSysLinkType is used in HTML link click events to identify the link type. These are the possible values of the TSysLinkType type:

Value	Meaning
sltURL	HREF link
sltID	ID

See Also

TCustomLinkLabel (see page 797)

OnLinkClick (see page 802)

TSysLinkEvent (see page 1555)

3.4 Types

The following table lists types in this documentation.

Types

Name	Description
NaturalNumber (see page 1547)	This is type ExtCtrls.NaturalNumber.
TBandDragEvent (see page 1548)	This is type ExtCtrls.TBandDragEvent.
TBandInfoEvent (see page 1548)	This is type ExtCtrls.TBandInfoEvent.
TBandMoveEvent (see page 1548)	This is type ExtCtrls.TBandMoveEvent.
TBandPaintEvent (see page 1549)	This is type ExtCtrls.TBandPaintEvent.
TBandPaintOptions (see page 1549)	This is type ExtCtrls.TBandPaintOptions.
TBeginBandMoveEvent (see page 1549)	This is type ExtCtrls.TBeginBandMoveEvent.
TCanResizeEvent (see page 1550)	This is type ExtCtrls.TCanResizeEvent.
TCategoryPanelClass (see page 1550)	This is type ExtCtrls.TCategoryPanelClass.
TCategoryPanelSurfaceClass (see page 1550)	This is type ExtCtrls.TCategoryPanelSurfaceClass.
TCellSpan (see page 1551)	This is type ExtCtrls.TCellSpan.
TColorBoxStyle (see page 1551)	TColorBoxStyle and TColorBoxStyles (see page 1541) specify how a color box control displays its selections.
TEditButtonClass (see page 1552)	This is type ExtCtrls.TEditButtonClass.
TEndBandMoveEvent (see page 1552)	This is type ExtCtrls.TEndBandMoveEvent.
TGetColorsEvent (see page 1552)	This is type ExtCtrls.TGetColorsEvent.
TLBGetColorsEvent (see page 1553)	This is type ExtCtrls.TLBGetColorsEvent.
TPanelBevel (see page 1553)	TPanelBevel specifies a bevel's cut.
TRowSize (see page 1554)	This is type ExtCtrls.TRowSize.
TSectionEvent (see page 1554)	TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 1021) control is resized.
TSplitterCanResizeEvent (see page 1554)	This is type ExtCtrls.TSplitterCanResizeEvent.
TSysLinkEvent (see page 1555)	Event routine for click on TCustomLinkLabel (see page 797) HTML link.

3.4.1 ExtCtrls.NaturalNumber Type

Pascal

```
NaturalNumber = 1..High(Integer);
```

C++

```
1..High(Integer) NaturalNumber;
```

File

ExtCtrls

Description

This is type ExtCtrls.NaturalNumber.

3.4.2 ExtCtrls.TBandDragEvent Type

Pascal

```
TBandDragEvent = procedure (Sender: TObject; Control: TControl; var Drag: Boolean) of object;
```

C++

```
(Sender: TObject; Control: TControl; var Drag: Boolean) ( TBandDragEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandDragEvent.

3.4.3 ExtCtrls.TBandInfoEvent Type

Pascal

```
TBandInfoEvent = procedure (Sender: TObject; Control: TControl; var Insets: TRect; var
PreferredSize, RowCount: Integer) of object;
```

C++

```
(Sender: TObject; Control: TControl; var Insets: TRect; var PreferredSize, RowCount: Integer)
( TBandInfoEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandInfoEvent.

3.4.4 ExtCtrls.TBandMoveEvent Type

Pascal

```
TBandMoveEvent = procedure (Sender: TObject; Control: TControl; var ARect: TRect) of object;
```

C++

```
(Sender: TObject; Control: TControl; var ARect: TRect) ( TBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandMoveEvent.

3.4.5 ExtCtrls.TBandPaintEvent Type

Pascal

```
TBandPaintEvent = procedure (Sender: TObject; Control: TControl; Canvas: TCanvas; var ARect:
TRect; var Options: TBandPaintOptions) of object;
```

C++

```
(Sender: TObject; Control: TControl; Canvas: TCanvas; var ARect: TRect; var Options:
TBandPaintOptions) ( TBandPaintEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandPaintEvent.

3.4.6 ExtCtrls.TBandPaintOptions Type

Pascal

```
TBandPaintOptions = set of TBandPaintOption;
```

C++

```
TBandPaintOption TBandPaintOptions;
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandPaintOptions.

3.4.7 ExtCtrls.TBeginBandMoveEvent Type

Pascal

```
TBeginBandMoveEvent = procedure (Sender: TObject; Control: TControl; var AllowMove: Boolean)
of object;
```

C++

```
(Sender: TObject; Control: TControl; var AllowMove: Boolean) ( TBeginBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBeginBandMoveEvent.

3.4.8 ExtCtrls.TCanResizeEvent Type

Pascal

```
TCanResizeEvent = TSplitterCanResizeEvent;
```

C++

```
TSplitterCanResizeEvent TCanResizeEvent;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCanResizeEvent.

3.4.9 ExtCtrls.TCategoryPanelClass Type

Pascal

```
TCategoryPanelClass = class of TCustomCategoryPanel;
```

C++

```
TCustomCategoryPanel TCategoryPanelClass;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCategoryPanelClass.

3.4.10 ExtCtrls.TCategoryPanelSurfaceClass Type

Pascal

```
TCategoryPanelSurfaceClass = class of TCategoryPanelSurface;
```

C++

```
TCategoryPanelSurface TCategoryPanelSurfaceClass;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCategoryPanelSurfaceClass.

3.4.11 ExtCtrls.TCellSpan Type

Pascal

```
TCellSpan = 1..MaxInt;
```

C++

```
1..MaxInt TCellSpan;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCellSpan.

3.4.12 ExtCtrls.TColorBoxStyle Type

TColorBoxStyle and TColorBoxStyles (see page 1541) specify how a color box control displays its selections.

Pascal

```
TColorBoxStyle = set of TColorBoxStyles;
```

C++

```
TColorBoxStyles TColorBoxStyle;
```

File

ExtCtrls

Description

TColorBoxStyle is a set of TColorBoxStyles (see page 1541) values. The possible values of TColorBoxStyles (see page 1541) are

Style	Meaning when included
cbStandardColors	The color box lists the first 16 color constants from the Graphics (see page 2619) unit. These are the constants such as clRed (see page 2916) or clBlack (see page 2899), that represent fixed RGB values.
cbExtendedColors	The color box includes clMoneyGreen (see page 2913), clSkyBlue (see page 2917), clCream (see page 2902), and clMedGray (see page 2911).
cbSystemColors	The color box includes the symbolic color constants that reflect the values specified in the Windows control panel.
cbIncludeNone	The color box includes clNone (see page 2914). This option only has an effect if Style includes cbSystemColors.
cbIncludeDefault	The color box includes clNone (see page 2914). This option only has an effect if Style includes cbSystemColors.
cbCustomColor	The first entry in the color box represents a custom color. When the user selects the Custom entry, the color box displays the Color selection dialog, where the user can specify a custom color.
cbPrettyNames	The color box lists the names as words rather than as the color constant strings. (for example 'Black' rather than 'clBlack (see page 2899)').

3.4.13 ExtCtrls.TEditButtonClass Type

Pascal

```
TEditButtonClass = class of TEditButton;
```

C++

```
TEditButton TEditButtonClass;
```

File

ExtCtrls

Description

This is type ExtCtrls.TEditButtonClass.

3.4.14 ExtCtrls.TEndBandMoveEvent Type

Pascal

```
TEndBandMoveEvent = procedure (Sender: TObject; Control: TControl) of object;
```

C++

```
(Sender: TObject; Control: TControl) ( TEndBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TEndBandMoveEvent.

3.4.15 ExtCtrls.TGetColorsEvent Type

Pascal

```
TGetColorsEvent = procedure (Sender: TCustomColorBox; Items: TStrings) of object;
```

C++

```
(Sender: TCustomColorBox; Items: TStrings) ( TGetColorsEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TGetColorsEvent.

3.4.16 ExtCtrls.TLBGetColorsEvent Type

Pascal

```
TLBGetColorsEvent = procedure (Sender: TCustomColorListBox; Items: TStrings) of object;
```

C++

```
(Sender: TCustomColorListBox; Items: TStrings) ( TLBGetColorsEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TLBGetColorsEvent.

3.4.17 ExtCtrls.TPanelBevel Type

TPanelBevel specifies a bevel's cut.

Pascal

```
TPanelBevel = TBevelCut;
```

C++

```
TBevelCut TPanelBevel;
```

File

ExtCtrls

Description

TPanelBevel can have one of the following values:

Value	Meaning
bvNone	The bevel does not exist.
bvLowered	The bevel appears lowered.
bvRaised	The bevel appears raised.
bvSpace	The bevel appears as a space if its kind is not bkTile. Otherwise, the bevel appears raised.

See Also

TWinControl

TWinControl

3.4.18 ExtCtrls.TRowSize Type

Pascal

```
TRowSize = 1..MaxInt;
```

C++

```
1..MaxInt TRowSize;
```

File

ExtCtrls

Description

This is type ExtCtrls.TRowSize.

3.4.19 ExtCtrls.TSectionEvent Type

TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 1021) control is resized.

Pascal

```
TSectionEvent = procedure (Sender: TObject; ASection, AWidth: Integer) of object;
```

C++

```
(Sender: TObject; ASection, AWidth: Integer) ( TSectionEvent)();
```

File

ExtCtrls

Description

Sender is the header control.

ASection is the index of the section that was resized.

AWidth is the width of the section.

3.4.20 ExtCtrls.TSplitterCanResizeEvent Type

Pascal

```
TSplitterCanResizeEvent = procedure (Sender: TObject; var NewSize: Integer; var Accept:
Boolean) of object;
```

C++

```
(Sender: TObject; var NewSize: Integer; var Accept: Boolean) ( TSplitterCanResizeEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TSplitterCanResizeEvent.

3.4.21 ExtCtrls.TSysLinkEvent Type

Event routine for click on TCustomLinkLabel (see page 797) HTML link.

Pascal

```
TSysLinkEvent = procedure (Sender: TObject; const Link: string; LinkType: TSysLinkType) of object;
```

C++

```
(Sender: TObject; const Link: string; LinkType: TSysLinkType) ( TSysLinkEvent)();
```

File

ExtCtrls

Description

TSysLinkEvent is a click event routine for a TCustomLinkLabel (see page 797) HTML link. It identifies the link URI and link type.

See Also

TCustomLinkLabel (see page 797)

OnLinkClick (see page 802)

TSysLinkType (see page 1546)

3.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DefaultCategoryPanelHeight (see page 1556)	This is constant ExtCtrls.DefaultCategoryPanelHeight.
NoColorSelected (see page 1556)	This is constant ExtCtrls.NoColorSelected.
WM_SYSTEM_TRAY_MESSAGE (see page 1556)	This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

3.5.1 ExtCtrls.DefaultCategoryPanelHeight Constant

Pascal

```
DefaultCategoryPanelHeight = 200;
```

C++

```
DefaultCategoryPanelHeight = 200;
```

File

ExtCtrls

Description

This is constant ExtCtrls.DefaultCategoryPanelHeight.

3.5.2 ExtCtrls.NoColorSelected Constant

Pascal

```
NoColorSelected = TColor($FF000000);
```

C++

```
NoColorSelected = TColor($FF000000);
```

File

ExtCtrls

Description

This is constant ExtCtrls.NoColorSelected.

3.5.3 ExtCtrls.WM_SYSTEM_TRAY_MESSAGE Constant

Pascal

```
WM_SYSTEM_TRAY_MESSAGE = WM_USER + 1;
```

C++

```
WM_SYSTEM_TRAY_MESSAGE = WM_USER + 1;
```

File

ExtCtrls

Description

This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

4 ExtDlgs Namespace

This is namespace ExtDlgs.

Topics

Name	Description
Classes (see page 1560)	The following table lists classes in this documentation.
Functions (see page 1585)	The following table lists functions in this documentation.
Constants (see page 1586)	The following table lists constants in this documentation.

Classes

	Name	Description
	TOpenPictureDialog (see page 1560)	TOpenPictureDialog displays a graphics-file selection dialog.
	TOpenTextFileDialog (see page 1570)	This is class ExtDlgs.TOpenTextFileDialog.
	TSavePictureDialog (see page 1577)	TSavePictureDialog displays a "Save As" dialog for saving graphics files.
	TSaveTextFileDialog (see page 1580)	This is class ExtDlgs.TSaveTextFileDialog.

Constants

Name	Description
DefaultEncodingNames (see page 1586)	This is constant ExtDlgs.DefaultEncodingNames.

Functions

	Name	Description
	StandardEncodingFromName (see page 1585)	This is function ExtDlgs.StandardEncodingFromName.

4.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOpenPictureDialog (see page 1560)	TOpenPictureDialog displays a graphics-file selection dialog.
	TOpenTextFileDialog (see page 1570)	This is class ExtDlgs.TOpenTextFileDialog.
	TSavePictureDialog (see page 1577)	TSavePictureDialog displays a "Save As" dialog for saving graphics files.
	TSaveTextFileDialog (see page 1580)	This is class ExtDlgs.TSaveTextFileDialog.

4.1.1 TOpenPictureDialog Class

TOpenPictureDialog displays a graphics-file selection dialog.

Class Hierarchy

Pascal

```
TOpenPictureDialog = class(TOpenDialog);
```

C++

```
class TOpenPictureDialog : public TOpenDialog;
```

File

ExtDlgs

Description

TOpenPictureDialog displays a modal Windows dialog box for selecting and opening graphics files. This component is just like TOpenDialog, except that it includes a rectangular preview region. If the selected image can be read by TPicture (see page 2812), it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

If the user selects a file of an unrecognized type, TPicture (see page 2812) raises an EInvalidGraphic (see page 2627) exception.

Delphi Examples:

```
{
This example uses an image, two buttons, an open picture dialog
and a save picture dialog on a form. Clicking on the Open/Save button
executes the open/save picture dialog.
An exception is raised when trying to open a file that
does not exist, or when trying to overwrite a file using the
save dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  { execute an open picture dialog }
  if OpenPictureDialog1.Execute then
    { first check if file exists }
```

```
    if FileExists(OpenPictureDialog1.FileName) then
      { if it exists, load the data into the image component }
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName)
    else
      { otherwise raise an exception }
      raise Exception.Create('File does not exist.');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  { execute a save picture dialog }
  if SavePictureDialog1.Execute then
    { first check if file exists }
    if FileExists(SavePictureDialog1.FileName) then
      { if it exists, raise an exception }
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      { otherwise, save the image data into the file }
      Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Refresh;
end;
```

See Also

EInvalidGraphic (see page 2627)

TOpenDialog

TPicture (see page 2812)

TSavePictureDialog (see page 1577)

GraphicFilter

4.1.1.1 TOpenPictureDialog Members

The following tables list the members exposed by TOpenPictureDialog.

Protected Methods

	Name	Description
V	DoClose (see page 1562)	Generates an OnClose event.
V	DoSelectionChange (see page 1563)	This is DoSelectionChange, a member of class TOpenPictureDialog.
V	DoShow (see page 1563)	Generates an OnShow event.
V	PreviewClick (see page 1565)	This is PreviewClick, a member of class TOpenPictureDialog.

Protected Properties

	Name	Description
R	ImageCtrl (see page 1569)	This is ImageCtrl, a member of class TOpenPictureDialog.
R	PictureLabel (see page 1570)	This is PictureLabel, a member of class TOpenPictureDialog.

Public Methods

	Name	Description
V	Create (see page 1562)	Creates and initializes a TOpenPictureDialog (see page 1560) instance.
V	Execute (see page 1564)	Displays the file-selection dialog.

Published Properties

	Name	Description
	Filter (see page 1566)	Determines the file masks (filters) available in the dialog.

4.1.1.2 TOpenPictureDialog.Create Constructor

Creates and initializes a TOpenPictureDialog (see page 1560) instance.

Pascal

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

C++

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

Description

The Create method generates a TOpenPictureDialog (see page 1560) instance, but the dialog does not appear at runtime until the Execute (see page 1564) method is called. Create initializes the Filter (see page 1566) property by defining a filter for each file type that TPicture (see page 2812) supports.

See Also

Execute (see page 1564)

Filter

TPicture (see page 2812)

4.1.1.3 TOpenPictureDialog Methods

The methods of the TOpenPictureDialog class are listed here.

Protected Methods

	Name	Description
	DoClose (see page 1562)	Generates an OnClose event.
	DoSelectionChange (see page 1563)	This is DoSelectionChange, a member of class TOpenPictureDialog.
	DoShow (see page 1563)	Generates an OnShow event.
	PreviewClick (see page 1565)	This is PreviewClick, a member of class TOpenPictureDialog.

Public Methods

	Name	Description
	Execute (see page 1564)	Displays the file-selection dialog.

4.1.1.3.1 TOpenPictureDialog.DoClose Method

Generates an OnClose event.

Pascal

```
procedure DoClose; override;
```

C++

```
virtual __fastcall DoClose();
```

Description

DoClose is called automatically when the user closes the dialog. Override this method to provide additional processing other

than calling the OnClose event handler. For example, to perform shutdown operations that use protected methods of the dialog class, override DoClose rather than writing an OnClose event handler (which can access only public methods).

See Also

OnClose

4.1.1.3.2 TOpenPictureDialog.DoSelectionChange Method

Pascal

```
procedure DoSelectionChange; override;
```

C++

```
virtual __fastcall DoSelectionChange();
```

Description

This is DoSelectionChange, a member of class TOpenPictureDialog.

4.1.1.3.3 TOpenPictureDialog.DoShow Method

Generates an OnShow event.

Pascal

```
procedure DoShow; override;
```

C++

```
virtual __fastcall DoShow();
```

Description

DoShow is called automatically when the dialog is first displayed. Override this method to provide additional processing other than calling the OnShow event handler. For example, to perform initializations that use protected methods of the dialog class, override DoShow rather than writing an OnShow event handler (which can access only public methods).

C++ Examples:

```
/*
The following example is taken from TOpenPictureDialog. It
calculates the preview rectangle based on the difference
between the client area of the dialog and the static area of
the standard controls.
*/

#include <ExtDlgs.hpp>

class MyOpenPictureDialog : public TOpenPictureDialog
{
private:    // User declarations
    TPanel *FPicturePanel;  // can't use the private versions in TOpenPictureDialog
    TLabel *FPictureLabel;
    TSpeedButton *FPreviewButton;
    TPanel *FPaintPanel;
    TImage *FImageCtrl;
    AnsiString FSavedFilename;
protected:  // User declarations
    DYNAMIC void __fastcall DoShow(void);
public:     // User declarations
    __fastcall MyOpenPictureDialog(TComponent* Owner);
    virtual bool __fastcall  Execute(HWND ParentWnd){ return TOpenDialog::Execute(ParentWnd); }
};
```

```
__fastcall MyOpenPictureDialog::MyOpenPictureDialog(TComponent* Owner)
    : TOpenPictureDialog(Owner)
{
}

#include <memory>       //for STL auto_ptr class

void __fastcall MyOpenPictureDialog::DoShow(void)
{
//  RECT *PreviewRect = new RECT();
  std::auto_ptr<RECT> PreviewRect(new RECT());
  TRect StaticRect;
  FPicturePanel = new TPanel(this); // the owner will clean this up
  FPictureLabel = new TLabel(this); // the owner will clean this up
  FPreviewButton = new TSpeedButton(this); // the owner will clean this up
  FPaintPanel = new TPanel(this); // the owner will clean this up
  FImageCtrl = new TImage(this); // the owner will clean this up
  // Set preview area to entire dialog
  GetClientRect(Handle, PreviewRect.get());
  StaticRect = GetStaticRect();
  // Move preview area to right of static area
  PreviewRect->left = StaticRect.Left + (StaticRect.Right - StaticRect.Left);
  PreviewRect->top += 4;
  FPicturePanel->BoundsRect.Left = PreviewRect->left;
  FPicturePanel->BoundsRect.Right = PreviewRect->right;
  FPicturePanel->BoundsRect.Top = PreviewRect->top;
  FPicturePanel->BoundsRect.Bottom = PreviewRect->bottom;

  FPreviewButton->Left = FPaintPanel->BoundsRect.Right - FPreviewButton->Width - 2;
  FImageCtrl->Picture = NULL;
  FSavedFilename = "";
  FPaintPanel->Caption = "";
  FPicturePanel->ParentWindow = Handle;
//  TOpenDialog::DoShow();   // the above code is equivalent to the inherited version
}

TForm1 *Form1;
MyOpenPictureDialog *OpenPictureDialog1;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (OpenPictureDialog1->Execute(GetParentHandle()))
      Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OpenPictureDialog1 =  new MyOpenPictureDialog(Form1); // the owner will clean this up
}
```

See Also

OnShow

4.1.1.3.4 TOpenPictureDialog.Execute Method

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

Delphi Examples:

```
{
This example uses an image, two buttons, an open picture dialog
and a save picture dialog on a form. Clicking on the Open/Save button
executes the open/save picture dialog.
An exception is raised when trying to open a file that
does not exist, or when trying to overwrite a file using the
save dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  { execute an open picture dialog }
  if OpenPictureDialog1.Execute then
    { first check if file exists }
    if FileExists(OpenPictureDialog1.FileName) then
      { if it exists, load the data into the image component }
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName)
    else
      { otherwise raise an exception }
      raise Exception.Create('File does not exist.');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  { execute a save picture dialog }
  if SavePictureDialog1.Execute then
    { first check if file exists }
    if FileExists(SavePictureDialog1.FileName) then
      { if it exists, raise an exception }
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      { otherwise, save the image data into the file }
      Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Refresh;
end;
```

4.1.1.3.5 TOpenPictureDialog.PreviewClick Method

Pascal

```
procedure PreviewClick(Sender: TObject); virtual;
```

C++

```
virtual __fastcall PreviewClick(TObject * Sender);
```

Description

This is PreviewClick, a member of class TOpenPictureDialog.

4.1.1.4 TOpenPictureDialog Properties

The properties of the TOpenPictureDialog class are listed here.

Protected Properties

	Name	Description
	ImageCtrl (see page 1569)	This is ImageCtrl, a member of class TOpenPictureDialog.
	PictureLabel (see page 1570)	This is PictureLabel, a member of class TOpenPictureDialog.

Published Properties

	Name	Description
	Filter (see page 1566)	Determines the file masks (filters) available in the dialog.

4.1.1.4.1 TOpenPictureDialog.Filter Property

Determines the file masks (filters) available in the dialog.

Pascal

```
property Filter;
```

C++

```
__property Filter;
```

Description

The file-selection dialog includes a drop-down list of file types under the edit box. When the user picks a file type from the list, only files of the selected type are displayed in the dialog.

To configure file masks at design time, click on the ellipsis marks (...) to the right of the Filter property in the Object Inspector. This opens the Filter editor. In the left column of the Filter editor, under Filter Name, type a brief description of each file type that will be available at runtime. In the right column, under Filter, type the file mask corresponding to each description. For example, the description "Text files" might appear to the left of the mask "*.txt", and the description "Pascal source files" might appear to the left of the mask "*.pas". Since the description appears in the drop-down list at runtime, it is often helpful to show the mask explicitly in the description (for example, "Text files (*.txt)").

To create file masks in program code, assign a value to the Filter property that consists of a description and a mask separated by a vertical bar (pipe) character. Do not include spaces around the vertical bar. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT";
```

Multiple filters should be separated by vertical bars. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS";
```

To include multiple masks in a single filter, separate the masks with semicolons. This works both in the Object Inspector and in program code. For example,

```
OpenDialog1.Filter := 'Pascal files|*.PAS;*.DPK;*.DPR';
OpenDialog1->Filter = "Pascal files|*.PAS;*.DPK;*.DPR";
```

If no value is assigned to Filter, the dialog displays all file types.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
#include <memory>       //for STL auto_ptr class
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, (fmOpenRead |
fmShareDenyWrite)));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, (fmCreate |
fmShareDenyRead)));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
```

```
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
{
This example uses a memo box, two buttons, an open dialog
and a save dialog on a form. When the application runs,
```

```
  the open dialog and save dialog filters are initialized,
  so to open and save *.txt files or files with any extension.
  Clicking on the Open/Save button executes the open/save dialog.
  An exception is raised when trying to open a file that
  does not exist, or when trying to overwrite a file using the
  save dialog.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    { execute an open file dialog }
    if OpenDialog1.Execute then
      { first check if file exists }
      if FileExists(OpenDialog1.FileName) then
        { if it exists, load the data into the memo box }
        Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
      else
        { otherwise raise an exception }
        raise Exception.Create('File does not exist.');
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    { execute a save file dialog }
    if SaveDialog1.Execute then
      { first check if file exists }
      if FileExists(SaveDialog1.FileName) then
        { if it exists, raise an exception }
        raise Exception.Create('File already exists. Cannot overwrite.')
      else
        { otherwise, save the memo box lines into the file }
        Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    {
    initialize the dialog filters to open/save *.txt files
    and also files with arbitrary extension
    }
    OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
    SaveDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
  end;
```

See Also

FilterIndex

GraphicFilter (see page 2854)

4.1.1.4.2 TOpenPictureDialog.ImageCtrl Property

Pascal

```
property ImageCtrl: TImage;
```

C++

```
__property TImage ImageCtrl;
```

Description

This is ImageCtrl, a member of class TOpenPictureDialog.

4.1.1.4.3 TOpenPictureDialog.PictureLabel Property

Pascal

```
property PictureLabel: TLabel;
```

C++

```
__property TLabel PictureLabel;
```

Description

This is PictureLabel, a member of class TOpenPictureDialog.

4.1.2 TOpenTextFileDialog Class

Class Hierarchy

Pascal

```
TOpenTextFileDialog = class(TOpenDialog);
```

C++

```
class TOpenTextFileDialog : public TOpenDialog;
```

File

ExtDlgs

Description

This is class ExtDlgs.TOpenTextFileDialog.

4.1.2.1 TOpenTextFileDialog Members

The following tables list the members exposed by TOpenTextFileDialog.

Protected Methods

	Name	Description
	DoClose (see page 1572)	Generates an OnClose event.
	DoShow (see page 1572)	Generates an OnShow event.

Public Methods

	Name	Description
	Create (see page 1571)	Creates and initializes a TOpenDialog instance.
	Destroy (see page 1571)	Destroys the TOpenDialog and frees its memory.
	Execute (see page 1573)	Displays the file-selection dialog.

Published Properties

	Name	Description
	EncodingIndex (see page 1576)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 1576)	This is Encodings, a member of class TOpenTextFileDialog.

4.1.2.2 TOpenTextFileDialog.Create Constructor

Creates and initializes a TOpenDialog instance.

Pascal

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

C++

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

Description

Create generates a TOpenDialog instance, but the new dialog does not appear on the form at runtime until the Execute method is called.

AOwner is the component that is responsible for freeing the TOpenDialog instance. It becomes the value of the Owner property.

See Also

Execute

Owner

Destroy

4.1.2.3 TOpenTextFileDialog.Destroy Destructor

Destroys the TOpenDialog and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOpenTextFileDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TOpenDialog reference is not nil before calling Destroy.

See Also

System::TObject::Free

Create

4.1.2.4 TOpenTextFileDialog Methods

The methods of the TOpenTextFileDialog class are listed here.

Protected Methods

	Name	Description
	DoClose (see page 1572)	Generates an OnClose event.
	DoShow (see page 1572)	Generates an OnShow event.

Public Methods

	Name	Description
	Execute (see page 1573)	Displays the file-selection dialog.

4.1.2.4.1 TOpenTextFileDialog.DoClose Method

Generates an OnClose event.

Pascal

```
procedure DoClose; override;
```

C++

```
virtual __fastcall DoClose();
```

Description

DoClose is called automatically when the user closes the dialog. Override this method to provide additional processing other than calling the OnClose event handler. For example, to perform shutdown operations that use protected methods of the dialog class, override DoClose rather than writing an OnClose event handler (which can access only public methods).

See Also

OnClose

4.1.2.4.2 TOpenTextFileDialog.DoShow Method

Generates an OnShow event.

Pascal

```
procedure DoShow; override;
```

C++

```
virtual __fastcall DoShow();
```

Description

DoShow is called automatically when the dialog is first displayed. Override this method to provide additional processing other than calling the OnShow event handler. For example, to perform initializations that use protected methods of the dialog class, override DoShow rather than writing an OnShow event handler (which can access only public methods).

C++ Examples:

```
/*
The following example is taken from TOpenPictureDialog. It
calculates the preview rectangle based on the difference
between the client area of the dialog and the static area of
the standard controls.
*/

#include <ExtDlgs.hpp>

class MyOpenPictureDialog : public TOpenPictureDialog
{
private:    // User declarations
    TPanel *FPicturePanel;  // can't use the private versions in TOpenPictureDialog
    TLabel *FPictureLabel;
    TSpeedButton *FPreviewButton;
    TPanel *FPaintPanel;
    TImage *FImageCtrl;
    AnsiString FSavedFilename;
```

```
protected:  // User declarations
    DYNAMIC void __fastcall DoShow(void);
public:     // User declarations
    __fastcall MyOpenPictureDialog(TComponent* Owner);
    virtual bool __fastcall  Execute(HWND ParentWnd){ return TOpenDialog::Execute(ParentWnd); }
};

__fastcall MyOpenPictureDialog::MyOpenPictureDialog(TComponent* Owner)
    : TOpenPictureDialog(Owner)
{
}

#include <memory>       //for STL auto_ptr class

void __fastcall MyOpenPictureDialog::DoShow(void)
{
//  RECT *PreviewRect = new RECT();
  std::auto_ptr<RECT> PreviewRect(new RECT());
  TRect StaticRect;
  FPicturePanel = new TPanel(this); // the owner will clean this up
  FPictureLabel = new TLabel(this); // the owner will clean this up
  FPreviewButton = new TSpeedButton(this); // the owner will clean this up
  FPaintPanel = new TPanel(this); // the owner will clean this up
  FImageCtrl = new TImage(this); // the owner will clean this up
  // Set preview area to entire dialog
  GetClientRect(Handle, PreviewRect.get());
  StaticRect = GetStaticRect();
  // Move preview area to right of static area
  PreviewRect->left = StaticRect.Left + (StaticRect.Right - StaticRect.Left);
  PreviewRect->top += 4;
  FPicturePanel->BoundsRect.Left = PreviewRect->left;
  FPicturePanel->BoundsRect.Right = PreviewRect->right;
  FPicturePanel->BoundsRect.Top = PreviewRect->top;
  FPicturePanel->BoundsRect.Bottom = PreviewRect->bottom;

  FPreviewButton->Left = FPaintPanel->BoundsRect.Right - FPreviewButton->Width - 2;
  FImageCtrl->Picture = NULL;
  FSavedFilename = "";
  FPaintPanel->Caption = "";
  FPicturePanel->ParentWindow = Handle;
//  TOpenDialog::DoShow();   // the above code is equivalent to the inherited version
}

TForm1 *Form1;
MyOpenPictureDialog *OpenPictureDialog1;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (OpenPictureDialog1->Execute(GetParentHandle()))
      Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OpenPictureDialog1 =  new MyOpenPictureDialog(Form1); // the owner will clean this up
}
```

See Also

OnShow

4.1.2.4.3 TOpenTextFileDialog.Execute Method

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
  Memo1.Lines.Clear;
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
else
  Memo1->Clear();
```

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.  Click on the text edit to open the
open dialog box.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
{
This example uses a memo box, two buttons, an open dialog
and a save dialog on a form. When the application runs,
the open dialog and save dialog filters are initialized,
so to open and save *.txt files or files with any extension.
Clicking on the Open/Save button executes the open/save dialog.
An exception is raised when trying to open a file that
does not exist, or when trying to overwrite a file using the
save dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  { execute an open file dialog }
  if OpenDialog1.Execute then
    { first check if file exists }
    if FileExists(OpenDialog1.FileName) then
      { if it exists, load the data into the memo box }
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
    else
```

```
      { otherwise raise an exception }
      raise Exception.Create('File does not exist.');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  { execute a save file dialog }
  if SaveDialog1.Execute then
    { first check if file exists }
    if FileExists(SaveDialog1.FileName) then
      { if it exists, raise an exception }
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      { otherwise, save the memo box lines into the file }
      Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {
  initialize the dialog filters to open/save *.txt files
  and also files with arbitrary extension
  }
  OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
  SaveDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
end;
```

See Also

Files

FileName

4.1.2.5 TOpenTextFileDialog Properties

The properties of the TOpenTextFileDialog class are listed here.

Published Properties

	Name	Description
	EncodingIndex (see page 1576)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 1576)	This is Encodings, a member of class TOpenTextFileDialog.

4.1.2.5.1 TOpenTextFileDialog.EncodingIndex Property

Pascal

```
property EncodingIndex: Integer;
```

C++

```
__property int EncodingIndex;
```

Description

This is EncodingIndex, a member of class TOpenTextFileDialog.

4.1.2.5.2 TOpenTextFileDialog.Encodings Property

Pascal

```
property Encodings: TStrings;
```

C++

```
__property TStrings Encodings;
```

Description

This is Encodings, a member of class TOpenTextFileDialog.

4.1.3 TSavePictureDialog Class

TSavePictureDialog displays a "Save As" dialog for saving graphics files.

Class Hierarchy

Pascal

```
TSavePictureDialog = class(TOpenPictureDialog);
```

C++

```
class TSavePictureDialog : public TOpenPictureDialog;
```

File

ExtDlgs

Description

TSavePictureDialog displays a modal Windows dialog box for selecting file names and saving graphics files. This component is just like TSaveDialog, except that it includes a rectangular preview region. If the selected image can be read by TPicture (see page 2812), it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

Delphi Examples:

```
{
This example uses an image, two buttons, an open picture dialog
and a save picture dialog on a form. Clicking on the Open/Save button
executes the open/save picture dialog.
An exception is raised when trying to open a file that
does not exist, or when trying to overwrite a file using the
save dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  { execute an open picture dialog }
  if OpenPictureDialog1.Execute then
    { first check if file exists }
    if FileExists(OpenPictureDialog1.FileName) then
      { if it exists, load the data into the image component }
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName)
    else
      { otherwise raise an exception }
      raise Exception.Create('File does not exist.');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  { execute a save picture dialog }
  if SavePictureDialog1.Execute then
```

```
    { first check if file exists }
    if FileExists(SavePictureDialog1.FileName) then
      { if it exists, raise an exception }
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      { otherwise, save the image data into the file }
      Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Refresh;
end;
```

See Also

TOpenPictureDialog (see page 1560)

TPicture (see page 2812)

TSaveDialog

4.1.3.1 TSavePictureDialog Members

The following tables list the members exposed by TSavePictureDialog.

Protected Methods

	Name	Description
	DoClose (see page 1562)	Generates an OnClose event.
	DoSelectionChange (see page 1563)	This is DoSelectionChange, a member of class TOpenPictureDialog.
	DoShow (see page 1563)	Generates an OnShow event.
	PreviewClick (see page 1565)	This is PreviewClick, a member of class TOpenPictureDialog.

Protected Properties

	Name	Description
	ImageCtrl (see page 1569)	This is ImageCtrl, a member of class TOpenPictureDialog.
	PictureLabel (see page 1570)	This is PictureLabel, a member of class TOpenPictureDialog.

Public Methods

	Name	Description
	Create (see page 1562)	Creates and initializes a TOpenPictureDialog (see page 1560) instance.
	Execute (see page 1579)	This is the overview for the Execute method overload.

Published Properties

	Name	Description
	Filter (see page 1566)	Determines the file masks (filters) available in the dialog.

4.1.3.2 TSavePictureDialog Methods

The methods of the TSavePictureDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 1579)	This is the overview for the Execute method overload.

4.1.3.2.1 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TSavePictureDialog.Execute () (see page 1579)	Displays the Save As dialog box.
	TSavePictureDialog.Execute (HWND) (see page 1579)	Displays the Save As dialog box.

4.1.3.2.1.1 TSavePictureDialog.Execute Method ()

Displays the Save As dialog box.

Pascal

```
function Execute: Boolean; override;
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the Save As dialog, returning true when the user selects a file name and clicks Save.

Delphi Examples:

```
{
This example uses an image, two buttons, an open picture dialog
and a save picture dialog on a form. Clicking on the Open/Save button
executes the open/save picture dialog.
An exception is raised when trying to open a file that
does not exist, or when trying to overwrite a file using the
save dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  { execute an open picture dialog }
  if OpenPictureDialog1.Execute then
    { first check if file exists }
    if FileExists(OpenPictureDialog1.FileName) then
      { if it exists, load the data into the image component }
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName)
    else
      { otherwise raise an exception }
      raise Exception.Create('File does not exist.');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  { execute a save picture dialog }
  if SavePictureDialog1.Execute then
    { first check if file exists }
    if FileExists(SavePictureDialog1.FileName) then
      { if it exists, raise an exception }
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      { otherwise, save the image data into the file }
      Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Refresh;
end;
```

4.1.4 TSaveTextFileDialog Class

Class Hierarchy

Pascal

```
TSaveTextFileDialog = class(TOpenTextFileDialog);
```

C++

```
class TSaveTextFileDialog : public TOpenTextFileDialog;
```

File

ExtDlgs

Description

This is class ExtDlgs.TSaveTextFileDialog.

4.1.4.1 TSaveTextFileDialog Members

The following tables list the members exposed by TSaveTextFileDialog.

Protected Methods

	Name	Description
	DoClose (see page 1572)	Generates an OnClose event.
	DoShow (see page 1572)	Generates an OnShow event.

Public Methods

	Name	Description
	Create (see page 1571)	Creates and initializes a TOpenDialog instance.
	Destroy (see page 1571)	Destroys the TOpenDialog and frees its memory.
	Execute (see page 1581)	This is the overview for the Execute method overload.

Published Properties

	Name	Description
	EncodingIndex (see page 1576)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 1576)	This is Encodings, a member of class TOpenTextFileDialog.

4.1.4.2 TSaveTextFileDialog Methods

The methods of the TSaveTextFileDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 1581)	This is the overview for the Execute method overload.

4.1.4.2.1 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TSaveTextFileDialog.Execute () (see page 1581)	Displays the dialog box.
	TSaveTextFileDialog.Execute (HWND) (see page 1581)	Displays the file-selection dialog.

4.1.4.2.1.1 TSaveTextFileDialog.Execute Method ()

Displays the dialog box.

Pascal

```
function Execute: Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
```

Description

TCommonDialog introduces the abstract Execute method, which descendant classes use to open the dialog instance at runtime. Generally, Execute returns true when the user makes a selection and clicks OK, and returns false when the user closes the dialog without making a selection.

See Also

TaskModalDialog

4.1.4.2.1.2 TSaveTextFileDialog.Execute Method (HWND)

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
  Memo1.Lines.Clear;
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
else
  Memo1->Clear();
```

C++ Examples:

```
/*
```

```
  The following code allows the user to use a dialog box to
  redefine the icon for the application at run time. When the
  user clicks Button1, OpenDialog1 executes and the user
  specifies an icon filename. The Icon is then assigned to the
  application.  At runtime, click Button1 to select the icon,
  then minimize the application to see the icon.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
    OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
    OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
    OpenPictureDialog1->Options.Clear();
    OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
    while (true)
    {
      if (OpenPictureDialog1->Execute())
      {
        if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
        {
          Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
          break;
        }
        else // reset Options to remove ofExtensionDifferent
        {
          OpenPictureDialog1->Options.Clear();
          OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
        }
      }
      else // user cancelled
        break;
    }
  }
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
```

```
    else { User cancelled }
      Done := True;
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.  Click on the text edit to open the
  open dialog box.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  This example uses a memo box, two buttons, an open dialog
  and a save dialog on a form. When the application runs,
  the open dialog and save dialog filters are initialized,
  so to open and save *.txt files or files with any extension.
  Clicking on the Open/Save button executes the open/save dialog.
  An exception is raised when trying to open a file that
  does not exist, or when trying to overwrite a file using the
  save dialog.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    { execute an open file dialog }
    if OpenDialog1.Execute then
      { first check if file exists }
      if FileExists(OpenDialog1.FileName) then
        { if it exists, load the data into the memo box }
        Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
      else
        { otherwise raise an exception }
        raise Exception.Create('File does not exist.');
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    { execute a save file dialog }
    if SaveDialog1.Execute then
      { first check if file exists }
      if FileExists(SaveDialog1.FileName) then
        { if it exists, raise an exception }
        raise Exception.Create('File already exists. Cannot overwrite.')
      else
        { otherwise, save the memo box lines into the file }
        Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    {
    initialize the dialog filters to open/save *.txt files
    and also files with arbitrary extension
    }
    OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
```

```
    SaveDialog1.Filter := 'Text files (*.txt)|*.TXT|Any file (*.*)|*.*';
  end;
```

See Also

Files

FileName

4.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	StandardEncodingFromName (see page 1585)	This is function ExtDlgs.StandardEncodingFromName.

4.2.1 ExtDlgs.StandardEncodingFromName Function

Pascal

```
function StandardEncodingFromName(const Name: string): TEncoding;
```

C++

```
TEncoding StandardEncodingFromName(const UnicodeString Name);
```

File

ExtDlgs

Description

This is function ExtDlgs.StandardEncodingFromName.

4.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DefaultEncodingNames (see page 1586)	This is constant ExtDlgs.DefaultEncodingNames.

4.3.1 ExtDlgs.DefaultEncodingNames Constant

Pascal

```
DefaultEncodingNames: array[0..5] of string = (SANSIEncoding, SASCIIEncoding,
SUnicodeEncoding, SBigEndianEncoding, SUTF8Encoding, SUTF7Encoding);
```

C++

```
array[0..5] of string DefaultEncodingNames = (SANSIEncoding, SASCIIEncoding, SUnicodeEncoding,
SBigEndianEncoding, SUTF8Encoding, SUTF7Encoding);
```

File

ExtDlgs

Description

This is constant ExtDlgs.DefaultEncodingNames.

5 FileCtrl Namespace

This is namespace FileCtrl.

Topics

Name	Description
Classes (see page 1589)	The following table lists classes in this documentation.
Functions (see page 1879)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1883)	The following table lists structs, records, enums in this documentation.
Types (see page 1887)	The following table lists types in this documentation.
Constants (see page 1889)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDirectoryListBox (see page 1589)	TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.
	TDriveComboBox (see page 1668)	TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.
	TFileListBox (see page 1734)	TFileListBox is a specialized list box that lists all the files in a specified directory.
	TFilterComboBox (see page 1813)	TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

Constants

Name	Description
WNTYPE_DRIVE (see page 1889)	This is constant FileCtrl.WNTYPE_DRIVE.

Functions

	Name	Description
	DirectoryExists (see page 1879)	Determines whether a specified directory exists.
	ForceDirectories (see page 1879)	Creates a new directory, including the creation of parent directories as needed.
	MinimizeName (see page 1880)	Shortens a fully qualified path name so that it can be drawn with a specified length limit.
	ProcessPath (see page 1881)	Parses a file name into its constituent parts.
	SelectDirectory (see page 1881)	Brings up a dialog to allow the user to enter a directory name.
	SelectDirectory (see page 1881)	Brings up a dialog to allow the user to enter a directory name.

Structs, Records, Enums

	Name	Description
	TDriveType (see page 1883)	This is record FileCtrl.TDriveType.
	TFileAttr (see page 1883)	TFileAttr and TFileType (see page 1887) determine which files a file list box displays.
	TSelectDirExtOpt (see page 1884)	This is record FileCtrl.TSelectDirExtOpt.
	TSelectDirOpt (see page 1885)	TSelectDirOpt and TSelectDirOpts (see page 1888) determine how a directory selection dialog responds when the user enters a nonexistent directory.
	TTextCase (see page 1886)	TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

Types

Name	Description
TFileType (see page 1887)	TFileType describes possible combinations of file attributes.
TSelectDirExtOpts (see page 1887)	This is type FileCtrl.TSelectDirExtOpts.
TSelectDirOpts (see page 1888)	This is type FileCtrl.TSelectDirOpts.

5.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDirectoryListBox (see page 1589)	TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.
	TDriveComboBox (see page 1668)	TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.
	TFileListBox (see page 1734)	TFileListBox is a specialized list box that lists all the files in a specified directory.
	TFilterComboBox (see page 1813)	TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

5.1.1 TDirectoryListBox Class

TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.

Class Hierarchy

Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TDirectoryListBox = class(TCustomListBox);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TDirectoryListBox : public TCustomListBox;
```

File

FileCtrl

Description

TDirectoryListBox implements the generic behavior introduced in TCustomListBox. TDirectoryListBox publishes many of the properties inherited from TCustomListBox and introduces new behavior to let the user change directories or drives.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
```

```
  }
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

See Also

TDriveComboBox (see page 1668)

TFileListBox (see page 1734)

TFilterComboBox (see page 1813)

TCustomListBox

5.1.1.1 TDirectoryListBox Members

The following tables list the members exposed by TDirectoryListBox.

Protected Fields

	Name	Description
	ClosedBMP (see page 1593)	This is ClosedBMP, a member of class TDirectoryListBox.
	CurrentBMP (see page 1593)	This is CurrentBMP, a member of class TDirectoryListBox.
	FDirectory (see page 1594)	This is FDirectory, a member of class TDirectoryListBox.
	FOnChange (see page 1594)	This is FOnChange, a member of class TDirectoryListBox.
	OpenedBMP (see page 1594)	This is OpenedBMP, a member of class TDirectoryListBox.

Protected Methods

	Name	Description
	BuildList (see page 1595)	This is BuildList, a member of class TDirectoryListBox.
	Change (see page 1595)	This is Change, a member of class TDirectoryListBox.
	CreateWnd (see page 1595)	Creates a Windows control corresponding to the list box component.
	DblClick (see page 1596)	OnDblClick event dispatcher.
	DrawItem (see page 1597)	Generates an OnDrawItem event.
	KeyPress (see page 1598)	Responds when the user presses a key.
	Notification (see page 1598)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 1599)	This is ReadBitmaps, a member of class TDirectoryListBox.
	ReadDirectoryNames (see page 1599)	This is ReadDirectoryNames, a member of class TDirectoryListBox.

Public Methods

	Name	Description
	Create (see page 1592)	Creates and initializes a directory list box.
	Destroy (see page 1593)	Disposes of an instance of a directory list box.
	DisplayCase (see page 1596)	Preserves or converts the case of the string passed to it.
	FileCompareText (see page 1597)	Compares two strings, taking into account the case-sensitivity of the file system.
	GetItemPath (see page 1597)	This is GetItemPath, a member of class TDirectoryListBox.
	OpenCurrent (see page 1598)	Opens the selected directory.
	Update (see page 1600)	Refreshes the directory list.

Public Properties

	Name	Description
	CaseSensitive (see page 1608)	Indicates whether the system supports case-sensitive directory information.
	Directory (see page 1611)	Determines the current directory for the directory list box control.
	Drive (see page 1617)	Determines the drive for which the list box displays the directory structure.
	PreserveCase (see page 1631)	Indicates whether the file system preserves the case of file names when writing to disk.

Published Events

	Name	Description
	OnChange (see page 1636)	Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.
	OnClick (see page 1638)	Occurs when the user clicks the control.
	OnContextPopup (see page 1639)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1640)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1640)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1643)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 1647)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1648)	Occurs when a control receives the input focus.
	OnExit (see page 1650)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1653)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1657)	Occurs when a key is pressed.
	OnKeyUp (see page 1658)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1659)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1660)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1663)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1663)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1663)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1665)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 1668)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1602)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1605)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1606)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 1606)	Specifies which edges of the control are beveled.
	BevelInner (see page 1606)	Specifies the cut of the inner bevel.
	BevelKind (see page 1607)	Specifies the control's bevel style.

	BevelOuter (see page 1607)	Specifies the cut of the outer bevel.
	Color (see page 1608)	Specifies the background color of the control.
	Columns (see page 1609)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 1610)	Specifies the size constraints for the control.
	Ctl3D (see page 1610)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirLabel (see page 1612)	Provides a simple way to display the current directory as the caption of a label control.
	DoubleBuffered (see page 1613)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1615)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1615)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1618)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 1618)	Provides a simple way to connect a directory list box with a file list box.
	Font (see page 1621)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1622)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1622)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 1623)	Determines whether the list box displays the partial items.
	ItemHeight (see page 1623)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 1624)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1624)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1625)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1625)	Determines where a control looks for its font information.
	ParentShowHint (see page 1625)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1626)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1631)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1633)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1633)	Determines if the user can tab to a control.
	Visible (see page 1634)	Determines whether the component appears onscreen.

5.1.1.2 TDirectoryListBox.Create Constructor

Creates and initializes a directory list box.

Pascal

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

C++

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

Description

Call Create to instantiate a directory list box at runtime. For directory list boxes created at design time, Create is called automatically.

See Also

Create

Sorted

Style

ParentWindow

5.1.1.3 TDirectoryListBox.Destroy Destructor

Disposes of an instance of a directory list box.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDirectoryListBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free checks that the list box reference is not nil before it calls Destroy.

Destroy frees the bitmaps used for the Open, Close and Current folders, and then calls the inherited Destroy method.

See Also

Destroy

5.1.1.4 TDirectoryListBox Fields

The fields of the TDirectoryListBox class are listed here.

Protected Fields

	Name	Description
	ClosedBMP (see page 1593)	This is ClosedBMP, a member of class TDirectoryListBox.
	CurrentBMP (see page 1593)	This is CurrentBMP, a member of class TDirectoryListBox.
	FDirectory (see page 1594)	This is FDirectory, a member of class TDirectoryListBox.
	FOnChange (see page 1594)	This is FOnChange, a member of class TDirectoryListBox.
	OpenedBMP (see page 1594)	This is OpenedBMP, a member of class TDirectoryListBox.

5.1.1.4.1 TDirectoryListBox.ClosedBMP Field

Pascal

```
ClosedBMP: TBitmap;
```

C++

```
TBitmap ClosedBMP;
```

Description

This is ClosedBMP, a member of class TDirectoryListBox.

5.1.1.4.2 TDirectoryListBox.CurrentBMP Field

Pascal

```
CurrentBMP: TBitmap;
```

C++

```
TBitmap CurrentBMP;
```

Description

This is CurrentBMP, a member of class TDirectoryListBox.

5.1.1.4.3 TDirectoryListBox.FDirectory Field

Pascal

```
FDirectory: string;
```

C++

```
UnicodeString FDirectory;
```

Description

This is FDirectory, a member of class TDirectoryListBox.

5.1.1.4.4 TDirectoryListBox.FOnChange Field

Pascal

```
FOnChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnChange;
```

Description

This is FOnChange, a member of class TDirectoryListBox.

5.1.1.4.5 TDirectoryListBox.OpenedBMP Field

Pascal

```
OpenedBMP: TBitmap;
```

C++

```
TBitmap OpenedBMP;
```

Description

This is OpenedBMP, a member of class TDirectoryListBox.

5.1.1.5 TDirectoryListBox Methods

The methods of the TDirectoryListBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 1595)	This is BuildList, a member of class TDirectoryListBox.
	Change (see page 1595)	This is Change, a member of class TDirectoryListBox.
	CreateWnd (see page 1595)	Creates a Windows control corresponding to the list box component.
	DblClick (see page 1596)	OnDblClick event dispatcher.
	DrawItem (see page 1597)	Generates an OnDrawItem event.
	KeyPress (see page 1598)	Responds when the user presses a key.
	Notification (see page 1598)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 1599)	This is ReadBitmaps, a member of class TDirectoryListBox.

	ReadDirectoryNames (see page 1599)	This is ReadDirectoryNames, a member of class TDirectoryListBox.

Public Methods

	Name	Description
	DisplayCase (see page 1596)	Preserves or converts the case of the string passed to it.
	FileCompareText (see page 1597)	Compares two strings, taking into account the case-sensitivity of the file system.
	GetItemPath (see page 1597)	This is GetItemPath, a member of class TDirectoryListBox.
	OpenCurrent (see page 1598)	Opens the selected directory.
	Update (see page 1600)	Refreshes the directory list.

5.1.1.5.1 TDirectoryListBox.BuildList Method

Pascal

```
procedure BuildList; virtual;
```

C++

```
virtual __fastcall BuildList();
```

Description

This is BuildList, a member of class TDirectoryListBox.

5.1.1.5.2 TDirectoryListBox.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TDirectoryListBox.

5.1.1.5.3 TDirectoryListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns

CreateHandle

CreateParams

CreateWindowHandle

CreateWnd

ItemIndex

Items

TabWidth

TopIndex

5.1.1.5.4 TDirectoryListBox.DblClick Method

OnDblClick event dispatcher.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click

OnDblClick

5.1.1.5.5 TDirectoryListBox.DisplayCase Method

Preserves or converts the case of the string passed to it.

Pascal

```
function DisplayCase(const S: String): String;
```

C++

```
__fastcall UnicodeString DisplayCase(const UnicodeString S);
```

Description

DisplayCase converts S to all-lower-case (using the currently installed language driver) if neither PreserveCase (see page 1631) nor CaseSensitive (see page 1608) is true. Otherwise, it returns S unmodified.

See Also

CaseSensitive (see page 1608)

PreserveCase (see page 1631)

5.1.1.5.6 TDirectoryListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight

MeasureItem

OnDrawItem

OnMeasureItem

Style

5.1.1.5.7 TDirectoryListBox.FileCompareText Method

Compares two strings, taking into account the case-sensitivity of the file system.

Pascal

```
function FileCompareText(const A: String; const B: String): Integer;
```

C++

```
__fastcall int FileCompareText(const UnicodeString A, const UnicodeString B);
```

Description

If CaseSensitive (see page 1608) is false, FileCompareText compares the strings passed to it case-insensitively. Otherwise, it uses AnsiCompareStr to compare the strings.

See Also

AnsiCompareStr

CaseSensitive (see page 1608)

5.1.1.5.8 TDirectoryListBox.GetItemPath Method

Pascal

```
function GetItemPath(Index: Integer): string;
```

C++

```
__fastcall UnicodeString GetItemPath(int Index);
```

Description

This is GetItemPath, a member of class TDirectoryListBox.

5.1.1.5.9 TDirectoryListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress

5.1.1.5.10 TDirectoryListBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.1.5.11 TDirectoryListBox.OpenCurrent Method

Opens the selected directory.

Pascal

```
procedure OpenCurrent;
```

C++

```
__fastcall OpenCurrent();
```

Description

OpenCurrent opens the directory selected in the directory list box, as if the user had double-clicked the directory.

C++ Examples:

```
/*
This example uses a directory list box, a button, and a
label on a form. When the user selects a directory in the
directory list box and clicks the button, the selected
directory opens, and the path of the second directory
displayed in the list box appears as the caption of the
label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DirectoryListBox1->OpenCurrent();
  Label1->Caption = DirectoryListBox1->GetItemPath(1);
}
```

Delphi Examples:

```
{
This example uses a directory list box, a button, and a
label on a form. When the user selects a directory in the
directory list box and clicks the button, the selected
directory opens, and the path of the second directory
displayed in the list box appears as the caption of the
label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DirectoryListBox1.OpenCurrent;
  Label1.Caption := DirectoryListBox1.GetItemPath(1);
end;
```

5.1.1.5.12 TDirectoryListBox.ReadBitmaps Method

Pascal

```
procedure ReadBitmaps; virtual;
```

C++

```
virtual __fastcall ReadBitmaps();
```

Description

This is ReadBitmaps, a member of class TDirectoryListBox.

5.1.1.5.13 TDirectoryListBox.ReadDirectoryNames Method

Pascal

```
function ReadDirectoryNames(const ParentDirectory: string; DirectoryList: TStringList):
Integer;
```

C++

```
__fastcall int ReadDirectoryNames(const UnicodeString ParentDirectory, TStringList
```

```
DirectoryList);
```

Description

This is ReadDirectoryNames, a member of class TDirectoryListBox.

5.1.1.5.14 TDirectoryListBox.Update Method

Refreshes the directory list.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update to refresh the directory list after new directories have been created or existing directories deleted.

C++ Examples:

```
/*
The following sample code sets the directory of
DirectoryListBox1 to c:\\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
c:\\WINDOWS\\mytemp is added, but note that it is not updated
in DirectoryListBox1 until Button2 is pressed and Update is
called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current directory to see the magic happen.  Also,
capitalization matters in C++ and the file spearators must
be "\\".
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ForceDirectories(Edit1->Text);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  DirectoryListBox1->Update();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  DirectoryListBox1->Directory = "c:\\WINDOWS";
}
```

Delphi Examples:

```
{
The following sample code sets the directory of
DirectoryListBox1 to C:\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
mytemp is added to C:\WINDOWS, but note that it is not
updated in DirectoryListBox1 until Button2 is pressed
and Update is called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current path to see the magic happen.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  MkDir(Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DirectoryListBox1.Directory := 'c:\WINDOWS';
end;
```

5.1.1.6 TDirectoryListBox Properties

The properties of the TDirectoryListBox class are listed here.

Public Properties

	Name	Description
R	CaseSensitive (see page 1608)	Indicates whether the system supports case-sensitive directory information.
	Directory (see page 1611)	Determines the current directory for the directory list box control.
	Drive (see page 1617)	Determines the drive for which the list box displays the directory structure.
R	PreserveCase (see page 1631)	Indicates whether the file system preserves the case of file names when writing to disk.

Published Properties

	Name	Description
	Align (see page 1602)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1605)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1606)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 1606)	Specifies which edges of the control are beveled.
	BevelInner (see page 1606)	Specifies the cut of the inner bevel.
	BevelKind (see page 1607)	Specifies the control's bevel style.
	BevelOuter (see page 1607)	Specifies the cut of the outer bevel.
	Color (see page 1608)	Specifies the background color of the control.
	Columns (see page 1609)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 1610)	Specifies the size constraints for the control.
	Ctl3D (see page 1610)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirLabel (see page 1612)	Provides a simple way to display the current directory as the caption of a label control.
	DoubleBuffered (see page 1613)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1615)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1615)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1618)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 1618)	Provides a simple way to connect a directory list box with a file list box.
	Font (see page 1621)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1622)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1622)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 1623)	Determines whether the list box displays the partial items.
	ItemHeight (see page 1623)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 1624)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1624)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 1625)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1625)	Determines where a control looks for its font information.
	ParentShowHint (see page 1625)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1626)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1631)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1633)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1633)	Determines if the user can tab to a control.
	Visible (see page 1634)	Determines whether the component appears onscreen.

5.1.1.6.1 TDirectoryListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
```

```
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
```

```
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
```

```
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

5.1.1.6.2 TDirectoryListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

5.1.1.6.3 TDirectoryListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

5.1.1.6.4 TDirectoryListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

5.1.1.6.5 TDirectoryListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

5.1.1.6.6 TDirectoryListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

5.1.1.6.7 TDirectoryListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

5.1.1.6.8 TDirectoryListBox.CaseSensitive Property

Indicates whether the system supports case-sensitive directory information.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Read CaseSensitive to ascertain whether the current file system supports case-sensitive file names. CaseSensitive returns true if it does and false if it does not.

See Also

DisplayCase (see page 1596)

FileCompareText (see page 1597)

PreserveCase (see page 1631)

5.1.1.6.9 TDirectoryListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
```

```
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

5.1.1.6.10 TDirectoryListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns;
```

C++

```
__property Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight

Width

5.1.1.6.11 TDirectoryListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

5.1.1.6.12 TDirectoryListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

5.1.1.6.13 TDirectoryListBox.Directory Property

Determines the current directory for the directory list box control.

Pascal

```
property Directory: string;
```

C++

```
__property UnicodeString Directory;
```

Description

The directory list box displays the value of the Directory property as the current directory in the list box. The example shows how a directory list box and a file list box can work together through their Directory properties.

If the form contains a file list box, you can set the FileList (see page 1618) property to have the file list box automatically display the files from the directory specified by the Directory property.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

DirLabel (see page 1612)

Drive (see page 1617)

FileList (see page 1618)

5.1.1.6.14 TDirectoryListBox.DirLabel Property

Provides a simple way to display the current directory as the caption of a label control.

Pascal

```
property DirLabel: TLabel;
```

C++

```
__property TLabel DirLabel;
```

Description

Use DirLabel to specify a label. When the current directory changes in the directory list box, the change is reflected in the caption of the label.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Caption

Directory (see page 1611)

DirList (see page 1685)

FileEdit (see page 1763)

FileList (see page 1618)

5.1.1.6.15 TDirectoryListBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
```

```
    Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

5.1.1.6.16 TDirectoryListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

5.1.1.6.17 TDirectoryListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
```

```
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

5.1.1.6.18 TDirectoryListBox.Drive Property

Determines the drive for which the list box displays the directory structure.

Pascal

```
property Drive: Char;
```

C++

```
__property Char Drive;
```

Description

When the value of Drive changes, the Directory (see page 1611) value changes to the current directory on the specified drive.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
```

```
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Directory (see page 1611)

DirList (see page 1685)

Text

5.1.1.6.19 TDirectoryListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

5.1.1.6.20 TDirectoryListBox.FileList Property

Provides a simple way to connect a directory list box with a file list box.

Pascal

```
property FileList: TFileListBox;
```

C++

```
__property TFileListBox FileList;
```

Description

Use FileList to specify a file list box in which to display the files of the current directory.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
```

```
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
```

```
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    FileListBox1.FileEdit := Edit1;
    FilterComboBox1.FileList := FileListBox1;
    DirectoryListBox1.FileList := FileListBox1;
    DirectoryListBox1.DirLabel := Label1;
    DriveComboBox1.DirList := DirectoryListBox1;
    Button1.Default := True;
  end;
  {
  This example shows how to use a splitter to divide a form
  into resizable panes.  All objects on the form are created
  dynamically in the OnCreate event handler of the form. To
  run this example, add ExtCtrls and FileCtrl to the uses
  clause of the form's implementation section.
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

DirLabel (see page 1612)

Directory (see page 1611)

DirList (see page 1685)

FileEdit (see page 1763)

Filter (see page 1833)

Mask (see page 1768)

Mask (see page 1837)

5.1.1.6.21 TDirectoryListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

5.1.1.6.22 TDirectoryListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

5.1.1.6.23 TDirectoryListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

5.1.1.6.24 TDirectoryListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight;
```

C++

```
__property IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight

Items

Style

Using TDBListBox and TDBComboBox

5.1.1.6.25 TDirectoryListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight;
```

C++

```
__property ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem

IntegralHeight

Items

MeasureItem

OnDrawItem

OnMeasureItem

Style

Using TDBListBox and TDBComboBox

5.1.1.6.26 TDirectoryListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

5.1.1.6.27 TDirectoryListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

5.1.1.6.28 TDirectoryListBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

5.1.1.6.29 TDirectoryListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

5.1.1.6.30 TDirectoryListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

5.1.1.6.31 TDirectoryListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
```

```
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

5.1.1.6.32 TDirectoryListBox.PreserveCase Property

Indicates whether the file system preserves the case of file names when writing to disk.

Pascal

```
property PreserveCase: Boolean;
```

C++

```
__property Boolean PreserveCase;
```

Description

Read PreserveCase to ascertain whether the current file system preserves the case of file names. PreserveCase returns true if it does and false if it does not.

See Also

CaseSensitive (see page 1608)

DisplayCase (see page 1596)

5.1.1.6.33 TDirectoryListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

5.1.1.6.34 TDirectoryListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

5.1.1.6.35 TDirectoryListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

5.1.1.6.36 TDirectoryListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

5.1.1.7 TDirectoryListBox Events

The events of the TDirectoryListBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 1636)	Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.
	OnClick (see page 1638)	Occurs when the user clicks the control.

	OnContextPopup (see page 1639)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1640)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1640)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1643)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 1647)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1648)	Occurs when a control receives the input focus.
	OnExit (see page 1650)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1653)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1657)	Occurs when a key is pressed.
	OnKeyUp (see page 1658)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1659)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1660)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1663)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1663)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1663)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1665)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 1668)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

5.1.1.7.1 TDirectoryListBox.OnChange Event

Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to perform an action whenever a new directory is selected.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
```

```
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

5.1.1.7.2 TDirectoryListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

5.1.1.7.3 TDirectoryListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

5.1.1.7.4 TDirectoryListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

5.1.1.7.5 TDirectoryListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

5.1.1.7.6 TDirectoryListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

5.1.1.7.7 TDirectoryListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

5.1.1.7.8 TDirectoryListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

5.1.1.7.9 TDirectoryListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
```

```
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

5.1.1.7.10 TDirectoryListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
```

```
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
```

```
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

5.1.1.7.11 TDirectoryListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

5.1.1.7.12 TDirectoryListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

5.1.1.7.13 TDirectoryListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

5.1.1.7.14 TDirectoryListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse

button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
```

```
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

5.1.1.7.15 TDirectoryListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

5.1.1.7.16 TDirectoryListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

5.1.1.7.17 TDirectoryListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

5.1.1.7.18 TDirectoryListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

5.1.1.7.19 TDirectoryListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

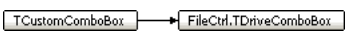
OnDragOver

OnDragDrop

5.1.2 TDriveComboBox Class

TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.

Class Hierarchy



Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TDriveComboBox = class(TCustomComboBox);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TDriveComboBox : public TCustomComboBox;
```

File

FileCtrl

Description

Add TDriveComboBox to a form to allow users to select a drive. Use TDriveComboBox along with TFileListBox (see page 1734), TFilterComboBox (see page 1813), and TDirectoryListBox (see page 1589) to add full file selection capabilities to a form.

To add a standard Windows 95 file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

See Also

TDirectoryListBox (see page 1589)

TFileListBox (see page 1734)

TFilterComboBox (see page 1813)

TOpenDialog

TSaveDialog

5.1.2.1 TDriveComboBox Members

The following tables list the members exposed by TDriveComboBox.

Protected Fields

	Name	Description
	CDROMBMP (see page 1672)	This is CDROMBMP, a member of class TDriveComboBox.
	FixedBMP (see page 1672)	This is FixedBMP, a member of class TDriveComboBox.
	FloppyBMP (see page 1672)	This is FloppyBMP, a member of class TDriveComboBox.
	NetworkBMP (see page 1672)	This is NetworkBMP, a member of class TDriveComboBox.
	RAMBMP (see page 1673)	This is RAMBMP, a member of class TDriveComboBox.

Protected Methods

	Name	Description
	BuildList (see page 1673)	This is BuildList, a member of class TDriveComboBox.
	Click (see page 1673)	Respond to user click.
	CreateWnd (see page 1674)	Creates the Windows control that represents the combo box.
	DrawItem (see page 1674)	Generates an OnDrawItem event.
	Notification (see page 1675)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
	Create (see page 1671)	Creates and initializes an instance of TDriveComboBox (see page 1668).
	Destroy (see page 1671)	Destroys an instance of TDriveComboBox (see page 1668).

Public Properties

	Name	Description
	Drive (see page 1690)	Indicates the currently selected drive displayed in the edit portion of the combo box.
	Text (see page 1704)	Contains a text string associated with the control.

Published Events

	Name	Description
	OnChange (see page 1708)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 1710)	Occurs when the user clicks the control.
	OnContextPopup (see page 1712)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1712)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1713)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1716)	Occurs when the user drags an object over a control.
	OnDropDown (see page 1719)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 1720)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1721)	Occurs when a control receives the input focus.
	OnExit (see page 1724)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1727)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1730)	Occurs when a key is pressed.
	OnKeyUp (see page 1731)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 1733)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1733)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 1734)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1676)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1680)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1680)	Positions to matching list items as you type.
	AutoDropDown (see page 1681)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 1682)	Specifies which edges of the control are beveled.
	BevelInner (see page 1682)	Specifies the cut of the inner bevel.
	BevelKind (see page 1683)	Specifies the control's bevel style.
	BevelOuter (see page 1683)	Specifies the cut of the outer bevel.
	Color (see page 1684)	Specifies the background color of the control.
	Constraints (see page 1684)	Specifies the size constraints for the control.
	Ctl3D (see page 1685)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirList (see page 1685)	Connects the drive combo box to a directory list box.
	DoubleBuffered (see page 1687)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1688)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1689)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1692)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1692)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1693)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1694)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 1694)	Determines where a control looks for its color information.

	ParentCtl3D (see page 1695)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1695)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1695)	Determines where a control looks for its font information.
	ParentShowHint (see page 1696)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1696)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1701)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1703)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1704)	Determines if the user can tab to a control.
	TextCase (see page 1705)	Determines if the volume name in the Text (see page 1704) property appears in uppercase or lowercase.
	Visible (see page 1706)	Determines whether the component appears onscreen.

5.1.2.2 TDriveComboBox.Create Constructor

Creates and initializes an instance of TDriveComboBox (see page 1668).

Pascal

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

C++

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

Description

Call Create to create a TDriveComboBox (see page 1668) object at runtime. Drive (see page 1690) combo boxes placed on a form at design time are created automatically.

After allocating the memory for the combo box, Create fills the combo box with all the drives on the system, and initializes the Drive (see page 1690) property to the current drive.

See Also

Drive (see page 1690)

Create

5.1.2.3 TDriveComboBox.Destroy Destructor

Destroys an instance of TDriveComboBox (see page 1668).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDriveComboBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the drive combo box reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

5.1.2.4 TDriveComboBox Fields

The fields of the TDriveComboBox class are listed here.

Protected Fields

	Name	Description
	CDROMBMP (see page 1672)	This is CDROMBMP, a member of class TDriveComboBox.
	FixedBMP (see page 1672)	This is FixedBMP, a member of class TDriveComboBox.
	FloppyBMP (see page 1672)	This is FloppyBMP, a member of class TDriveComboBox.
	NetworkBMP (see page 1672)	This is NetworkBMP, a member of class TDriveComboBox.
	RAMBMP (see page 1673)	This is RAMBMP, a member of class TDriveComboBox.

5.1.2.4.1 TDriveComboBox.CDROMBMP Field

Pascal

```
CDROMBMP: TBitmap;
```

C++

```
TBitmap CDROMBMP;
```

Description

This is CDROMBMP, a member of class TDriveComboBox.

5.1.2.4.2 TDriveComboBox.FixedBMP Field

Pascal

```
FixedBMP: TBitmap;
```

C++

```
TBitmap FixedBMP;
```

Description

This is FixedBMP, a member of class TDriveComboBox.

5.1.2.4.3 TDriveComboBox.FloppyBMP Field

Pascal

```
FloppyBMP: TBitmap;
```

C++

```
TBitmap FloppyBMP;
```

Description

This is FloppyBMP, a member of class TDriveComboBox.

5.1.2.4.4 TDriveComboBox.NetworkBMP Field

Pascal

```
NetworkBMP: TBitmap;
```

C++

```
TBitmap NetworkBMP;
```

Description

This is NetworkBMP, a member of class TDriveComboBox.

5.1.2.4.5 TDriveComboBox.RAMBMP Field

Pascal

```
RAMBMP: TBitmap;
```

C++

```
TBitmap RAMBMP;
```

Description

This is RAMBMP, a member of class TDriveComboBox.

5.1.2.5 TDriveComboBox Methods

The methods of the TDriveComboBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 1673)	This is BuildList, a member of class TDriveComboBox.
	Click (see page 1673)	Respond to user click.
	CreateWnd (see page 1674)	Creates the Windows control that represents the combo box.
	DrawItem (see page 1674)	Generates an OnDrawItem event.
	Notification (see page 1675)	Responds to notifications that components are being created or destroyed.

5.1.2.5.1 TDriveComboBox.BuildList Method

Pascal

```
procedure BuildList; virtual;
```

C++

```
virtual __fastcall BuildList();
```

Description

This is BuildList, a member of class TDriveComboBox.

5.1.2.5.2 TDriveComboBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

5.1.2.5.3 TDriveComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle

5.1.2.5.4 TDriveComboBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

When Style is csOwnerDrawFixed or csOwnerDrawVariable, DrawItem is called whenever an item in the list needs to be drawn. If an OnDrawItem event handler is assigned, DrawItem generates an OnDrawItem event, passing the Index, Rect, and State parameters to the event handler. Otherwise, OnDrawItem fills the rectangle passed in Rect and draws any text associated with

the indexed item, ignoring the State parameter.

Override DrawItem to provide class-specific drawing code for the items in the list.

See Also

OnDrawItem

Style

5.1.2.5.5 TDriveComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.2.6 TDriveComboBox Properties

The properties of the TDriveComboBox class are listed here.

Public Properties

	Name	Description
	Drive (see page 1690)	Indicates the currently selected drive displayed in the edit portion of the combo box.
	Text (see page 1704)	Contains a text string associated with the control.

Published Properties

	Name	Description
	Align (see page 1676)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1680)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1680)	Positions to matching list items as you type.
	AutoDropDown (see page 1681)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 1682)	Specifies which edges of the control are beveled.
	BevelInner (see page 1682)	Specifies the cut of the inner bevel.

	BevelKind (see page 1683)	Specifies the control's bevel style.
	BevelOuter (see page 1683)	Specifies the cut of the outer bevel.
	Color (see page 1684)	Specifies the background color of the control.
	Constraints (see page 1684)	Specifies the size constraints for the control.
	Ctl3D (see page 1685)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirList (see page 1685)	Connects the drive combo box to a directory list box.
	DoubleBuffered (see page 1687)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1688)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1689)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1692)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1692)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1693)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1694)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 1694)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1695)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1695)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1695)	Determines where a control looks for its font information.
	ParentShowHint (see page 1696)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1696)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1701)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1703)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1704)	Determines if the user can tab to a control.
	TextCase (see page 1705)	Determines if the volume name in the Text (see page 1704) property appears in uppercase or lowercase.
	Visible (see page 1706)	Determines whether the component appears onscreen.

5.1.2.6.1 TDriveComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the

edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

5.1.2.6.2 TDriveComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

5.1.2.6.3 TDriveComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
```

```
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

AutoCompleteDelay

5.1.2.6.4 TDriveComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown;
```

C++

```
__property AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete

OnDropDown

DroppedDown

5.1.2.6.5 TDriveComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

5.1.2.6.6 TDriveComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

5.1.2.6.7 TDriveComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

5.1.2.6.8 TDriveComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

5.1.2.6.9 TDriveComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

5.1.2.6.10 TDriveComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

5.1.2.6.11 TDriveComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

5.1.2.6.12 TDriveComboBox.DirList Property

Connects the drive combo box to a directory list box.

Pascal

```
property DirList: TDirectoryListBox;
```

C++

```
__property TDirectoryListBox DirList;
```

Description

Set DirList to a directory list box control that should update when the user changes the drive. When a new drive is selected in the drive combo box, the specified directory list box updates to display the directory structure and the current directory on the new drive.

Note: If the form contains a file list box as well, it can be updated automatically using the FileList property of the directory list box object.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

TDirectoryListBox (see page 1589)

TFileListBox (see page 1734)

5.1.2.6.13 TDriveComboBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

5.1.2.6.14 TDriveComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

5.1.2.6.15 TDriveComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}
```

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

5.1.2.6.16 TDriveComboBox.Drive Property

Indicates the currently selected drive displayed in the edit portion of the combo box.

Pascal

```
property Drive: Char;
```

C++

```
__property Char Drive;
```

Description

Set Drive to initialize the combo box to a particular drive. Read Drive to obtain the drive that was selected by the user.

When the user uses the drive combo box to select a new drive, the selected drive becomes the value of the Drive property. The value of the Text (see page 1704) property also changes to the new volume name when the Drive property value changes.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text

5.1.2.6.17 TDriveComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

5.1.2.6.18 TDriveComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

5.1.2.6.19 TDriveComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

5.1.2.6.20 TDriveComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

5.1.2.6.21 TDriveComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

5.1.2.6.22 TDriveComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

5.1.2.6.23 TDriveComboBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

5.1.2.6.24 TDriveComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own

Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

5.1.2.6.25 TDriveComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

5.1.2.6.26 TDriveComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
```

```
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

5.1.2.6.27 TDriveComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

5.1.2.6.28 TDriveComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

5.1.2.6.29 TDriveComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

5.1.2.6.30 TDriveComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                       //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
```

```
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines

5.1.2.6.31 TDriveComboBox.TextCase Property

Determines if the volume name in the Text (see page 1704) property appears in uppercase or lowercase.

Pascal

```
property TextCase: TTextCase;
```

C++

```
__property TTextCase TextCase;
```

Description

Set TextCase to the case that should be used to represent the drive letter that is the value of the Text (see page 1704) property.

Note: Changes made to the TextCase property at design time don't appear until the application runs.

See Also

Drive (see page 1690)

Text

5.1.2.6.32 TDriveComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

5.1.2.7 TDriveComboBox Events

The events of the TDriveComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 1708)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 1710)	Occurs when the user clicks the control.
	OnContextPopup (see page 1712)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1712)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1713)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1716)	Occurs when the user drags an object over a control.
	OnDropDown (see page 1719)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 1720)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1721)	Occurs when a control receives the input focus.
	OnExit (see page 1724)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1727)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1730)	Occurs when a key is pressed.
	OnKeyUp (see page 1731)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 1733)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1733)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 1734)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

5.1.2.7.1 TDriveComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
```

```
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
    end;
  ComboBox1.ItemIndex := 0;
  PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
```

```
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text

OnCloseUp

5.1.2.7.2 TDriveComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

5.1.2.7.3 TDriveComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

5.1.2.7.4 TDriveComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

5.1.2.7.5 TDriveComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

5.1.2.7.6 TDriveComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

5.1.2.7.7 TDriveComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown;
```

C++

```
__property OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown

OnCloseUp

5.1.2.7.8 TDriveComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

5.1.2.7.9 TDriveComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
```

```
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

5.1.2.7.10 TDriveComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

5.1.2.7.11 TDriveComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

5.1.2.7.12 TDriveComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

5.1.2.7.13 TDriveComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown

OnKeyPress

5.1.2.7.14 TDriveComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

5.1.2.7.15 TDriveComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

5.1.2.7.16 TDriveComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

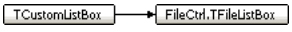
OnDragOver

OnDragDrop

5.1.3 TFileListBox Class

TFileListBox is a specialized list box that lists all the files in a specified directory.

Class Hierarchy



Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TFileListBox = class(TCustomListBox);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TFileListBox : public TCustomListBox;
```

File

FileCtrl

Description

Add TFileListBox to a form to allow users to select a file. Use TFileListBox along with TDriveComboBox (see page 1668), TFilterComboBox (see page 1813), and TDirectoryListBox (see page 1589) to add full file selection capabilities to a form.

Note: To add a standard Windows file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

See Also

TDirectoryListBox (see page 1589)

TDriveComboBox (see page 1668)

TFilterComboBox (see page 1813)

TOpenDialog

TSaveDialog

5.1.3.1 TFileListBox Members

The following tables list the members exposed by TFileListBox.

Protected Fields

	Name	Description
	DirBMP (see page 1739)	This is DirBMP, a member of class TFileListBox.
	ExeBMP (see page 1739)	This is ExeBMP, a member of class TFileListBox.
	FDirectory (see page 1739)	This is FDirectory, a member of class TFileListBox.
	FDirList (see page 1740)	This is FDirList, a member of class TFileListBox.
	FFileEdit (see page 1740)	This is FFileEdit, a member of class TFileListBox.
	FFileType (see page 1740)	This is FFileType, a member of class TFileListBox.
	FFilterCombo (see page 1740)	This is FFilterCombo, a member of class TFileListBox.
	FLastSel (see page 1741)	This is FLastSel, a member of class TFileListBox.
	FMask (see page 1741)	This is FMask, a member of class TFileListBox.
	FOnChange (see page 1741)	This is FOnChange, a member of class TFileListBox.
	FShowGlyphs (see page 1741)	This is FShowGlyphs, a member of class TFileListBox.
	UnknownBMP (see page 1741)	This is UnknownBMP, a member of class TFileListBox.

Protected Methods

	Name	Description
	Change (see page 1743)	This is Change, a member of class TFileListBox.
	Click (see page 1744)	Respond to user click.
	CreateWnd (see page 1744)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 1745)	Generates an OnDrawItem event.
	GetFilePath (see page 1745)	This is GetFilePath, a member of class TFileListBox.
	Notification (see page 1745)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 1746)	This is ReadBitmaps, a member of class TFileListBox.
	ReadFileNames (see page 1746)	This is ReadFileNames, a member of class TFileListBox.

Public Methods

	Name	Description
	ApplyFilePath (see page 1742)	Updates the file list box to represent a fully qualified file name.
	Create (see page 1738)	Creates an instance of TFileListBox (see page 1734).
	Destroy (see page 1738)	Destroys an instance of TFileListBox (see page 1734).
	Update (see page 1746)	Refreshes the directory list for the file list box.

Public Properties

	Name	Description
	Directory (see page 1756)	Specifies the directory that contains all the files listed in the file list box.
	Drive (see page 1761)	Specifies the drive which contains the files listed in the list box.
	FileName (see page 1764)	Indicates the name of the selected file in the list box, including the path name.

Published Events

	Name	Description
	OnChange (see page 1782)	Occurs immediately after the user changes the selected file in the list.
	OnClick (see page 1782)	Occurs when the user clicks the control.
	OnContextPopup (see page 1784)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 1785)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1785)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1788)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 1791)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1792)	Occurs when a control receives the input focus.
	OnExit (see page 1795)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1797)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1801)	Occurs when a key is pressed.
	OnKeyUp (see page 1802)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1804)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 1805)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1807)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1807)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1808)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1810)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 1812)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1748)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1751)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1752)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 1752)	Specifies which edges of the control are beveled.
	BevelInner (see page 1753)	Specifies the cut of the inner bevel.
	BevelKind (see page 1753)	Specifies the control's bevel style.
	BevelOuter (see page 1754)	Specifies the cut of the outer bevel.
	Color (see page 1754)	Specifies the background color of the control.
	Constraints (see page 1755)	Specifies the size constraints for the control.
	Ctl3D (see page 1755)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1757)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1759)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1759)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1762)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 1762)	Determines if the user can select a sequential range of items in the list box.
	FileEdit (see page 1763)	Links the directory list box to an edit control that displays the currently selected file.
	FileType (see page 1764)	Determines which files are displayed in the file list box based on the attributes of the files.
	Font (see page 1765)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1766)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1767)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 1767)	Determines whether the list box displays the partial items.
	ItemHeight (see page 1768)	Specifies the height in pixels of an item in an owner-draw list box.
	Mask (see page 1768)	Limits which files are displayed in the file list box.
	MultiSelect (see page 1769)	Specifies whether the user can select more than one item.
	ParentColor (see page 1770)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1770)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 1771)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1771)	Determines where a control looks for its font information.
	ParentShowHint (see page 1771)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1772)	Identifies the pop-up menu associated with the control.
	ShowGlyphs (see page 1777)	Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.
	ShowHint (see page 1777)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1779)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1779)	Determines if the user can tab to a control.
	Visible (see page 1780)	Determines whether the component appears onscreen.

5.1.3.2 TFileListBox.Create Constructor

Creates an instance of TFileListBox (see page 1734).

Pascal

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

C++

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

Description

Call Create to create a TFileListBox (see page 1734) object at runtime. File list boxes placed on a form at design time are created automatically.

See Also

Directory (see page 1756)

Drive (see page 1761)

FileType (see page 1764)

Mask (see page 1768)

MultiSelect

Sorted

Create

Width

5.1.3.3 TFileListBox.Destroy Destructor

Destroys an instance of TFileListBox (see page 1734).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFileListBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the file list box reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

5.1.3.4 TFileListBox Fields

The fields of the TFileListBox class are listed here.

Protected Fields

	Name	Description
	DirBMP (see page 1739)	This is DirBMP, a member of class TFileListBox.
	ExeBMP (see page 1739)	This is ExeBMP, a member of class TFileListBox.
	FDirectory (see page 1739)	This is FDirectory, a member of class TFileListBox.
	FDirList (see page 1740)	This is FDirList, a member of class TFileListBox.
	FFileEdit (see page 1740)	This is FFileEdit, a member of class TFileListBox.
	FFileType (see page 1740)	This is FFileType, a member of class TFileListBox.
	FFilterCombo (see page 1740)	This is FFilterCombo, a member of class TFileListBox.
	FLastSel (see page 1741)	This is FLastSel, a member of class TFileListBox.
	FMask (see page 1741)	This is FMask, a member of class TFileListBox.
	FOnChange (see page 1741)	This is FOnChange, a member of class TFileListBox.
	FShowGlyphs (see page 1741)	This is FShowGlyphs, a member of class TFileListBox.
	UnknownBMP (see page 1741)	This is UnknownBMP, a member of class TFileListBox.

5.1.3.4.1 TFileListBox.DirBMP Field

Pascal

```
DirBMP: TBitmap;
```

C++

```
TBitmap DirBMP;
```

Description

This is DirBMP, a member of class TFileListBox.

5.1.3.4.2 TFileListBox.ExeBMP Field

Pascal

```
ExeBMP: TBitmap;
```

C++

```
TBitmap ExeBMP;
```

Description

This is ExeBMP, a member of class TFileListBox.

5.1.3.4.3 TFileListBox.FDirectory Field

Pascal

```
FDirectory: string;
```

C++

```
UnicodeString FDirectory;
```

Description

This is FDirectory, a member of class TFileListBox.

5.1.3.4.4 TFileListBox.FDirList Field

Pascal

```
FDirList: TDirectoryListBox;
```

C++

```
TDirectoryListBox FDirList;
```

Description

This is FDirList, a member of class TFileListBox.

5.1.3.4.5 TFileListBox.FFileEdit Field

Pascal

```
FFileEdit: TEdit;
```

C++

```
TEdit FFileEdit;
```

Description

This is FFileEdit, a member of class TFileListBox.

5.1.3.4.6 TFileListBox.FFileType Field

Pascal

```
FFileType: TFileType;
```

C++

```
TFileType FFileType;
```

Description

This is FFileType, a member of class TFileListBox.

5.1.3.4.7 TFileListBox.FFilterCombo Field

Pascal

```
FFilterCombo: TFilterComboBox;
```

C++

```
TFilterComboBox FFilterCombo;
```

Description

This is FFilterCombo, a member of class TFileListBox.

5.1.3.4.8 TFileListBox.FLastSel Field

Pascal

```
FLastSel: Integer;
```

C++

```
int FLastSel;
```

Description

This is FLastSel, a member of class TFileListBox.

5.1.3.4.9 TFileListBox.FMask Field

Pascal

```
FMask: string;
```

C++

```
UnicodeString FMask;
```

Description

This is FMask, a member of class TFileListBox.

5.1.3.4.10 TFileListBox.FOnChange Field

Pascal

```
FOnChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnChange;
```

Description

This is FOnChange, a member of class TFileListBox.

5.1.3.4.11 TFileListBox.FShowGlyphs Field

Pascal

```
FShowGlyphs: Boolean;
```

C++

```
Boolean FShowGlyphs;
```

Description

This is FShowGlyphs, a member of class TFileListBox.

5.1.3.4.12 TFileListBox.UnknownBMP Field

Pascal

```
UnknownBMP: TBitmap;
```

C++

```
TBitmap UnknownBMP;
```

Description

This is UnknownBMP, a member of class TFileListBox.

5.1.3.5 TFileListBox Methods

The methods of the TFileListBox class are listed here.

Protected Methods

	Name	Description
	Change (see page 1743)	This is Change, a member of class TFileListBox.
	Click (see page 1744)	Respond to user click.
	CreateWnd (see page 1744)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 1745)	Generates an OnDrawItem event.
	GetFilePath (see page 1745)	This is GetFilePath, a member of class TFileListBox.
	Notification (see page 1745)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 1746)	This is ReadBitmaps, a member of class TFileListBox.
	ReadFileNames (see page 1746)	This is ReadFileNames, a member of class TFileListBox.

Public Methods

	Name	Description
	ApplyFilePath (see page 1742)	Updates the file list box to represent a fully qualified file name.
	Update (see page 1746)	Refreshes the directory list for the file list box.

5.1.3.5.1 TFileListBox.ApplyFilePath Method

Updates the file list box to represent a fully qualified file name.

Pascal

```
procedure ApplyFilePath(const EditText: string); virtual;
```

C++

```
virtual __fastcall ApplyFilePath(const UnicodeString EditText);
```

Description

Call ApplyFilePath to change the Directory (see page 1756), Drive (see page 1761), FileName (see page 1764), and Mask (see page 1768) properties so that they match the values implied by the EditText parameter. For example, if an edit control is linked to the file list box using the FileEdit (see page 1763) property, call ApplyFilePath from the OnChange (see page 1782) event handler of the edit control to update the file list box whenever the user types a value in the edit control.

ApplyFilePath updates the Mask (see page 1768) property only if the EditText parameter includes wildcards. If the form contains a TDirectoryListBox (see page 1589) object that specifies this file list box object in its FileList property, the Directory (see page 1756) and Drive (see page 1761) properties of the directory list box object are updated as well. If the form also includes a TDriveComboBox (see page 1668) object that specifies the directory list box in its DirList property, the Drive (see page 1761) property of the drive combo box will also be updated. However, the Mask (see page 1768) property of any TFilterComboBox (see page 1813) will not be updated.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Directory (see page 1756)

Drive (see page 1761)

FileEdit (see page 1763)

FileName (see page 1764)

Mask (see page 1768)

TDirectoryListBox (see page 1589)

TDriveComboBox (see page 1668)

TFilterComboBox (see page 1813)

5.1.3.5.2 TFileListBox.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TFileListBox.

5.1.3.5.3 TFileListBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

5.1.3.5.4 TFileListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns

CreateHandle

CreateParams

CreateWindowHandle

CreateWnd

ItemIndex

Items

TabWidth

TopIndex

5.1.3.5.5 TFileListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight

MeasureItem

OnDrawItem

OnMeasureItem

Style

5.1.3.5.6 TFileListBox.GetFilePath Method

Pascal

```
function GetFilePath: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetFilePath();
```

Description

This is GetFilePath, a member of class TFileListBox.

5.1.3.5.7 TFileListBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.3.5.8 TFileListBox.ReadBitmaps Method

Pascal

```
procedure ReadBitmaps; virtual;
```

C++

```
virtual __fastcall ReadBitmaps();
```

Description

This is ReadBitmaps, a member of class TFileListBox.

5.1.3.5.9 TFileListBox.ReadFileNames Method

Pascal

```
procedure ReadFileNames; virtual;
```

C++

```
virtual __fastcall ReadFileNames();
```

Description

This is ReadFileNames, a member of class TFileListBox.

5.1.3.5.10 TFileListBox.Update Method

Refreshes the directory list for the file list box.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update whenever the files listed by the file list box may be out of date. For example, if an application creates, renames, or deletes files, calling Update will ensure that those changes are reflected in the file list box.

The list box is updated automatically when the Drive (see page 1761), Directory (see page 1756), FileType (see page 1764), or Mask (see page 1768) property is changed.

See Also

Directory (see page 1756)

Drive (see page 1761)

FileType (see page 1764)

Mask (see page 1768)

5.1.3.6 TFileListBox Properties

The properties of the TFileListBox class are listed here.

Public Properties

	Name	Description
	Directory (see page 1756)	Specifies the directory that contains all the files listed in the file list box.
	Drive (see page 1761)	Specifies the drive which contains the files listed in the list box.
	FileName (see page 1764)	Indicates the name of the selected file in the list box, including the path name.

Published Properties

	Name	Description
	Align (see page 1748)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1751)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1752)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 1752)	Specifies which edges of the control are beveled.
	BevelInner (see page 1753)	Specifies the cut of the inner bevel.
	BevelKind (see page 1753)	Specifies the control's bevel style.
	BevelOuter (see page 1754)	Specifies the cut of the outer bevel.
	Color (see page 1754)	Specifies the background color of the control.
	Constraints (see page 1755)	Specifies the size constraints for the control.
	Ctl3D (see page 1755)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1757)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1759)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1759)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1762)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 1762)	Determines if the user can select a sequential range of items in the list box.
	FileEdit (see page 1763)	Links the directory list box to an edit control that displays the currently selected file.
	FileType (see page 1764)	Determines which files are displayed in the file list box based on the attributes of the files.
	Font (see page 1765)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1766)	Determines the behavior of the input method editor (IME).

	ImeName (see page 1767)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 1767)	Determines whether the list box displays the partial items.
	ItemHeight (see page 1768)	Specifies the height in pixels of an item in an owner-draw list box.
	Mask (see page 1768)	Limits which files are displayed in the file list box.
	MultiSelect (see page 1769)	Specifies whether the user can select more than one item.
	ParentColor (see page 1770)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1770)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1771)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1771)	Determines where a control looks for its font information.
	ParentShowHint (see page 1771)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1772)	Identifies the pop-up menu associated with the control.
	ShowGlyphs (see page 1777)	Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.
	ShowHint (see page 1777)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1779)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1779)	Determines if the user can tab to a control.
	Visible (see page 1780)	Determines whether the component appears onscreen.

5.1.3.6.1 TFileListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
```

```
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
```

```
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

5.1.3.6.2 TFileListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

5.1.3.6.3 TFileListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

5.1.3.6.4 TFileListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

5.1.3.6.5 TFileListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

5.1.3.6.6 TFileListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

5.1.3.6.7 TFileListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

5.1.3.6.8 TFileListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
```

```
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

5.1.3.6.9 TFileListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

5.1.3.6.10 TFileListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

5.1.3.6.11 TFileListBox.Directory Property

Specifies the directory that contains all the files listed in the file list box.

Pascal

```
property Directory: string;
```

C++

```
__property UnicodeString Directory;
```

Description

Set Directory to update the file list box so that it displays the files in a new directory. If the form includes a TDirectoryListBox (see page 1589) object, set the FileList property of the directory list box to have the Directory property updated automatically when the user selects a new directory.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Drive (see page 1761)

Directory (see page 1611)

FileList (see page 1618)

5.1.3.6.12 TFileListBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
```

```
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

5.1.3.6.13 TFileListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

5.1.3.6.14 TFileListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

5.1.3.6.15 TFileListBox.Drive Property

Specifies the drive which contains the files listed in the list box.

Pascal

```
property Drive: char;
```

C++

```
__property char Drive;
```

Description

Set the value of Drive to the drive that contains the files that the list box should display. When the value of Drive changes, the Directory (see page 1756) value also changes to the current directory on the specified drive.

If the form includes a TDirectoryListBox (see page 1589) object, set the FileList property of the directory list box to have the Drive property updated automatically when the Drive property of the directory list box changes.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
```

```
  and the OnChange event handler for the directory list box.
  When the user changes the drive using the combo box, the
  directory list box and file list box will update to reflect
  the new drive and the current directory on that drive.  When
  the user double clicks on a directory in the file list box
  will update to reflect the new directory.
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;

  procedure TForm1.DriveComboBox1Change(Sender: TObject);
  begin
    DirectoryListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Directory (see page 1756)

Drive (see page 1617)

FileList (see page 1618)

5.1.3.6.16 TFileListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

5.1.3.6.17 TFileListBox.ExtendedSelect Property

Determines if the user can select a sequential range of items in the list box.

Pascal

```
property ExtendedSelect;
```

C++

```
__property ExtendedSelect;
```

Description

ExtendedSelect works with the MultiSelect property. If ExtendedSelect is true and MultiSelect is true, the user can select an item then hold down the Shift key and select another and all the items in between the two selected items also become selected.

If the user doesn't hold down the Shift or Ctrl key while selecting a second item, the first selected item becomes unselected— in other words, the user must use the Ctrl key to select multiple noncontiguous items, or the Shift key to select a range of items.

If ExtendedSelect is false and MultiSelect is true, the user can select multiple items without using the Shift or Ctrl key, but they can't select a range of items in one operation.

If MultiSelect is false, the setting of ExtendedSelect has no effect as the user will not be able to select more than one item at a time in the list box.

See Also

MultiSelect

5.1.3.6.18 TFileListBox.FileEdit Property

Links the directory list box to an edit control that displays the currently selected file.

Pascal

```
property FileEdit: TEdit;
```

C++

```
__property TEdit FileEdit;
```

Description

Set FileEdit to an edit control that displays the selected file from the file list box, as is commonly done in Open and Save dialog boxes. If no file is selected in the file list box, the text of the edit control is the current value of the file list box's Mask (see page 1768) property.

Note: If the user changes the text of the edit control, the results are not automatically reflected in the file list box. Use the ApplyFilePath (see page 1742) method to update the file list box when the edit control's text changes.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

ApplyFilePath (see page 1742)

FileName (see page 1764)

Mask (see page 1768)

5.1.3.6.19 TFileListBox.FileName Property

Indicates the name of the selected file in the list box, including the path name.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Set FileName to initialize the file list box to a particular file. Read FileName to obtain the name of the file the user selected.

If the FileEdit (see page 1763) property is set, the linked edit control displays the value of FileName whenever the FileName property changes.

See Also

FileEdit (see page 1763)

FileType (see page 1764)

5.1.3.6.20 TFileListBox.FileType Property

Determines which files are displayed in the file list box based on the attributes of the files.

Pascal

```
property FileType: TFileType;
```

C++

```
__property TFileType FileType;
```

Description

Set FileType to the type of files the list box should display. FileType can specify more than one type of file. For example, if the value of FileType is a set containing the values ftReadOnly and ftHidden, files that have either the read-only or the hidden attributes are displayed in the list box.

To set the FileType property using the Object Inspector, click the FileType property to see the attribute values. Then set each value to True or False true or false to specify whether it is included in the FileType set.

Note: The files that are listed can also be limited by the form of their names. To restrict the files to particular names, use the Mask (see page 1768) property or include a TFilterComboBox (see page 1813) on the form.

See Also

Mask (see page 1768)

TFilterComboBox (see page 1813)

5.1.3.6.21 TFileListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
```

```
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

5.1.3.6.22 TFileListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

5.1.3.6.23 TFileListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

5.1.3.6.24 TFileListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight;
```

C++

```
__property IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight

Items

Style

Using TDBListBox and TDBComboBox

5.1.3.6.25 TFileListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight;
```

C++

```
__property ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem

IntegralHeight

Items

MeasureItem

OnDrawItem

OnMeasureItem

Style

Using TDBListBox and TDBComboBox

5.1.3.6.26 TFileListBox.Mask Property

Limits which files are displayed in the file list box.

Pascal

```
property Mask: string;
```

C++

```
__property UnicodeString Mask;
```

Description

Set Mask to a regular expression to limit the list box to files that match the mask. The value of the mask is a file name that may include wildcards. The asterisk (*) is a wildcard which matches any number of arbitrary characters. The question mark (?) is a wildcard which matches a single arbitrary character. The file mask *.* displays all files, which is the default value.

To specify multiple file masks, separate the file mask specifications with semicolons. For example, *.DLL;*.EXE.

If the form includes a TFilterComboBox (see page 1813) object, set the FileList property of the filter combo box to have the Mask property updated automatically when the Mask property of the filter combo box changes.

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

FileEdit (see page 1763)

FileName (see page 1764)

FileList (see page 1832)

Mask (see page 1837)

5.1.3.6.27 TFileListBox.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect;
```

C++

```
__property MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount property indicates the number of selected items. When MultiSelect is false, the user can

only select one item, which is indicated by the ItemIndex property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl descendant, override the protected SetMultiSelect method.

See Also

SelCount

ItemIndex

5.1.3.6.28 TFileListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

5.1.3.6.29 TFileListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

5.1.3.6.30 TFileListBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

5.1.3.6.31 TFileListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

5.1.3.6.32 TFileListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

5.1.3.6.33 TFileListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

5.1.3.6.34 TFileListBox.ShowGlyphs Property

Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.

Pascal

```
property ShowGlyphs: Boolean;
```

C++

```
__property Boolean ShowGlyphs;
```

Description

Set ShowGlyphs to true to display icons next to the files in the list that indicate the file type. Set ShowGlyphs to false to list only the file names.

5.1.3.6.35 TFileListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

5.1.3.6.36 TFileListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

5.1.3.6.37 TFileListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

5.1.3.6.38 TFileListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

5.1.3.7 TFileListBox Events

The events of the TFileListBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 1782)	Occurs immediately after the user changes the selected file in the list.
	OnClick (see page 1782)	Occurs when the user clicks the control.
	OnContextPopup (see page 1784)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1785)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1785)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1788)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 1791)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1792)	Occurs when a control receives the input focus.
	OnExit (see page 1795)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1797)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1801)	Occurs when a key is pressed.
	OnKeyUp (see page 1802)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 1804)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 1805)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 1807)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1807)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 1808)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1810)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 1812)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

5.1.3.7.1 TFileListBox.OnChange Event

Occurs immediately after the user changes the selected file in the list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the FileName (see page 1764) property changes. If the FileEdit (see page 1763) property has been set, the edit control is updated before the OnChange event.

See Also

FileEdit (see page 1763)

FileName (see page 1764)

5.1.3.7.2 TFileListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
```

```
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

5.1.3.7.3 TFileListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

5.1.3.7.4 TFileListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

5.1.3.7.5 TFileListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
```

```
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
```

```
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

5.1.3.7.6 TFileListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
```

```
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
```

```
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

5.1.3.7.7 TFileListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
```

```
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

5.1.3.7.8 TFileListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
```

```
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

5.1.3.7.9 TFileListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
```

```
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

5.1.3.7.10 TFileListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
```

```
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
```

```
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

5.1.3.7.11 TFileListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

5.1.3.7.12 TFileListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
```

```
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

5.1.3.7.13 TFileListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events.

This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

5.1.3.7.14 TFileListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
```

```
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

5.1.3.7.15 TFileListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

5.1.3.7.16 TFileListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

5.1.3.7.17 TFileListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

5.1.3.7.18 TFileListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

5.1.3.7.19 TFileListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

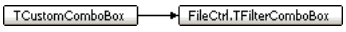
OnDragOver

OnDragDrop

5.1.4 TFilterComboBox Class

TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

Class Hierarchy



Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TFilterComboBox = class(TCustomComboBox);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TFilterComboBox : public TCustomComboBox;
```

File

FileCtrl

Description

Add a filter combo box to a form that includes a file selection control to provide a set of predefined file filters. Use TFilterComboBox along with TDriveComboBox (see page 1668), TDirectoryListBox (see page 1589), and TFileListBox (see page 1734) to add full file selection capabilities to a form.

To add a standard Win32 file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

See Also

TDirectoryListBox (see page 1589)

TDriveComboBox (see page 1668)

TFileListBox (see page 1734)

5.1.4.1 TFilterComboBox Members

The following tables list the members exposed by TFilterComboBox.

Protected Methods

	Name	Description
	BuildList (see page 1816)	This is BuildList, a member of class TFilterComboBox.
V	Change (see page 1816)	Generates an OnChange event.
V	Click (see page 1816)	Respond to user click.
V	CreateWnd (see page 1817)	Creates the Windows control that represents the combo box.

	Notification (see page 1817)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
	Create (see page 1815)	Creates and initializes an instance of TFilterComboBox (see page 1813).
	Destroy (see page 1815)	Destroys an instance of TFilterComboBox (see page 1813).

Public Properties

	Name	Description
	Mask (see page 1837)	Specifies the mask associated with the string selected in the filter combo box.
	Text (see page 1848)	Contains a text string associated with the control.

Published Events

	Name	Description
	OnChange (see page 1851)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 1853)	Occurs when the user clicks the control.
	OnContextPopup (see page 1855)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1856)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1856)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1859)	Occurs when the user drags an object over a control.
	OnDropDown (see page 1862)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 1864)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1865)	Occurs when a control receives the input focus.
	OnExit (see page 1867)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1870)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1874)	Occurs when a key is pressed.
	OnKeyUp (see page 1875)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 1876)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1877)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 1877)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 1819)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1822)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1823)	Positions to matching list items as you type.
	AutoDropDown (see page 1824)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 1825)	Specifies which edges of the control are beveled.
	BevelInner (see page 1825)	Specifies the cut of the inner bevel.
	BevelKind (see page 1825)	Specifies the control's bevel style.
	BevelOuter (see page 1826)	Specifies the cut of the outer bevel.
	Color (see page 1826)	Specifies the background color of the control.
	Constraints (see page 1827)	Specifies the size constraints for the control.
	Ctl3D (see page 1828)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1828)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1830)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1830)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1832)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 1832)	Links the filter combo box to a file list box control.

	Filter (see page 1833)	Contains all the file masks displayed in the filter combo box.
	Font (see page 1835)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1836)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1836)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 1838)	Determines where a control looks for its color information.
	ParentCtl3D (see page 1839)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1839)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1839)	Determines where a control looks for its font information.
	ParentShowHint (see page 1840)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1840)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1845)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1847)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1847)	Determines if the user can tab to a control.
	Visible (see page 1849)	Determines whether the component appears onscreen.

5.1.4.2 TFilterComboBox.Create Constructor

Creates and initializes an instance of TFilterComboBox (see page 1813).

Pascal

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

C++

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

Description

Call Create to create a TFilterComboBox (see page 1813) objectat runtime. Filter (see page 1833) combo boxes placed on a form at design time are created automatically.

See Also

Mask (see page 1837)

Style

Create

5.1.4.3 TFilterComboBox.Destroy Destructor

Destroys an instance of TFilterComboBox (see page 1813).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFilterComboBox();
```

Description

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

See Also

System::TObject::Free

5.1.4.4 TFilterComboBox Methods

The methods of the TFilterComboBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 1816)	This is BuildList, a member of class TFilterComboBox.
V	Change (see page 1816)	Generates an OnChange event.
V	Click (see page 1816)	Respond to user click.
V	CreateWnd (see page 1817)	Creates the Windows control that represents the combo box.
V	Notification (see page 1817)	Responds to notifications that components are being created or destroyed.

5.1.4.4.1 TFilterComboBox.BuildList Method

Pascal

```
procedure BuildList;
```

C++

```
__fastcall BuildList();
```

Description

This is BuildList, a member of class TFilterComboBox.

5.1.4.4.2 TFilterComboBox.Change Method

Generates an OnChange event.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange event. Override this method to add additional processing when the text changes, or to block the OnChange event.

See Also

OnChange

Changed

5.1.4.4.3 TFilterComboBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

5.1.4.4.4 TFilterComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle

5.1.4.4.5 TFilterComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.4.5 TFilterComboBox Properties

The properties of the TFilterComboBox class are listed here.

Public Properties

	Name	Description
	Mask (see page 1837)	Specifies the mask associated with the string selected in the filter combo box.
	Text (see page 1848)	Contains a text string associated with the control.

Published Properties

	Name	Description
	Align (see page 1819)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1822)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 1823)	Positions to matching list items as you type.
	AutoDropDown (see page 1824)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 1825)	Specifies which edges of the control are beveled.
	BevelInner (see page 1825)	Specifies the cut of the inner bevel.
	BevelKind (see page 1825)	Specifies the control's bevel style.
	BevelOuter (see page 1826)	Specifies the cut of the outer bevel.
	Color (see page 1826)	Specifies the background color of the control.
	Constraints (see page 1827)	Specifies the size constraints for the control.
	Ctl3D (see page 1828)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 1828)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1830)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 1830)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 1832)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 1832)	Links the filter combo box to a file list box control.
	Filter (see page 1833)	Contains all the file masks displayed in the filter combo box.
	Font (see page 1835)	Controls the attributes of text written on or in the control.
	ImeMode (see page 1836)	Determines the behavior of the input method editor (IME).
	ImeName (see page 1836)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 1838)	Determines where a control looks for its color information.

	ParentCtl3D (see page 1839)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 1839)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1839)	Determines where a control looks for its font information.
	ParentShowHint (see page 1840)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1840)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 1845)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 1847)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1847)	Determines if the user can tab to a control.
	Visible (see page 1849)	Determines whether the component appears onscreen.

5.1.4.5.1 TFilterComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
```

```
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
```

```
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

5.1.4.5.2 TFilterComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at

design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

5.1.4.5.3 TFilterComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete;
```

C++

```
__property AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

AutoCompleteDelay

5.1.4.5.4 TFilterComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown;
```

C++

```
__property AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete

OnDropDown

DroppedDown

5.1.4.5.5 TFilterComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

5.1.4.5.6 TFilterComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

5.1.4.5.7 TFilterComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

5.1.4.5.8 TFilterComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

5.1.4.5.9 TFilterComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically

set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

5.1.4.5.10 TFilterComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

5.1.4.5.11 TFilterComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

5.1.4.5.12 TFilterComboBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

5.1.4.5.13 TFilterComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

5.1.4.5.14 TFilterComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
```

```
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

5.1.4.5.15 TFilterComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

5.1.4.5.16 TFilterComboBox.FileList Property

Links the filter combo box to a file list box control.

Pascal

```
property FileList: TFileListBox;
```

C++

```
__property TFileListBox FileList;
```

Description

If the form includes a TFileListBox (see page 1734) object to display a list of files to choose from, set FileList to allow the filter combo box to limit which files are displayed in the list box. Once the two controls are connected, whenever a new filter is selected using a filter combo box, the file list box displays the files that match the selected filter.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Mask (see page 1837)

5.1.4.5.17 TFilterComboBox.Filter Property

Contains all the file masks displayed in the filter combo box.

Pascal

```
property Filter: string;
```

C++

```
__property UnicodeString Filter;
```

Description

Set the Filter property to specify the names that appear in the filter combo box, along with the masks associated with each name. Each mask is a file name that may include wildcards. The asterisk (*) is a wildcard which matches any number of arbitrary characters. The question mark (?) is a wildcard which matches a single arbitrary character. When the application runs, the file filter the user selects in the filter combo box becomes the value of the Mask (see page 1837) property.

To create the value of Filter for a single file type, follow these steps:

1Type some meaningful text to indicate the type of file.

2Type a | character (vertical bar).

3Type the file filter.

Don't put in any spaces around the | character in the string.

Here's an example:

```
FilterComboBox1.Filter := 'Text files|*.TXT';
FilterComboBox1->Filter = "Text files|*.TXT";
```

If Filter is set to this string, the string "Text (see page 1848) files" appears in the filter combo box. When the user selects "Text (see page 1848) files", the mask is set to "*.TXT".

To specify multiple file filters, extend the value of Filter, separating each file name/mask combination with a | character. Filter can specify an any number of file filters, as long as the string is less than 255 characters.

Here's an example of two file filters that could be specified as the value of the Filter property:

```
'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)|*.PAS'
"Text files (*.TXT)|*.TXT|C++ files (*.CPP)|*.CPP"
```

The previous example includes the file filters in parentheses in the text parts. This isn't required, but it's a common convention that helps users understand what to expect when they select a file filter.

The mask portion can include multiple file specifiers if they are separated with semicolons:

```
'All files|*.TXT;*.PAS;*.WB1'
"All files|*.TXT;*.CPP;*.RC"
```

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

Mask (see page 1837)

5.1.4.5.18 TFilterComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

5.1.4.5.19 TFilterComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

5.1.4.5.20 TFilterComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

5.1.4.5.21 TFilterComboBox.Mask Property

Specifies the mask associated with the string selected in the filter combo box.

Pascal

```
property Mask: string;
```

C++

```
__property UnicodeString Mask;
```

Description

Read Mask to obtain the mask that was selected by the user.

When the user uses the filter combo box to select a new filter, the mask associated with the selected file type becomes the value of the Mask property. If a file list box is linked to the filter combo box with the FileList (see page 1832) property, the file list will update when the value of Mask changes.

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
```

```
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

FileList (see page 1832)

Filter (see page 1833)

Mask (see page 1768)

5.1.4.5.22 TFilterComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

5.1.4.5.23 TFilterComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

5.1.4.5.24 TFilterComboBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

5.1.4.5.25 TFilterComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

5.1.4.5.26 TFilterComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

5.1.4.5.27 TFilterComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

5.1.4.5.28 TFilterComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

5.1.4.5.29 TFilterComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

5.1.4.5.30 TFilterComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

5.1.4.5.31 TFilterComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
```

```
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines

5.1.4.5.32 TFilterComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

5.1.4.6 TFilterComboBox Events

The events of the TFilterComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 1851)	Occurs when the user changes the text displayed in the edit region.

	OnClick (see page 1853)	Occurs when the user clicks the control.
	OnContextPopup (see page 1855)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 1856)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 1856)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1859)	Occurs when the user drags an object over a control.
	OnDropDown (see page 1862)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 1864)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 1865)	Occurs when a control receives the input focus.
	OnExit (see page 1867)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 1870)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1874)	Occurs when a key is pressed.
	OnKeyUp (see page 1875)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 1876)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 1877)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 1877)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

5.1.4.6.1 TFilterComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
```

```
        PageControl1.Pages[i]);
      end;
    ComboBox1.ItemIndex := 0;
    PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
  end;

  procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    if (Sender is TComboBox) then
      with (Sender as TComboBox) do
        PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
  end;
  {
  The following example assumes that a drive combo box, a file
  list box, and a directory list box are on a form. Add this
  code as the OnChange event handler for the drive combo box
  and the OnChange event handler for the directory list box.
  When the user changes the drive using the combo box, the
  directory list box and file list box will update to reflect
  the new drive and the current directory on that drive.  When
  the user double clicks on a directory in the file list box
  will update to reflect the new directory.
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;

  procedure TForm1.DriveComboBox1Change(Sender: TObject);
  begin
    DirectoryListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Text

OnCloseUp

5.1.4.6.2 TFilterComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).

- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

5.1.4.6.3 TFilterComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

5.1.4.6.4 TFilterComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

5.1.4.6.5 TFilterComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

5.1.4.6.6 TFilterComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
```

```
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

5.1.4.6.7 TFilterComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown;
```

C++

```
__property OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
```

```
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
  Form3.Show;
end;
```

See Also

DroppedDown

OnCloseUp

5.1.4.6.8 TFilterComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

5.1.4.6.9 TFilterComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
```

```
    (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
```

```
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

5.1.4.6.10 TFilterComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
```

```
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

5.1.4.6.11 TFilterComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
```

```
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

5.1.4.6.12 TFilterComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

5.1.4.6.13 TFilterComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
```

```
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

5.1.4.6.14 TFilterComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

5.1.4.6.15 TFilterComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

5.1.4.6.16 TFilterComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

5.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DirectoryExists (see page 1879)	Determines whether a specified directory exists.
	ForceDirectories (see page 1879)	Creates a new directory, including the creation of parent directories as needed.
	MinimizeName (see page 1880)	Shortens a fully qualified path name so that it can be drawn with a specified length limit.
	ProcessPath (see page 1881)	Parses a file name into its constituent parts.
	SelectDirectory (see page 1881)	Brings up a dialog to allow the user to enter a directory name.
	SelectDirectory (see page 1881)	Brings up a dialog to allow the user to enter a directory name.

5.2.1 FileCtrl.DirectoryExists Function

Determines whether a specified directory exists.

Pascal

```
function DirectoryExists(const Name: string): Boolean;
```

C++

```
Boolean DirectoryExists(const UnicodeString Name);
```

File

FileCtrl

Description

Call DirectoryExists to determine whether the directory specified by the Name parameter exists. If the directory exists, the function returns true. If the directory does not exist, the function returns false.

If a full path name is entered, DirectoryExists searches for the directory along the designated path. Otherwise, the Name parameter is interpreted as a relative path name from the current directory.

Note: The FileCtrl (see page 1587) unit (Windows only) also contains a DirectoryExists function. However, the FileCtrl (see page 1587) version is deprecated, and the SysUtils version preferred, even if the code does not need to be cross-platform.

See Also

ForceDirectories

SelectDirectory

FileExists

5.2.2 FileCtrl.ForceDirectories Function

Creates a new directory, including the creation of parent directories as needed.

Pascal

```
function ForceDirectories(const Dir: string): Boolean;
```

C++

```
Boolean ForceDirectories(const UnicodeString Dir);
```

File

FileCtrl

Description

ForceDirectories creates a new directory as specified in Dir, which must be a fully-qualified path name. If the directories given in the path do not yet exist, ForceDirectories attempts to create them.

ForceDirectories returns true if it successfully creates all necessary directories, false if it could not create a needed directory.

Warning: Important:Do not call ForceDirectories with an empty string. Doing so causes ForceDirectories to raise an exception.

Note: The FileCtrl (see page 1587) unit (Windows only) also contains a ForceDirectories function. However, the FileCtrl (see page 1587) version is deprecated, and the SysUtils version preferred, even if the code does not need to be cross-platform.

See Also

ExtractFileDir

GetCurrentDir

RemoveDir

SetCurrentDir

CreateDir

DirectoryExists

SelectDirectory

5.2.3 FileCtrl.MinimizeName Function

Shortens a fully qualified path name so that it can be drawn with a specified length limit.

Pascal

```
function MinimizeName(const Filename: TFileName; Canvas: TCanvas; MaxLen: Integer): TFileName;
```

C++

```
TFileName MinimizeName(const TFileName Filename, TCanvas Canvas, int MaxLen);
```

File

FileCtrl

Description

Use MinimizeName to obtain a version of a file name and path that can be drawn on a limited size drawing surface. MinimizeName replaces directories in the path portion of the file name with dots until the resulting name fits the specified number of pixels in length.

MinimizeName returns the shortened name.

Filename is the fully qualified path name that (possibly) needs to be shortened.

Canvas is the drawing surface on which the resulting name is to be rendered. It is used to determine font metrics.

MaxLen is the length, in pixels, available for drawing the file name on the canvas.

See Also

AdjustLineBreaks

WrapText

5.2.4 FileCtrl.ProcessPath Function

Parses a file name into its constituent parts.

Pascal

```
procedure ProcessPath(const EditText: string; var Drive: Char; var DirPart: string; var
FilePart: string);
```

C++

```
ProcessPath(const UnicodeString EditText, Char Drive, UnicodeString DirPart, UnicodeString
FilePart);
```

File

FileCtrl

Description

Call ProcessPath to parse a file name into its drive, path, and file name.

Pass the full file name that should be parsed as the EditText parameter. EditText must refer to an existing file or directory name.

ProcessPath returns the indicated drive, directory, and file name as the Drive, DirPart, and FilePart parameters.

See Also

ExtractFileDir

ExtractFileDrive

ExtractFileExt

ExtractFileName

ExtractFilePath

5.2.5 FileCtrl.SelectDirectory Function

Brings up a dialog to allow the user to enter a directory name.

Pascal

```
function SelectDirectory(var Directory: string; Options: TSelectDirOpts; HelpCtx: Longint):
Boolean; overload;
function SelectDirectory(const Caption: string; const Root: WideString; var Directory: string;
Options: TSelectDirExtOpts = [sdNewUI]; Parent: TWinControl = nil): Boolean; overload;
```

C++

```
Boolean SelectDirectory(UnicodeString Directory, TSelectDirOpts Options, Longint HelpCtx);
Boolean SelectDirectory(const UnicodeString Caption, const BSTR Root, UnicodeString Directory,
TSelectDirExtOpts Options = [sdNewUI], TWinControl * Parent = nil);
```

File

FileCtrl

Description

Call SelectDirectory to let the user enter a directory name.

Use the first syntax to call the Select Directory dialog box. The directory passed to the function with the Directory parameter appears as the currently selected directory when the dialog box appears. The name of the directory the user selects becomes the value of Directory when the function returns.

The HelpCtx parameter is the help context ID number.

Use the second syntax to display the Windows directory browser. The Caption parameter specifies a caption for the dialog. The Root parameter specifies the root directory from which to browse. The selected directory is returned as the Directory parameter. When using this syntax, SelectDirectory does not change the value of the current directory.

Warning: You can't use the same variable for the Root parameter and the Directory parameter.

The Options parameter is a set of values. If Options is the empty set, the user can only select a directory that already exists. No edit box is provided for the user to enter a new directory name. If Options is not empty, the included values determine how the dialog responds when the user types a nonexistent directory name.

With either syntax, SelectDirectory returns true if the user selected a directory and chose OK, and false if the user chose Cancel or closed the dialog box without selecting a directory.

See Also

DirectoryExists (see page 1879)

ForceDirectories (see page 1879)

5.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDriveType (see page 1883)	This is record FileCtrl.TDriveType.
	TFileAttr (see page 1883)	TFileAttr and TFileType (see page 1887) determine which files a file list box displays.
	TSelectDirExtOpt (see page 1884)	This is record FileCtrl.TSelectDirExtOpt.
	TSelectDirOpt (see page 1885)	TSelectDirOpt and TSelectDirOpts (see page 1888) determine how a directory selection dialog responds when the user enters a nonexistent directory.
	TTextCase (see page 1886)	TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

5.3.1 FileCtrl.TDriveType Enumeration

Pascal

```
TDriveType = (
  dtUnknown,
  dtNoDrive,
  dtFloppy,
  dtFixed,
  dtNetwork,
  dtCDROM,
  dtRAM
);
```

C++

```
enum TDriveType {
  dtUnknown,
  dtNoDrive,
  dtFloppy,
  dtFixed,
  dtNetwork,
  dtCDROM,
  dtRAM
};
```

File

FileCtrl

Description

This is record FileCtrl.TDriveType.

5.3.2 FileCtrl.TFileAttr Enumeration

TFileAttr and TFileType (see page 1887) determine which files a file list box displays.

Pascal

```
TFileAttr = (
  ftReadOnly,
  ftHidden,
  ftSystem,
  ftVolumeID,
  ftDirectory,
  ftArchive,
  ftNormal
);
```

C++

```
enum TFileAttr {
  ftReadOnly,
  ftHidden,
  ftSystem,
  ftVolumeID,
  ftDirectory,
  ftArchive,
  ftNormal
};
```

File

FileCtrl

Description

TFileType (see page 1887) is a set of TFileAttr values. These are the possible TFileAttr values:

Value	Meaning
ftReadOnly	The list box displays files with the read-only attribute.
ftHidden	The list box displays files with the hidden attribute.
ftSystem	The list box displays system files.
ftVolumeID	The list box displays the volume name.
ftDirectory	The list box displays directories.
ftArchive	The list box displays files with archive attribute.
ftNormal	The list box displays files with no special attributes.

See Also

SelectDirectory

5.3.3 FileCtrl.TSelectDirExtOpt Enumeration

Pascal

```
TSelectDirExtOpt = (
  sdNewFolder,
  sdShowEdit,
  sdShowShares,
  sdNewUI,
  sdShowFiles,
  sdValidateDir
);
```

C++

```
enum TSelectDirExtOpt {
  sdNewFolder,
  sdShowEdit,
  sdShowShares,
  sdNewUI,
  sdShowFiles,
  sdValidateDir
};
```

File

FileCtrl

Description

This is record FileCtrl.TSelectDirExtOpt.

5.3.4 FileCtrl.TSelectDirOpt Enumeration

TSelectDirOpt and TSelectDirOpts (see page 1888) determine how a directory selection dialog responds when the user enters a nonexistent directory.

Pascal

```
TSelectDirOpt = (
  sdAllowCreate,
  sdPerformCreate,
  sdPrompt
);
```

C++

```
enum TSelectDirOpt {
  sdAllowCreate,
  sdPerformCreate,
  sdPrompt
};
```

File

FileCtrl

Description

These are the possible values that can be added to the set of options:

Value	Meaning
sdAllowCreate	An edit box allows the user to type in the name of a directory that does not exist. This option does not create a directory: the application must read the name of the selected directory and create it if desired.
sdPerformCreate	Used only in combination with sdAllowCreate. If the user enters a directory name that does not exist, the directory selection dialog creates it.
sdPrompt	Used only in combination with sdAllowCreate. Displays a message box that informs the user when the entered directory does not exist and asks if the directory should be created. If the user chooses OK, the directory is created if the option set includes sdPerformCreate. If the option set does not include sdPerformCreate, the directory is not created: the application must read the directory name and create it.

See Also

SelectDirectory

5.3.5 FileCtrl.TTextCase Enumeration

TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

Pascal

```
TTextCase = (
  tcLowerCase,
  tcUpperCase
);
```

C++

```
enum TTextCase {
  tcLowerCase,
  tcUpperCase
};
```

File

FileCtrl

Description

TTextCase can have one of the following values:

Value	Meaning
tcLowerCase	The volume name is displayed in lowercase letters.
tcUpperCase	The volume name is displayed in uppercase letters.

5.4 Types

The following table lists types in this documentation.

Types

Name	Description
TFileType (see page 1887)	TFileType describes possible combinations of file attributes.
TSelectDirExtOpts (see page 1887)	This is type FileCtrl.TSelectDirExtOpts.
TSelectDirOpts (see page 1888)	This is type FileCtrl.TSelectDirOpts.

5.4.1 FileCtrl.TFileType Type

TFileType describes possible combinations of file attributes.

Pascal

```
TFileType = set of TFileAttr;
```

C++

```
TFileAttr TFileType;
```

File

FileCtrl

Description

TFileType describes possible combinations of file attributes.

See Also

TFileListView

5.4.2 FileCtrl.TSelectDirExtOpts Type

Pascal

```
TSelectDirExtOpts = set of TSelectDirExtOpt;
```

C++

```
TSelectDirExtOpt TSelectDirExtOpts;
```

File

FileCtrl

Description

This is type FileCtrl.TSelectDirExtOpts.

5.4.3 FileCtrl.TSelectDirOpts Type

Pascal

```
TSelectDirOpts = set of TSelectDirOpt;
```

C++

```
TSelectDirOpt TSelectDirOpts;
```

File

FileCtrl

Description

This is type FileCtrl.TSelectDirOpts.

5.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WNTYPE_DRIVE (see page 1889)	This is constant FileCtrl.WNTYPE_DRIVE.

5.5.1 FileCtrl.WNTYPE_DRIVE Constant

Pascal

```
WNTYPE_DRIVE = 1;
```

C++

```
WNTYPE_DRIVE = 1;
```

File

FileCtrl

Description

This is constant FileCtrl.WNTYPE_DRIVE.

6 FMTBcd Namespace

This is namespace FMTBcd.

Topics

Name	Description
Classes (see page 1894)	The following table lists classes in this documentation.
Functions (see page 1896)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1913)	The following table lists structs, records, enums in this documentation.
Types (see page 1914)	The following table lists types in this documentation.
Constants (see page 1915)	The following table lists constants in this documentation.

Classes

	Name	Description
	EBcdException (see page 1894)	EBcdException is the exception class for errors that occur when working with BCD values.
	EBcdOverflowException (see page 1894)	EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the requested precision and number of decimals.

Constants

Name	Description
_DefaultDecimals (see page 1915)	This is constant FMTBcd._DefaultDecimals.
_NoDecimal (see page 1915)	This is constant FMTBcd._NoDecimal.
DefaultFMTBcdScale (see page 1916)	This is constant FMTBcd.DefaultFMTBcdScale.
MaxBcdPrecision (see page 1916)	This is constant FMTBcd.MaxBcdPrecision.
MaxBcdScale (see page 1916)	This is constant FMTBcd.MaxBcdScale.
MaxFMTBcdDigits (see page 1917)	This is constant FMTBcd.MaxFMTBcdDigits.
MaxFMTBcdFractionSize (see page 1917)	This is constant FMTBcd.MaxFMTBcdFractionSize.
MaxStringDigits (see page 1917)	This is constant FMTBcd.MaxStringDigits.
NullBcd (see page 1918)	Represents a NULL TBcd (see page 1913) value.
SizeOfFraction (see page 1918)	This is constant FMTBcd.SizeOfFraction.
SizeOfTBCD (see page 1918)	This is constant FMTBcd.SizeOfTBCD.

Functions

	Name	Description
	BcdAdd (see page 1897)	Adds two TBcd (see page 1913) values.
	BcdCompare (see page 1897)	Compares two TBcd (see page 1913) values.

	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdFromBytes (see page 1899)	This is function FMTBcd.BcdFromBytes.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdPrecision (see page 1900)	Returns the precision of a binary-coded decimal (BCD) value.
	BcdScale (see page 1900)	Returns the number of decimal places for a binary-coded decimal (BCD) value.
	BcdSubtract (see page 1901)	Subtracts one BCD value from another.
	BcdToBytes (see page 1901)	This is function FMTBcd.BcdToBytes.
	BCDToCurr (see page 1901)	Converts a binary-coded decimal value (BCD) to the corresponding Currency value.
	BcdToDouble (see page 1902)	Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.
	BcdToInteger (see page 1902)	Converts a binary-coded decimal value (BCD) to an integer.
	BcdToStr (see page 1903)	Returns a string representation of a binary-coded decimal (BCD) value.
	BcdToStrF (see page 1903)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	CurrToBCD (see page 1904)	Converts a Currency value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 1905)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 1905)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	FormatBcd (see page 1905)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	IntegerToBcd (see page 1906)	Converts an integer value to the corresponding binary-coded decimal (BCD) value.
	IsBcdNegative (see page 1906)	Indicates whether a binary-coded decimal (BCD) value is less than 0.
	NormalizeBcd (see page 1907)	Converts a BCD value into another BCD value with a specified precision and number of decimals.
	OldBCDToCurr (see page 1907)	This is function FMTBcd.OldBCDToCurr.
	OldCurrToBCD (see page 1908)	This is function FMTBcd.OldCurrToBCD.
	RoundAt (see page 1908)	This is function FMTBcd.RoundAt.
	StrToBcd (see page 1908)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	TryStrToBcd (see page 1909)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	VarFMTBcd (see page 1910)	Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarIsFMTBcd (see page 1911)	Indicates whether a specified Variant represents a TBcd (see page 1913) value.
	VarToBcd (see page 1911)	Converts a Variant into a TBcd (see page 1913) type.

Structs, Records, Enums

	Name	Description
	TBcd (see page 1913)	TBcd stores a binary coded decimal value.

Types

Name	Description
PBcd (see page 1914)	This is type FMTBcd.PBcd.
TNibbleArray (see page 1914)	This is type FMTBcd.TNibbleArray.

6.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EBcdException (see page 1894)	EBcdException is the exception class for errors that occur when working with BCD values.
	EBcdOverflowException (see page 1894)	EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the requested precision and number of decimals.

6.1.1 EBcdException Class

EBcdException is the exception class for errors that occur when working with BCD values.

Class Hierarchy

Pascal

```
EBcdException = class(Exception);
```

C++

```
class EBcdException : public Exception;
```

File

FMTBcd

Description

EBcdException is raised when

An attempt to convert a Variant or string to a BCD value is unsuccessful.

An application tries to create a Variant that represents a BCD value, starting with a value that can't be successfully converted.

An application tries to divide by a BCD type whose value is 0.

See Also

VarFMTBcdCreate (see page 1910)

TBcd (see page 1913)

StrToBCD

BCDDivide

6.1.2 EBcdOverflowException Class

EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the

requested precision and number of decimals.

Class Hierarchy

Pascal

```
EBcdOverflowException = class(EBcdException);
```

C++

```
class EBcdOverflowException : public EBcdException;
```

File

FMTBcd

Description

EBcdOverflowException is raised when

A call to VarFMTBcdCreate (see page 1910) tries to create a BCD value with a precision and number of decimals that can't be supported.

An application tries to cast a Variant to a BCD value but it can't be expressed using the default precision and number of decimals.

See Also

VarFMTBcdCreate (see page 1910)

TBcd (see page 1913)

6.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	BcdAdd (see page 1897)	Adds two TBcd (see page 1913) values.
	BcdCompare (see page 1897)	Compares two TBcd (see page 1913) values.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdDivide (see page 1898)	Divides one BCD value by another.
	BcdFromBytes (see page 1899)	This is function FMTBcd.BcdFromBytes.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdMultiply (see page 1899)	Multiplies two BCD values.
	BcdPrecision (see page 1900)	Returns the precision of a binary-coded decimal (BCD) value.
	BcdScale (see page 1900)	Returns the number of decimal places for a binary-coded decimal (BCD) value.
	BcdSubtract (see page 1901)	Subtracts one BCD value from another.
	BcdToBytes (see page 1901)	This is function FMTBcd.BcdToBytes.
	BCDToCurr (see page 1901)	Converts a binary-coded decimal value (BCD) to the corresponding Currency value.
	BcdToDouble (see page 1902)	Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.
	BcdToInteger (see page 1902)	Converts a binary-coded decimal value (BCD) to an integer.
	BcdToStr (see page 1903)	Returns a string representation of a binary-coded decimal (BCD) value.
	BcdToStrF (see page 1903)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	CurrToBCD (see page 1904)	Converts a Currency value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 1905)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 1905)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	FormatBcd (see page 1905)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	IntegerToBcd (see page 1906)	Converts an integer value to the corresponding binary-coded decimal (BCD) value.
	IsBcdNegative (see page 1906)	Indicates whether a binary-coded decimal (BCD) value is less than 0.
	NormalizeBcd (see page 1907)	Converts a BCD value into another BCD value with a specified precision and number of decimals.
	OldBCDToCurr (see page 1907)	This is function FMTBcd.OldBCDToCurr.
	OldCurrToBCD (see page 1908)	This is function FMTBcd.OldCurrToBCD.
	RoundAt (see page 1908)	This is function FMTBcd.RoundAt.
	StrToBcd (see page 1908)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	TryStrToBcd (see page 1909)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	VarFMTBcd (see page 1910)	Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.

	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarFMTBcdCreate (see page 1910)	Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.
	VarIsFMTBcd (see page 1911)	Indicates whether a specified Variant represents a TBcd (see page 1913) value.
	VarToBcd (see page 1911)	Converts a Variant into a TBcd (see page 1913) type.

6.2.1 FMTBcd.BcdAdd Function

Adds two TBcd (see page 1913) values.

Pascal

```
procedure BcdAdd(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd);
```

C++

```
BcdAdd(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdAdd adds bcdIn1 to bcdIn2 and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the addition. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 1907) procedure.

See Also

BcdSubtract

BcdMultiply

BcdDivide

NormalizeBcd

6.2.2 FMTBcd.BcdCompare Function

Compares two TBcd (see page 1913) values.

Pascal

```
function BcdCompare(const bcd1: TBcd; const bcd2: TBcd): Integer;
```

C++

```
int BcdCompare(const TBcd bcd1, const TBcd bcd2);
```

File

FMTBcd

Description

Call BcdCompare to compare two TBcd (see page 1913) values. The return value is indicated in the following table:

Return value	Condition
-1	bcd1< bcd2
0	bcd1= bcd2
1	bcd1 > bcd2

See Also

CompareValue

6.2.3 FMTBcd.BcdDivide Function

Divides one BCD value by another.

Pascal

```
procedure BcdDivide(Dividend: string; Divisor: string; var bcdOut: TBcd); overload;
procedure BcdDivide(const Dividend: TBcd; const Divisor: TBcd; var bcdOut: TBcd); overload;
procedure BcdDivide(const Dividend: TBcd; const Divisor: Double; var bcdOut: TBcd); overload;
procedure BcdDivide(const Dividend: TBcd; const Divisor: string; var bcdOut: TBcd); overload;
```

C++

```
BcdDivide(UnicodeString Dividend, UnicodeString Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const TBcd Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const Double Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const UnicodeString Divisor, TBcd bcdOut);
```

File

FMTBcd

Description

BcdDivide divides Dividend by Divisor and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the division, if possible. (The number of decimal places in the result, however, will never exceed 10.) You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 1907) procedure.

See Also

BcdSubtract

BcdMultiply

BcdAdd

NormalizeBcd

6.2.4 FMTBcd.BcdFromBytes Function

Pascal

```
function BcdFromBytes(const AValue: TBytes): TBcd;
```

C++

```
TBcd BcdFromBytes(const TBytes AValue);
```

File

FMTBcd

Description

This is function FMTBcd.BcdFromBytes.

6.2.5 FMTBcd.BcdMultiply Function

Multiplies two BCD values.

Pascal

```
procedure BcdMultiply(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd); overload;
procedure BcdMultiply(const bcdIn: TBcd; const DoubleIn: Double; var bcdOut: TBcd); overload;
procedure BcdMultiply(const bcdIn: TBcd; const StringIn: string; var bcdOut: TBcd); overload;
procedure BcdMultiply(StringIn1: string; StringIn2: string; var bcdOut: TBcd); overload;
```

C++

```
BcdMultiply(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
BcdMultiply(const TBcd bcdIn, const Double DoubleIn, TBcd bcdOut);
BcdMultiply(const TBcd bcdIn, const UnicodeString StringIn, TBcd bcdOut);
BcdMultiply(UnicodeString StringIn1, UnicodeString StringIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdMultiply multiplies the first two arguments and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the multiplication, if possible. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 1907) procedure.

See Also

BcdSubtract

BcdDivide

BcdAdd

NormalizeBcd

6.2.6 FMTBcd.BcdPrecision Function

Returns the precision of a binary-coded decimal (BCD) value.

Pascal

```
function BcdPrecision(const Bcd: TBcd): Word;
```

C++

```
Word BcdPrecision(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdPrecision to determine the number of significant digits in a specified BCD value.

See Also

BcdScale

IsBcdNegative

NormalizeBcd

6.2.7 FMTBcd.BcdScale Function

Returns the number of decimal places for a binary-coded decimal (BCD) value.

Pascal

```
function BcdScale(const Bcd: TBcd): Word;
```

C++

```
Word BcdScale(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdScale to determine the number of decimal places in a specified BCD value.

See Also

BcdPrecision

IsBcdNegative

NormalizeBcd

6.2.8 FMTBcd.BcdSubtract Function

Subtracts one BCD value from another.

Pascal

```
procedure BcdSubtract(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd);
```

C++

```
BcdSubtract(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdSubtract subtracts bcdIn2 from bcdIn1 and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the subtraction. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 1907) procedure.

See Also

BcdMultiply

BcdDivide

BcdAdd

NormalizeBcd

6.2.9 FMTBcd.BcdToBytes Function

Pascal

```
function BcdToBytes(const Value: TBcd): TBytes;
```

C++

```
TBytes BcdToBytes(const TBcd Value);
```

File

FMTBcd

Description

This is function FMTBcd.BcdToBytes.

6.2.10 FMTBcd.BCDToCurr Function

Converts a binary-coded decimal value (BCD) to the corresponding Currency value.

Pascal

```
function BCDToCurr(const BCD: TBcd; var Curr: Currency): Boolean;
```

C++

```
Boolean BCDToCurr(const TBcd BCD, Currency Curr);
```

File

FMTBcd

Description

Use BCDToCurr to convert a binary-coded decimal value, such as is stored in a physical BCD database field, into a Currency value, which is the representation used by TBCDField objects. BCDToCurr returns true if the BCD value passed as the BCD parameter is successfully converted into a Currency object which is passed out as the Curr parameter.

See Also

CurrToBCD

TBCDField

TField

6.2.11 FMTBcd.BcdToDouble Function

Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.

Pascal

```
function BcdToDouble(const Bcd: TBcd): Double;
```

C++

```
Double BcdToDouble(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdToDouble to convert a binary-coded decimal value into a double precision floating point value. If the Bcd parameter represents a value that can't be expressed in double precision, BcdToDouble raises an exception.

See Also

DoubleToBcd

BCDToCurr

BCDToInteger

6.2.12 FMTBcd.BcdToInteger Function

Converts a binary-coded decimal value (BCD) to an integer.

Pascal

```
function BcdToInteger(const Bcd: TBcd; Truncate: Boolean = False): Integer;
```

C++

```
int BcdToInteger(const TBcd Bcd, Boolean Truncate = False);
```

File

FMTBcd

Description

Use BcdToInteger to convert a binary-coded decimal value into an integer.

Bcd is the value to convert.

Truncate indicates how to handle values that include a decimal portion. If Truncate is true, BcdToInteger truncates the value specified by Bcd and converts only the integer portion. If Truncate is false, BcdToInteger raises an exception if the Bcd parameter does not represent an integer value.

See Also

IntegerToBcd

BCDToCurr

BCDToDouble

6.2.13 FMTBcd.BcdToStr Function

Returns a string representation of a binary-coded decimal (BCD) value.

Pascal

```
function BcdToStr(const Bcd: TBcd): string; overload;
```

C++

```
UnicodeString BcdToStr(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdToStr to convert a binary-coded decimal value into a string. BcdToStr formats the string to display all of the digits stored in the BCD value.

See Also

BcdToStrF

StrToBcd

TryStrToBcd

6.2.14 FMTBcd.BcdToStrF Function

Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.

Pascal

```
function BcdToStrF(const Bcd: TBcd; Format: TFloatFormat; const Precision: Integer; const
Digits: Integer): string;
```

C++

```
UnicodeString BcdToStrF(const TBcd Bcd, TFloatFormat Format, const int Precision, const int
Digits);
```

File

FMTBcd

Description

Use BcdToStrF to convert a binary-coded decimal value into a string.

Value is the BCD value to convert to a string.

Format is the formatting code that indicates the overall type of formatting to use.

Precision indicates the maximum number of significant digits in the resulting string.

Digits indicates the maximum number of digits past the decimal point in the resulting string.

See Also

BcdToStr

StrToBcd

FloatToStrF

FormatBcd

TryStrToBcd

6.2.15 FMTBcd.CurrToBCD Function

Converts a Currency value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function CurrToBCD(const Curr: Currency; var BCD: TBcd; Precision: Integer = 32; Decimals:
Integer = 4): Boolean;
```

C++

```
Boolean CurrToBCD(const Currency Curr, TBcd BCD, int Precision = 32, int Decimals = 4);
```

File

FMTBcd

Description

Use CurrToBCD to convert a Currency value, such as is used by TBCDField objects, into a binary-coded decimal value, such as is stored in a physical BCD database field. CurrToBCD returns true if the Currency object passed as the Curr parameter is successfully converted into a BCD value and passed out as the BCD parameter. The Precision parameter specifies the number of digits represented in the BCD value, and the Decimals parameter specifies the number of those digits that are after the decimal point.

See Also

BCDToCurr

TBCDField

TField

6.2.16 FMTBcd.DoubleToBcd Function

Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function DoubleToBcd(const AValue: Double): TBcd; overload;
procedure DoubleToBcd(const AValue: Double; var bcd: TBcd); overload;
```

C++

```
TBcd DoubleToBcd(const Double AValue);
DoubleToBcd(const Double AValue, TBcd bcd);
```

File

FMTBcd

Description

Use DoubleToBcd to convert a double precision floating point value into a binary-coded decimal value. DoubleToBcd infers the number of decimal places and significant digits from the specified value aValue, using only as many as necessary. It never sets the number of significant digits to a value greater than 15.

aValue is the value to convert.

bcd returns the resulting BCD value when using the second syntax. When using the first syntax, the BCD value is the return value of the function.

See Also

BCDToDouble

CurrToBcd

IntegerToBcd

6.2.17 FMTBcd.FormatBcd Function

Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.

Pascal

```
function FormatBcd(const Format: string; Bcd: TBcd): string;
```

C++

```
UnicodeString FormatBcd(const UnicodeString Format, TBcd Bcd);
```

File

FMTBcd

Description

Use FormatBcd to convert a binary-coded decimal value into a string.

Format indicates how the resulting string should be formatted. It uses the same format specifiers as pertain to the FormatFloat function.

Bcd is the value to convert to a string.

See Also

BcdToStrF

FormatFloat

BcdToStr

TryStrToBcd

StrToBcd

6.2.18 FMTBcd.IntegerToBcd Function

Converts an integer value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function IntegerToBcd(const AValue: Integer): TBcd;
```

C++

```
TBcd IntegerToBcd(const int AValue);
```

File

FMTBcd

Description

Use IntegerToBcd to convert an integer into a binary-coded decimal value.

See Also

BCDToInteger

CurrToBcd

DoubleToBcd

6.2.19 FMTBcd.IsBcdNegative Function

Indicates whether a binary-coded decimal (BCD) value is less than 0.

Pascal

```
function IsBcdNegative(const Bcd: TBcd): Boolean;
```

C++

```
Boolean IsBcdNegative(const TBcd Bcd);
```

File

FMTBcd

Description

IsBcdNegative returns true if Bcd is less than 0, false if Bcd is greater than or equal to 0.

See Also

BcdPrecision

BcdScale

NormalizeBcd

6.2.20 FMTBcd.NormalizeBcd Function

Converts a BCD value into another BCD value with a specified precision and number of decimals.

Pascal

```
function NormalizeBcd(const InBcd: TBcd; var OutBcd: TBcd; const Prec: Word; const Scale:
Word): Boolean;
```

C++

```
Boolean NormalizeBcd(const TBcd InBcd, TBcd OutBcd, const Word Prec, const Word Scale);
```

File

FMTBcd

Description

Call NormalizeBcd to give a TBcd (see page 1913) value a specific precision and number of decimal places. If the specified precision or number of decimal places is less than that of the original BCD value, NormalizeBcd truncates the value (as opposed to rounding).

InBcd is the BCD value to convert.

OutBcd returns the resulting BCD value with the specified precision and decimal places.

Prec is the number of significant digits in OutBcd.

Scale is the number of decimal places in OutBcd.

NormalizeBcd returns true if the conversion can be made with no loss of precision. It returns false if the conversion required some digits to be truncated.

6.2.21 FMTBcd.OldBCDToCurr Function

Pascal

```
function OldBCDToCurr(const BCD: TBcd; var Curr: Currency): Boolean;
```

C++

```
Boolean OldBCDToCurr(const TBcd BCD, Currency Curr);
```

File

FMTBcd

Description

This is function FMTBcd.OldBCDToCurr.

6.2.22 FMTBcd.OldCurrToBCD Function

Pascal

```
function OldCurrToBCD(const Curr: Currency; var BCD: TBcd; Precision: Integer = 32; Decimals:
Integer = 4): Boolean;
```

C++

```
Boolean OldCurrToBCD(const Currency Curr, TBcd BCD, int Precision = 32, int Decimals = 4);
```

File

FMTBcd

Description

This is function FMTBcd.OldCurrToBCD.

6.2.23 FMTBcd.RoundAt Function

Pascal

```
function RoundAt(const Value: string; Position: SmallInt): string;
```

C++

```
UnicodeString RoundAt(const UnicodeString Value, SmallInt Position);
```

File

FMTBcd

Description

This is function FMTBcd.RoundAt.

6.2.24 FMTBcd.StrToBcd Function

Converts a string to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function StrToBcd(const AValue: string): TBcd;
```

C++

```
TBcd StrToBcd(const UnicodeString AValue);
```

File

FMTBcd

Description

Use StrToBcd to convert a string into a binary-coded decimal value. The decimals and precision are inferred from the digits present in the string.

StrToBcd converts the string specified by aValue and returns the resulting BCD value. If aValue does not represent a value that can be expressed as a TBcd (see page 1913), StrToBcd raises an exception.

Tip: To convert a string to a Bcd value without raising an exception for invalid strings, use the TryStrToBcd (see page 1909) function instead.

See Also

BcdToStr

BcdToStrF

FormatBcd

TryStrToBcd

6.2.25 FMTBcd.TryStrToBcd Function

Converts a string to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function TryStrToBcd(const AValue: string; var Bcd: TBcd): Boolean;
```

C++

```
Boolean TryStrToBcd(const UnicodeString AValue, TBcd Bcd);
```

File

FMTBcd

Description

Use TryStrToBcd to convert a string into a binary-coded decimal value. The decimals and precision are inferred from the digits present in the string.

TryStrToBcd converts the string specified by aValue and returns the resulting BCD value as Bcd.

TryStrToBcd returns true if it successfully converts aValue to a TBcd (see page 1913) value. If aValue can't be successfully converted to a TBcd (see page 1913) value, TryStrToBcd returns false.

See Also

BcdToStr

BcdToStrF

FormatBcd

StrToBcd

6.2.26 FMTBcd.VarFMTBcd Function

Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 1913) value.

Pascal

```
function VarFMTBcd: TVarType;
```

C++

```
TVarType VarFMTBcd();
```

File

FMTBcd

Description

VarFMTBcd returns the Variant type code that identifies a TBcd (see page 1913)-valued Variant. This is the value that the VarType function returns when the Variant stores its value as a TBcd (see page 1913) value.

The Variant type codes for custom Variants are dynamically allocated when the application loads the unit that implements them. VarFMTBcd allows the application to identify the Variant type code that is currently used for custom Variants that represent a BCD value. This type code, however, may not match the type code used by another application that uses a different set of custom Variant types.

See Also

VarIsFMTBcd

VarType

TBcd

VarToBcd

6.2.27 FMTBcd.VarFMTBcdCreate Function

Creates a new Variant that stores its value internally as a TBcd (see page 1913) value.

Pascal

```
procedure VarFMTBcdCreate(var ADest: Variant; const ABcd: TBcd); overload;
function VarFMTBcdCreate: Variant; overload;
function VarFMTBcdCreate(const AValue: string; Precision: Word; Scale: Word): Variant;
overload;
function VarFMTBcdCreate(const AValue: Double; Precision: Word = 18; Scale: Word = 4):
Variant; overload;
function VarFMTBcdCreate(const ABcd: TBcd): Variant; overload;
```

C++

```
VarFMTBcdCreate(Variant ADest, const TBcd ABcd);
Variant VarFMTBcdCreate();
Variant VarFMTBcdCreate(const UnicodeString AValue, Word Precision, Word Scale);
Variant VarFMTBcdCreate(const Double AValue, Word Precision = 18, Word Scale = 4);
Variant VarFMTBcdCreate(const TBcd ABcd);
```

File

FMTBcd

Description

Call VarBcdCreate to obtain a Variant that stores its value as a TBcd (see page 1913) value.

Using the first syntax, the Variant returned as aDest has the same value as ABcd.

Using the second syntax (passing no arguments), the returned Variant has a value of NullBcd (see page 1918).

Using the third syntax, the returned Variant has the same value as ABcd.

Using the fourth syntax, the returned Variant converts AValue (if necessary) to extract the value.

Using the remaining syntaxes, the AValue parameter is converted into a TBcd (see page 1913) value with the specified Precision and Scale and the result is the value of the Variant.

See Also

VarIsFMTBcd

VarToBcd

NullBcd

6.2.28 FMTBcd.VarIsFMTBcd Function

Indicates whether a specified Variant represents a TBcd (see page 1913) value.

Pascal

```
function VarIsFMTBcd(const AValue: Variant): Boolean; overload;
```

C++

```
Boolean VarIsFMTBcd(const Variant AValue);
```

File

FMTBcd

Description

VarIsBcd returns true if the Variant aValue stores its value internally as a TBcd (see page 1913) value. VarIsBcd returns false otherwise.

See Also

VarToBcd

TBcd

VarFMTBcd

6.2.29 FMTBcd.VarToBcd Function

Converts a Variant into a TBcd (see page 1913) type.

Pascal

```
function VarToBcd(const AValue: Variant): TBcd;
```

C++

```
TBcd VarToBcd(const Variant AValue);
```

File

FMTBcd

Description

Use VarDoubleAsBcd to obtain a TBcd (see page 1913) value from a Variant.

aValue is the Variant to convert.

See Also

VarIsFMTBcd

6.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TBcd (see page 1913)	TBcd stores a binary coded decimal value.

6.3.1 FMTBcd.TBcd Record

TBcd stores a binary coded decimal value.

Pascal

```
TBcd = packed record
  Precision: Byte;
  SignSpecialPlaces: Byte;
  Fraction: packed array [0..31] of Byte;
end;
```

C++

```
struct TBcd {
  Byte Precision;
  Byte SignSpecialPlaces;
  packed array [0..31] of Byte Fraction;
};
```

File

FMTBcd

Description

TBcd represents a binary coded decimal value. It contains the following fields:

Field	Contents
Precision	The number of digits in the decimal version of the value.
SignSpecialPlaces	The sign bit (0 is positive, any other value negative)
Fraction	An array of BCD nibbles, 00 to 99 per byte, high nibble first. Only the first Precision nibbles are used.

You can create a Variant that represents a TBcd value using the VarFMTBcdCreate (see page 1910) function. In fact, the easiest way to manipulate TBcd values is to create Variants for them and use the built-in operators provided by the Variant type.

6.4 Types

The following table lists types in this documentation.

Types

Name	Description
PBcd (see page 1914)	This is type FMTBcd.PBcd.
TNibbleArray (see page 1914)	This is type FMTBcd.TNibbleArray.

6.4.1 FMTBcd.PBcd Type

Pascal

```
PBcd = ^TBcd;
```

C++

```
^TBcd PBcd;
```

File

FMTBcd

Description

This is type FMTBcd.PBcd.

6.4.2 FMTBcd.TNibbleArray Type

Pascal

```
TNibbleArray = array[0..63] of Byte;
```

C++

```
array[0..63] of Byte TNibbleArray;
```

File

FMTBcd

Description

This is type FMTBcd.TNibbleArray.

6.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
_DefaultDecimals (see page 1915)	This is constant FMTBcd._DefaultDecimals.
_NoDecimal (see page 1915)	This is constant FMTBcd._NoDecimal.
DefaultFMTBcdScale (see page 1916)	This is constant FMTBcd.DefaultFMTBcdScale.
MaxBcdPrecision (see page 1916)	This is constant FMTBcd.MaxBcdPrecision.
MaxBcdScale (see page 1916)	This is constant FMTBcd.MaxBcdScale.
MaxFMTBcdDigits (see page 1917)	This is constant FMTBcd.MaxFMTBcdDigits.
MaxFMTBcdFractionSize (see page 1917)	This is constant FMTBcd.MaxFMTBcdFractionSize.
MaxStringDigits (see page 1917)	This is constant FMTBcd.MaxStringDigits.
NullBcd (see page 1918)	Represents a NULL TBcd (see page 1913) value.
SizeOfFraction (see page 1918)	This is constant FMTBcd.SizeOfFraction.
SizeOfTBCD (see page 1918)	This is constant FMTBcd.SizeOfTBCD.

6.5.1 FMTBcd._DefaultDecimals Constant

Pascal

```
_DefaultDecimals = 10;
```

C++

```
_DefaultDecimals = 10;
```

File

FMTBcd

Description

This is constant FMTBcd._DefaultDecimals.

6.5.2 FMTBcd._NoDecimal Constant

Pascal

```
_NoDecimal = -255;
```

C++

```
_NoDecimal = -255;
```

File

FMTBcd

Description

This is constant FMTBcd._NoDecimal.

6.5.3 FMTBcd.DefaultFMTBcdScale Constant

Pascal

```
DefaultFMTBcdScale = 6;
```

C++

```
DefaultFMTBcdScale = 6;
```

File

FMTBcd

Description

This is constant FMTBcd.DefaultFMTBcdScale.

6.5.4 FMTBcd.MaxBcdPrecision Constant

Pascal

```
MaxBcdPrecision = 18;
```

C++

```
MaxBcdPrecision = 18;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxBcdPrecision.

6.5.5 FMTBcd.MaxBcdScale Constant

Pascal

```
MaxBcdScale = 4;
```

C++

```
MaxBcdScale = 4;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxBcdScale.

6.5.6 FMTBcd.MaxFMTBcdDigits Constant

Pascal

```
MaxFMTBcdDigits = 32;
```

C++

```
MaxFMTBcdDigits = 32;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxFMTBcdDigits.

6.5.7 FMTBcd.MaxFMTBcdFractionSize Constant

Pascal

```
MaxFMTBcdFractionSize = 64;
```

C++

```
MaxFMTBcdFractionSize = 64;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxFMTBcdFractionSize.

6.5.8 FMTBcd.MaxStringDigits Constant

Pascal

```
MaxStringDigits = 100;
```

C++

```
MaxStringDigits = 100;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxStringDigits.

6.5.9 FMTBcd.NullBcd Constant

Represents a NULL TBcd (see page 1913) value.

Pascal

```
NullBcd: TBcd = (Precision: 0; SignSpecialPlaces: 0; Fraction:
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0));
```

C++

```
TBcd NullBcd = (Precision: 0; SignSpecialPlaces: 0; Fraction:
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0));
```

File

FMTBcd

Description

Use NullBcd to indicate an unknown or missing TBcd (see page 1913) value. NullBcd is a TBcd (see page 1913) with all its fields set to 0.

See Also

Null

6.5.10 FMTBcd.SizeOfFraction Constant

Pascal

```
SizeOfFraction = 32;
```

C++

```
SizeOfFraction = 32;
```

File

FMTBcd

Description

This is constant FMTBcd.SizeOfFraction.

6.5.11 FMTBcd.SizeOfTBCD Constant

Pascal

```
SizeOfTBCD = 2 + SizeOfFraction;
```

C++

```
SizeOfTBCD = 2 + SizeOfFraction;
```

File

FMTBcd

Description

This is constant FMTBcd.SizeOfTBCD.

7 Forms Namespace

This is namespace Forms.

Topics

Name	Description
Classes (see page 1924)	The following table lists classes in this documentation.
Interfaces (see page 2566)	The following table lists interfaces in this documentation.
Functions (see page 2574)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 2582)	The following table lists structs, records, enums in this documentation.
Types (see page 2595)	The following table lists types in this documentation.
Variables (see page 2612)	The following table lists variables in this documentation.

Classes

	Name	Description
	TApplication (see page 1924)	TApplication is the type used for a GUI windowed application.
	TControlScrollBar (see page 2009)	TControlScrollBar displays a horizontal or vertical scroll bar.
	TCustomActiveForm (see page 2024)	TCustomActiveForm is the base class for Active (see page 2145) Forms.
	TCustomDockForm (see page 2092)	TCustomDockForm is the default floating dock site class for TControl objects.
	TCustomForm (see page 2103)	TCustomForm is the base class from which to derive a window such as a form or dialog.
	TCustomFrame (see page 2237)	TCustomFrame is the base class from which TFrame (see page 2366) descends.
	TForm (see page 2241)	TForm represents a standard application window (form).
	TFrame (see page 2366)	TFrame is a container for components; it can be nested within forms or other frames.
	TGlassFrame (see page 2435)	TGlassFrame controls Windows Vista Aero glass effects.
	TMonitor (see page 2439)	TMonitor represents a monitor on which an application runs.
	TScreen (see page 2445)	TScreen represents the state of the screen in which an application runs.
	TScrollBox (see page 2483)	TScrollBox represents a scrolling area (a scroll box) in a window.
	TScrollingWinControl (see page 2554)	TScrollingWinControl is the base class for controls that support scrolling.

Functions

	Name	Description
	DisableTaskWindows (see page 2574)	This is function Forms.DisableTaskWindows.
	DoneCtl3D (see page 2574)	This is function Forms.DoneCtl3D.
	EnableTaskWindows (see page 2575)	This is function Forms.EnableTaskWindows.
	ForegroundTask (see page 2575)	Indicates whether the current thread is running in the foreground.

	GetParentForm (see page 2576)	Returns the form or property page that contains a specified control.
	InitCtl3D (see page 2576)	This is function Forms.InitCtl3D.
	IsAccel (see page 2576)	Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.
	KeyboardStateToShiftState (see page 2577)	Convert array of virtual keys to TShiftState.
	KeyboardStateToShiftState (see page 2577)	Convert array of virtual keys to TShiftState.
	KeyDataToShiftState (see page 2578)	Convert keyboard message data to TShiftState.
	KeysToShiftState (see page 2578)	Convert mouse event parameter to TShiftState.
	RestoreFocusState (see page 2579)	This is function Forms.RestoreFocusState.
	SaveFocusState (see page 2579)	This is function Forms.SaveFocusState.
	SetAutoSubClass (see page 2579)	This is function Forms.SetAutoSubClass.
	Subclass3DDlg (see page 2580)	This is function Forms.Subclass3DDlg.
	Subclass3DWnd (see page 2580)	This is function Forms.Subclass3DWnd.
	ValidParentForm (see page 2580)	Returns the form or property page that contains a specified control.

Interfaces

	Name	Description
	IDesignerHook (see page 2566)	IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.
	IOleForm (see page 2572)	IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

Structs, Records, Enums

	Name	Description
	TActiveFormBorderStyle (see page 2582)	Specifies the style of an active form's border.
	TBorderIcon (see page 2583)	TBorderIcons (see page 2596) and TBorderIcons (see page 2596) specify the buttons that appear on the title bar of a form window.
	TCloseAction (see page 2584)	TCloseAction describes how a form should respond when it is closed.
	TCMHintShowPause (see page 2584)	This is record Forms.TCMHintShowPause.
	TCursorRec (see page 2585)	This is record Forms.TCursorRec.
	TDefaultMonitor (see page 2585)	Specifies the monitor on which the form appears.
	TFormBorderStyle (see page 2586)	TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 2597) is the type of the BorderStyle property for other controls in windows-only applications.
	TFormStyle (see page 2587)	TFormStyle describes form style.
	TMonitorDefaultTo (see page 2587)	TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.
	TPopupForm (see page 2588)	This is record Forms.TPopupForm.
	TPopupMode (see page 2588)	TPopupMode enumerates the possible values of a Form's PopupMode property.
	TPopupWnd (see page 2589)	This is record Forms.TPopupWnd.
	TPosition (see page 2590)	TPosition describes the positioning of a form.
	TPrintScale (see page 2591)	Represents the proportions of a printed form.
	TScrollBarKind (see page 2591)	TScrollBarKind indicates the orientation of a scrollbar.
	TScrollBarStyle (see page 2592)	TScrollBarStyle specifies a scroll bar style.
	TShowAction (see page 2592)	This is record Forms.TShowAction.
	TTileMode (see page 2593)	Represents how MDI child forms are arranged when the Tile method is called.
	TTimerMode (see page 2593)	This is record Forms.TTimerMode.
	TWindowState (see page 2594)	TWindowState indicates whether a form is maximized, minimized, or normally sized.

Types

Name	Description
PCursorRec (see page 2595)	This is type Forms.PCursorRec.
PHintInfo (see page 2596)	PHintInfo declares a pointer to a Help Window appearance record.
TBorderIcons (see page 2596)	TBorderIcons specifies the buttons that appear on the title bar of a form window.
TBorderStyle (see page 2597)	TBorderStyle is the type of the BorderStyle property for framed controls.

TCloseEvent (see page 2597)	TCloseEvent is the type of a form's OnClose event handle.
TCloseQueryEvent (see page 2598)	TCloseQueryEvent is used for event handlers that are called when a window is about to close.
TCursorRecType (see page 2598)	This is type Forms.TCursorRecType.
TCustomFormClass (see page 2599)	TCustomFormClass defines the metaclass for TCustomForm (see page 2103).
TCustomFrameClass (see page 2599)	TCustomFrameClass defines the metaclass for TCustomFrame (see page 2237).
TExceptionEvent (see page 2599)	TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.
TFocusState (see page 2600)	This is type Forms.TFocusState.
TFormClass (see page 2600)	TFormClass defines the metaclass for TForm (see page 2241).
TFormState (see page 2601)	TFormState describes the possible state of a form.
TGetHandleEvent (see page 2601)	This is type Forms.TGetHandleEvent.
THelpEvent (see page 2602)	THelpEvent is the type of an OnHelp event handler.
THintInfo (see page 2602)	THintInfo is used to define the appearance and behavior of the Help window.
TIdleEvent (see page 2604)	TIdleEvent is the type of event handler that perform actions when the application is idle.
TMessageEvent (see page 2605)	TMessageEvent is the type of an OnMessage event handler.
TPopupFormArray (see page 2605)	This is type Forms.TPopupFormArray.
TPopupWndArray (see page 2605)	This is type Forms.TPopupWndArray.
TScrollBarInc (see page 2606)	TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 2009).
TSetLayeredWindowAttributes (see page 2606)	This is type Forms.TSetLayeredWindowAttributes.
TSettingChangeEvent (see page 2607)	TSettingChangeEvent is the type of an OnSettingChange event handler.
TShortCutEvent (see page 2609)	TShortCutEvent is the type of an OnShortCut event handler.
TShowHintEvent (see page 2610)	TShowHintEvent is the type of the OnShowHint event handler.
TTaskWindowList (see page 2610)	This is type Forms.TTaskWindowList.
TWindowHook (see page 2611)	TWindowHook is the type for a dialog procedure.

Variables

Name	Description
Application (see page 2612)	Represents application-level information.
Ctl3DBtnWndProc (see page 2614)	This is variable Forms.Ctl3DBtnWndProc.
Ctl3DCtlColorEx (see page 2614)	This is variable Forms.Ctl3DCtlColorEx.
Ctl3DDlgFramePaint (see page 2614)	This is variable Forms.Ctl3DDlgFramePaint.
HintWindowClass (see page 2615)	Specifies the class for the window used to display help hints.
Screen (see page 2615)	Represents a screen device.
SetLayeredWindowAttributes (see page 2617)	This is variable Forms.SetLayeredWindowAttributes.

7.1 Classes

The following table lists classes in this documentation.

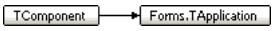
Classes

	Name	Description
	TApplication (see page 1924)	TApplication is the type used for a GUI windowed application.
	TControlScrollBar (see page 2009)	TControlScrollBar displays a horizontal or vertical scroll bar.
	TCustomActiveForm (see page 2024)	TCustomActiveForm is the base class for Active (see page 2145) Forms (see page 1921).
	TCustomDockForm (see page 2092)	TCustomDockForm is the default floating dock site class for TControl objects.
	TCustomForm (see page 2103)	TCustomForm is the base class from which to derive a window such as a form or dialog.
	TCustomFrame (see page 2237)	TCustomFrame is the base class from which TFrame (see page 2366) descends.
	TForm (see page 2241)	TForm represents a standard application window (form).
	TFrame (see page 2366)	TFrame is a container for components; it can be nested within forms or other frames.
	TGlassFrame (see page 2435)	TGlassFrame controls Windows Vista Aero glass effects.
	TMonitor (see page 2439)	TMonitor represents a monitor on which an application runs.
	TScreen (see page 2445)	TScreen represents the state of the screen in which an application runs.
	TScrollBox (see page 2483)	TScrollBox represents a scrolling area (a scroll box) in a window.
	TScrollingWinControl (see page 2554)	TScrollingWinControl is the base class for controls that support scrolling.

7.1.1 TApplication Class

TApplication is the type used for a GUI windowed application.

Class Hierarchy



Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TApplication = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TApplication : public TComponent;
```

File

Forms

Description

TApplication encapsulates a windowed application. The methods and properties introduced in TApplication reflect the fundamentals established in the Windows operating system to create, run, sustain, and destroy an application. TApplication thereby simplifies the interface between the developer and the Windows environment. For this purpose, TApplication encapsulates behavior for

Windows message processing.

Context-sensitive online help.

Menu accelerator and key processing.

Exception handling.

Managing the fundamental parts defined by the Windows operating system for an application, such as MainWindow, WindowClass, and so on

Each GUI application automatically declares an Application (see page 2612) variable as the instance of the application. If the application is not a Web server application, control panel applet, or NT service application, this variable is of type TApplication.

TApplication does not appear on the Component palette, nor is it available in the form designer to visually manipulate; so it has no published properties. Nevertheless, some of its public properties can be set or modified at design time in the Forms (see page 1921) and Application (see page 2612) pages of the Project|Options dialog box. In addition, you can add a TApplicationEvents component to any form in the project to intercept the application's events, and supply event handlers using the IDE.

Note: For application-wide properties and methods that affect the display, see TScreen (see page 2445).

See Also

Application (see page 2612)

Exception

TComponent

TControl

TForm (see page 2241)

TWinControl

TScreen (see page 2445)

TApplicationEvents

Console Applications

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Controlling Application Behavior

Working at the Application Level

7.1.1.1 TApplication Members

The following tables list the members exposed by TApplication.

Protected Methods

	Name	Description
	Idle (see page 1945)	This is Idle, a member of class TApplication.
	IsDlgMsg (see page 1947)	This is IsDlgMsg, a member of class TApplication.

	IsHintMsg (see page 1947)	This is IsHintMsg, a member of class TApplication.
	IsKeyMsg (see page 1948)	This is IsKeyMsg, a member of class TApplication.
	IsMDIMsg (see page 1948)	This is IsMDIMsg, a member of class TApplication.
	IsPreProcessMessage (see page 1948)	This is IsPreProcessMessage, a member of class TApplication.
	IsShortCut (see page 1949)	This is IsShortCut, a member of class TApplication.

Public Events

	Name	Description
	OnActionExecute (see page 1990)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1991)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1992)	Occurs when an application becomes active.
	OnDeactivate (see page 1992)	Occurs when an application becomes inactive.
	OnException (see page 1993)	Occurs when an unhandled exception occurs in the application.
	OnGetActiveFormHandle (see page 1995)	This is OnGetActiveFormHandle, a member of class TApplication.
	OnGetMainFormHandle (see page 1995)	This is OnGetMainFormHandle, a member of class TApplication.
	OnHelp (see page 1995)	Occurs when the application receives a request for help.
	OnHint (see page 1997)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 1976).
	OnIdle (see page 1998)	Occurs when an application becomes idle.
	OnMessage (see page 2003)	Occurs when the application receives a Windows message.
	OnMinimize (see page 2004)	Occurs when an application is minimized.
	OnModalBegin (see page 2006)	Occurs when a modal form is opened.
	OnModalEnd (see page 2006)	Occurs when a modal form is closed.
	OnRestore (see page 2006)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 2007)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 2007)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 2008)	Occurs when the application is about to display the hint window for a Help Hint (see page 1976).

Public Methods

	Name	Description
	ActivateHint (see page 1930)	Displays a hint window for the control at a specified position.
	AddPopupForm (see page 1931)	This is AddPopupForm, a member of class TApplication.
	BringToFront (see page 1931)	Sets the last active window as the topmost window on the desktop above all other applications.
	CancelHint (see page 1931)	Cancels the display of a hint for a control.
	ControlDestroyed (see page 1931)	Notifies the application when a control is destroyed.
	Create (see page 1928)	Creates a new application.
	CreateForm (see page 1932)	Creates a new form.
	CreateHandle (see page 1934)	Generates a main window for the application if it does not already have one.
	Destroy (see page 1928)	Destroys the application instance and all of its associated memory.
	DoApplicationIdle (see page 1935)	Processes messages in the Message queue for certain events.
	ExecuteAction (see page 1935)	Generates an OnActionExecute (see page 1990) event.
	HandleException (see page 1935)	Provides default handling of exceptions for the application.
	HandleMessage (see page 1937)	Interrupts the execution of an application while Windows processes a message in the Windows message queue.
	HelpCommand (see page 1941)	Provides quick access to any of the Help commands in the WinHelp API (application programming interface).
	HelpContext (see page 1941)	Displays a specified help topic.
	HelpJump (see page 1942)	Displays a specified help topic.
	HelpKeyword (see page 1942)	Displays a specified help topic.
	HelpShowTableOfContents (see page 1943)	Shows the Table of Contents within the help system.
	HideHint (see page 1943)	Hides the current hint.
	HintMouseMessage (see page 1944)	Regulates the display of hint windows.

	HookMainWindow (see page 1944)	Enables a native Windows dialog box to receive messages sent to the application's main window.
	HookSynchronizeWakeup (see page 1944)	This is HookSynchronizeWakeup, a member of class TApplication.
	Initialize (see page 1945)	Provides an opportunity to initialize subsystems.
	IsRightToLeft (see page 1948)	Specifies whether the application adjusts its interface for Middle Eastern locales.
	MessageBox (see page 1949)	Displays a specified message to the user.
	Minimize (see page 1951)	Shrinks an application to the Windows task bar.
	ModalFinished (see page 1952)	Track closing of modal form.
	ModalStarted (see page 1952)	Track opening of modal form.
	NormalizeAllTopMosts (see page 1952)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	NormalizeTopMosts (see page 1953)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	ProcessMessages (see page 1954)	Interrupts the execution of an application so that it can process the message queue.
	RemovePopupForm (see page 1956)	This is RemovePopupForm, a member of class TApplication.
	Restore (see page 1957)	Restores a minimized application to its normal size.
	RestoreTopMosts (see page 1958)	Restores forms designated as fsStayOnTop to be topmost again.
	Run (see page 1959)	Executes the application.
	ShowException (see page 1961)	Displays a message box for exceptions that are not caught by application code.
	Terminate (see page 1962)	Ends application execution.
	UnhookMainWindow (see page 1964)	Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 1944) method.
	UnhookSynchronizeWakeup (see page 1964)	Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 1944).
	UpdateAction (see page 1965)	Generates an OnActionUpdate (see page 1991) event.
	UseRightToLeftAlignment (see page 1966)	Specifies whether the application object is in a right-to-left alignment mode.
	UseRightToLeftReading (see page 1966)	Specifies whether the application object is in a right-to-left text mode.
	UseRightToLeftScrollBar (see page 1966)	Specifies whether the vertical scroll bar appears on the left side controls.

Public Properties

	Name	Description
	ActionUpdateDelay (see page 1968)	Delays the call to DoActionIdle.
R	Active (see page 1968)	Specifies whether the application is active and has focus.
R	ActiveFormHandle (see page 1968)	Gives the Windows handle of the currently active form.
	AllowTesting (see page 1969)	Represents information for the IDE.
	AutoDragDocking (see page 1969)	Indicates whether dragged windows are automatically docked.
	BiDiKeyboard (see page 1969)	Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.
	BiDiMode (see page 1970)	Indicates the layout of the application when running under Middle Eastern versions of Windows.
R	CurrentHelpFile (see page 1970)	Indicates the current help file.
	DefaultFont (see page 1971)	This is DefaultFont, a member of class TApplication.
	DialogHandle (see page 1971)	Provides a mechanism for using non-VCL dialog boxes in an application.
	EnumAllWindowsOnActivateHint (see page 1972)	This is EnumAllWindowsOnActivateHint, a member of class TApplication.
R	ExeName (see page 1972)	Contains the file name of the application's executable file including path information.
	Handle (see page 1974)	Provides access to the window handle of the main form (window) of the application.
	HelpFile (see page 1974)	Specifies the name of the default file the application uses to display Help.
R	HelpSystem (see page 1976)	Provides access to the interface for working with the help system.
	Hint (see page 1976)	Specifies the text string that appears in the Help Hint box.
	HintColor (see page 1979)	Determines the color of the hint boxes for the Help Hints for the application.
	HintHidePause (see page 1980)	Specifies the time interval to wait before hiding the Help Hint (see page 1976) if the mouse has not moved from the control or menu item.
	HintPause (see page 1980)	Specifies the time interval that passes before the control's Help Hint (see page 1976) appears when the user places the mouse pointer on a control or menu item.

	HintShortCuts (see page 1980)	Specifies whether the application's pop-up hints include shortcut information.
	HintShortPause (see page 1981)	Specifies the time period to wait before bringing up a hint if another hint has already been shown.
	Icon (see page 1981)	Determines which icon represents the application on the Windows task bar.
R	MainForm (see page 1983)	Identifies which form in the application is the main window.
R	MainFormHandle (see page 1983)	Gives the Windows handle of the application main form.
	MainFormOnTaskBar (see page 1983)	Controls how Windows taskbar buttons are handled by VCL.
R	ModalLevel (see page 1985)	This is ModalLevel, a member of class TApplication.
	ModalPopupMode (see page 1985)	This is ModalPopupMode, a member of class TApplication.
	NonBiDiKeyboard (see page 1985)	Specifies the name of the keyboard map that should be used unless text reads from right to left.
R	PopupControlWnd (see page 1986)	This is PopupControlWnd, a member of class TApplication.
	ShowHint (see page 1986)	Determines whether Help Hints are enabled or disabled for the entire application.
	ShowMainForm (see page 1987)	Determines whether the application shows its main form on startup.
R	Terminated (see page 1988)	Reports whether the application is in the process of shutting down.
	Title (see page 1988)	Contains the text that appears below the icon representing the application when it is minimized.
	UpdateFormatSettings (see page 1989)	Specifies whether format settings are updated automatically when the user alters the system configuration.
	UpdateMetricSettings (see page 1989)	Specifies whether the settings used for the hint window font and the icon title font are updated.

7.1.1.2 TApplication.Create Constructor

Creates a new application.

Pascal

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

C++

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

Description

Do not call Create directly. Each GUI application automatically creates an instance of an application object.

For TApplication (see page 1924), the Create method initializes default values for some of its properties. It also prepares the application to run by setting up objects such as hint windows, the application icon, the main window handle, the instance handle, lists for managing child windows of the main window (form), and so on.

See Also

CreateForm (see page 1932)

CreateHandle (see page 1934)

Destroy (see page 1928)

Handle (see page 1974)

MainForm (see page 1983)

Run (see page 1959)

7.1.1.3 TApplication.Destroy Destructor

Destroys the application instance and all of its associated memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TApplication();
```

Description

Never call Application.Destroy or Application.Free. Instead, use the application's Terminate (see page 1962) method to end the application.

Destroy signals that the application has been deactivated before calling the inherited destructor. It then frees memory for hooked windows, the application instance, other object instances and lists that were created at application startup or that accumulated during execution

Since the Application (see page 2612) owns all forms (usually), it destroys these forms in its destructor.

See Also

Terminate (see page 1962)

Terminated (see page 1988)

OnDeactivate (see page 1992)

7.1.1.4 TApplication Methods

The methods of the TApplication class are listed here.

Protected Methods

	Name	Description
	Idle (see page 1945)	This is Idle, a member of class TApplication.
	IsDlgMsg (see page 1947)	This is IsDlgMsg, a member of class TApplication.
	IsHintMsg (see page 1947)	This is IsHintMsg, a member of class TApplication.
	IsKeyMsg (see page 1948)	This is IsKeyMsg, a member of class TApplication.
	IsMDIMsg (see page 1948)	This is IsMDIMsg, a member of class TApplication.
	IsPreProcessMessage (see page 1948)	This is IsPreProcessMessage, a member of class TApplication.
	IsShortCut (see page 1949)	This is IsShortCut, a member of class TApplication.

Public Methods

	Name	Description
	ActivateHint (see page 1930)	Displays a hint window for the control at a specified position.
	AddPopupForm (see page 1931)	This is AddPopupForm, a member of class TApplication.
	BringToFront (see page 1931)	Sets the last active window as the topmost window on the desktop above all other applications.
	CancelHint (see page 1931)	Cancels the display of a hint for a control.
	ControlDestroyed (see page 1931)	Notifies the application when a control is destroyed.
	CreateForm (see page 1932)	Creates a new form.
	CreateHandle (see page 1934)	Generates a main window for the application if it does not already have one.
	DoApplicationIdle (see page 1935)	Processes messages in the Message queue for certain events.
	ExecuteAction (see page 1935)	Generates an OnActionExecute (see page 1990) event.
	HandleException (see page 1935)	Provides default handling of exceptions for the application.
	HandleMessage (see page 1937)	Interrupts the execution of an application while Windows processes a message in the Windows message queue.
	HelpCommand (see page 1941)	Provides quick access to any of the Help commands in the WinHelp API (application programming interface).
	HelpContext (see page 1941)	Displays a specified help topic.

	HelpJump (see page 1942)	Displays a specified help topic.
	HelpKeyword (see page 1942)	Displays a specified help topic.
	HelpShowTableOfContents (see page 1943)	Shows the Table of Contents within the help system.
	HideHint (see page 1943)	Hides the current hint.
	HintMouseMessage (see page 1944)	Regulates the display of hint windows.
	HookMainWindow (see page 1944)	Enables a native Windows dialog box to receive messages sent to the application's main window.
	HookSynchronizeWakeup (see page 1944)	This is HookSynchronizeWakeup, a member of class TApplication.
	Initialize (see page 1945)	Provides an opportunity to initialize subsystems.
	IsRightToLeft (see page 1948)	Specifies whether the application adjusts its interface for Middle Eastern locales.
	MessageBox (see page 1949)	Displays a specified message to the user.
	Minimize (see page 1951)	Shrinks an application to the Windows task bar.
	ModalFinished (see page 1952)	Track closing of modal form.
	ModalStarted (see page 1952)	Track opening of modal form.
	NormalizeAllTopMosts (see page 1952)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	NormalizeTopMosts (see page 1953)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	ProcessMessages (see page 1954)	Interrupts the execution of an application so that it can process the message queue.
	RemovePopupForm (see page 1956)	This is RemovePopupForm, a member of class TApplication.
	Restore (see page 1957)	Restores a minimized application to its normal size.
	RestoreTopMosts (see page 1958)	Restores forms designated as fsStayOnTop to be topmost again.
	Run (see page 1959)	Executes the application.
	ShowException (see page 1961)	Displays a message box for exceptions that are not caught by application code.
	Terminate (see page 1962)	Ends application execution.
	UnhookMainWindow (see page 1964)	Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 1944) method.
	UnhookSynchronizeWakeup (see page 1964)	Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 1944).
	UpdateAction (see page 1965)	Generates an OnActionUpdate (see page 1991) event.
	UseRightToLeftAlignment (see page 1966)	Specifies whether the application object is in a right-to-left alignment mode.
	UseRightToLeftReading (see page 1966)	Specifies whether the application object is in a right-to-left text mode.
	UseRightToLeftScrollBar (see page 1966)	Specifies whether the vertical scroll bar appears on the left side controls.

7.1.1.4.1 TApplication.ActivateHint Method

Displays a hint window for the control at a specified position.

Pascal

```
procedure ActivateHint(CursorPos: TPoint);
```

C++

```
__fastcall ActivateHint(TPoint CursorPos);
```

Description

ActivateHint locates the control or menu item at the position specified by CursorPos, where CursorPos represents a screen coordinates in pixels. After locating the control, ActivateHint displays the control's Hint (see page 1976) in a hint window.

See Also

THintWindow

Hint (see page 1976)

OnHint (see page 1997)

Hint

7.1.1.4.2 TApplication.AddPopupForm Method

Pascal

```
function AddPopupForm(APopupForm: TCustomForm): Integer;
```

C++

```
__fastcall int AddPopupForm(TCustomForm APopupForm);
```

Description

This is AddPopupForm, a member of class TApplication.

7.1.1.4.3 TApplication.BringToFront Method

Sets the last active window as the topmost window on the desktop above all other applications.

Pascal

```
procedure BringToFront;
```

C++

```
__fastcall BringToFront();
```

Description

Use BringToFront to find the window owned by the MainForm (see page 1983) that was last active and bring it to the foreground. BringToFront also tests to see if the window is visible and enabled before making it the topmost window.

See Also

BringToFront

MainForm (see page 1983)

7.1.1.4.4 TApplication.CancelHint Method

Cancels the display of a hint for a control.

Pascal

```
procedure CancelHint;
```

C++

```
__fastcall CancelHint();
```

Description

Applications rarely, if ever, call CancelHint directly. CancelHint is used internally to cancel the display of hints at appropriate times, such as when a window is not valid or when the application closes. CancelHint calls a series of routines that clean up the display and timing issues of the hint window for a control.

See Also

HideHint (see page 1943)

Hint (see page 1976)

7.1.1.4.5 TApplication.ControlDestroyed Method

Notifies the application when a control is destroyed.

Pascal

```
procedure ControlDestroyed(Control: TControl);
```

C++

```
__fastcall ControlDestroyed(TControl * Control);
```

Description

ControlDestroyed is for internal use and is called automatically by TControl. Do not call it directly.

ControlDestroyed notifies the application that the control passed as the Control parameter has been destroyed, so that the properties which refer to controls, such as MainForm (see page 1983) and ActiveCustomForm, can be updated to specify a valid object or nil (Delphi) or NULL (C++).

See Also

Destroy (see page 1928)

MainForm (see page 1983)

ActiveCustomForm (see page 2452)

7.1.1.4.6 TApplication.CreateForm Method

Creates a new form.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference);
```

C++

```
__fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Call CreateForm to dynamically create a form at runtime. Developers do not need to add code for creating most forms, because typically one or more calls to CreateForm are added automatically to the project's source when using the form designer.

CreateForm creates a new form of the type specified by the FormClass parameter and assigns it to the variable given by the Reference parameter. The owner of the new form is the Application (see page 2612) object.

Note: By default, the form created by the first call to CreateForm in a project becomes the application's main form.

C++ Examples:

```
/*
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*_proj.cpp) file. To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.
You will need to set up your project with the following steps
before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place
               Form5 on the available forms list.
o          Change the code of your project (*_proj.cpp) file to
               look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
```

```
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.  You will need to set up
your project with the following steps before using the codeexample:    Add four additional
forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
```

```
with TForm5.Create(nil) do
try
  Application.MainFormOnTaskbar := True;
  ProgressBar1.Max := 100;
  Show;  // show a splash screen contain ProgressBar control
  Update; // force display of Form5
  Application.CreateForm(TForm1, Form1);
  ProgressBar1.StepBy(25);
  Label1.Caption :=  'Form1 loaded successfully.';
  Update; // force display of Form5
  Sleep(3000);
  Application.CreateForm(TForm2, Form2);
  ProgressBar1.StepBy(25);
  Label1.Caption :=  'Form2 loaded successfully.';
  Update; // force display of Form5
  Sleep(3000);
  Application.CreateForm(TForm3, Form3);
  ProgressBar1.StepBy(25);
  Label1.Caption :=  'Form3 loaded successfully.';
  Update; // force display of Form5
  Sleep(3000);
  Application.CreateForm(TForm4, Form4);
  ProgressBar1.StepBy(25);
  Label1.Caption :=  'Form4 loaded successfully.';
  Update; // force display of Form5
  Sleep(3000);
finally
  Free;
end;
Application.Run;
```

See Also

CreateHandle (see page 1934)

Handle (see page 1974)

MainForm (see page 1983)

Run (see page 1959)

7.1.1.4.7 TApplication.CreateHandle Method

Generates a main window for the application if it does not already have one.

Pascal

```
procedure CreateHandle;
```

C++

```
__fastcall CreateHandle();
```

Description

Do not call CreateHandle directly. In EXEs, the application's constructor calls CreateHandle. In DLLs it does not.

CreateHandle encapsulates much of what traditional Windows programs write in WinMain. For example, CreateHandle registers the Window Class, creates the application window and determines how the application window is shown.

See Also

Create (see page 1928)

CreateForm (see page 1932)

Handle (see page 1974)

MainForm (see page 1983)

Run (see page 1959)

7.1.1.4.8 TApplication.DoApplicationIdle Method

Processes messages in the Message queue for certain events.

Pascal

```
procedure DoApplicationIdle;
```

C++

```
__fastcall DoApplicationIdle();
```

Description

The DoApplicationIdle method forces a call to the protected method TApplication.Idle (see page 1945). Idle (see page 1945) triggers the event OnIdle (see page 1998), processes the events OnMouseEnter and OnMouseLeave, and updates the hints.

7.1.1.4.9 TApplication.ExecuteAction Method

Generates an OnActionExecute (see page 1990) event.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; reintroduce;
```

C++

```
__fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

When the user invokes an action, a series of calls ensues to respond to that action. First, the action list that contains the action generates an OnExecute event. If the action list does not handle the OnExecute event, then the action is routed to the Application (see page 2612) object's ExecuteAction method, which invokes the OnActionExecute (see page 1990) event handler. (The application's ExecuteAction method responds to all unhandled actions in the application.) If the OnActionExecute (see page 1990) event handler does not handle the action, then it is routed to the action's OnExecute event handler. If that does not handle the action, the active control's ExecuteAction method is called, followed (if necessary) by the active form's ExecuteAction method.

Do not call ExecuteAction. It is called automatically when the user invokes an action. The Action parameter specifies the action that was invoked. ExecuteAction checks whether the application knows how to perform its function, and if so, executes the action and returns true. Otherwise, ExecuteAction returns false, and the default processing for the action continues.

See Also

OnActionExecute (see page 1990)

UpdateAction (see page 1965)

Execute

HandlesTarget

ExecuteAction

7.1.1.4.10 TApplication.HandleException Method

Provides default handling of exceptions for the application.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

If an exception passes through all the try blocks in the application code, the application automatically calls HandleException, which displays a dialog box indicating an error occurred. Unless the exception object is EAbort, HandleException calls the OnException (see page 1993) event handler, if one exists. Otherwise, it calls ShowException (see page 1961) to display a message dialog box indicating an error occurred.

To assign other exception handling code for the application, use the OnException (see page 1993) event handler.

Delphi Examples:

```
{
AllocateHWnd, DeallocateHWnd example
The following code is from the implementation of TTimer. It
shows how the timer component's constructor creates a hidden
window to respond to Timer messages and how the destructor
frees that window.
}
{
TTimer implements a WndProc method that becomes the window
procedure for the hidden window.
}
procedure MyTTimer.WndProc(var Msg: TMessage);
begin
  with Msg do
    if Msg = WM_TIMER then { check for timer messages }
      try
        Timer; { this calls the OnTimer event handler }
      except
        Application.HandleException(Self);
      end
    else
 { Any other messages are passed to DefWindowProc, which
 tells Windows to handle the message.
 Note that the first parameter, FWindowHandle, is the handle
 of the window receiving this message. It is obtained from the
 call to AllocateHWnd in the constructor. }
      Result := DefWindowProc(FWindowHandle, Msg, wParam, lParam);
end;
{
The TTimer constructor uses AllocateHWnd to create the
window and save its handle.
}
constructor MyTTimer.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FEnabled := True;
  FInterval := 1000;
  FWindowHandle := AllocateHWnd(WndProc);
end;

{ The TTimer destructor calls DeallocateHWnd to free the hidden window. }
destructor MyTTimer.Destroy;
begin
  FEnabled := False;
//  UpdateTimer; done by TTimer Destroy
  DeallocateHWnd(FWindowHandle);
  inherited Destroy;
end;
```

See Also

EAbort

OnException (see page 1993)

ShowException (see page 1961)

7.1.1.4.11 TApplication.HandleMessage Method

Interrupts the execution of an application while Windows processes a message in the Windows message queue.

Pascal

```
procedure HandleMessage;
```

C++

```
__fastcall HandleMessage();
```

Description

HandleMessage interrupts the execution of the application so that Windows can process a single message from the Windows message queue before returning control to the application. If the message queue is empty, HandleMessage generates an OnIdle (see page 1998) event and starts the process of updating the actions in the application.

Note: If the application goes idle, HandleMessage may take a long time to return. Therefore, do not call HandleMessage when waiting for something message-based while priority actions are also being processed. Instead, call ProcessMessages (see page 1954) when processing more than just messages.

C++ Examples:

```
/*
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
*/
// global variables to show the order of events
int  XPos, YPos, Delta;

// This is a utility procedure to display messages
// add this at the beginning of the unit

void StatusMsg(TForm *Form, TCanvas *Canvas, char *Message, bool Bkg)
{
  if (!Bkg)
    Canvas->Font->Style = TFontStyles() << fsBold; // foreground messages are bold
  Canvas->TextOut(XPos, YPos, Message);
  if (!Bkg)
    Canvas->Font->Style = TFontStyles();
  // change Xpos and YPos to prepare for the next message
  YPos += Delta;
  if (YPos >= Form->ClientHeight - 10)
  {
    YPos = 10;
    XPos += 180;
  }
  if (XPos >= Form->ClientWidth - 100)
  {
    if (Canvas->Font->Color == clRed)
      Canvas->Font->Color = clBlack;
    else
```

```
      Canvas->Font->Color = clRed;
    XPos = 10;
  }
}

/*
This is the OnCreate event handler for Form1.  It
initializes global values and sets the application's OnIdle
event handler
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Button1->Caption = "Do not yield";
  Button2->Caption  = "Handle Message";
  Application->OnIdle = MyIdleHandler;
  XPos = 10;
  YPos = 10;
  if (Canvas->Font->Height > 0)
    Delta = Canvas->Font->Height + 1;
  else
    Delta = 1 - Canvas->Font->Height;
}

/*
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some background
processing for the application.  This one simply writes a
message to let you know when it's there.
*/
void __fastcall TForm1::MyIdleHandler(TObject *Sender, bool &Done)
{
  StatusMsg(Form1, Canvas, "This represents a background process.", true);
}

/*
This is the OnClick event handler for Button1.  It
simulates a lengthy process
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button1Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
      Y = random(J);
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button1Click handler is done", false);
}

/*
This is the OnClick event handler for button2.  It
simulates a lengthy process that allows other messages to be
processed.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button2Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
```

```
    {
      Y = random(J);
    }
    // Yield to OnIdle or other messages
    Application->HandleMessage();
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button2Click handler is done", false);
}
```

Delphi Examples:

```
{
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
}
var
{ global variables to show the order of events. }
  XPos, YPos, Delta: integer;

{ This is a utility procedure to display messages }
{ add this at the beginning of the implementation section }
procedure StatusMsg(
  MyForm : TForm1; Canvas : TCanvas; Message : string; Bkg : Boolean);
begin
  if not bkg then
    Canvas.Font.Style := [fsBold]; {foreground messages are bold }
  Canvas.TextOut(XPos, YPos, Message);
  if not bkg then
    Canvas.Font.Style := [];
  { change Xpos and YPos to prepare for the next message }
  Ypos := Ypos + Delta;
  if YPos >= MyForm.ClientHeight - 10 then
  begin
    YPos := 10;
    Xpos := Xpos + 180;
  end;
  if (Xpos >= MyForm.ClientWidth - 100) then
  begin
    if (Canvas.Font.Color = clRed) then
      Canvas.Font.Color := clBlack
    else
      Canvas.Font.Color := clRed;
    Xpos := 10;
  end;
end;

{This is the Form's OnCreate event handler.  }
{ It initializes global variables and sets the  the OnIdle
event handler }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Do not yield';
  Button2.Caption := 'Handle Message';
  Application.OnIdle:= MyIdleHandler;
  XPos := 10;
  YPos := 10;
  Delta := Abs(Canvas.Font.Height) + 1;
 end;

{
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some
background processing for the application.  This one simply
```

```
  writes a message to let you know when it's there.
  }
  procedure TForm1.MyIdleHandler(Sender: TObject; var Done: Boolean);
  begin
    StatusMsg(
      Form1, Canvas, 'This represents a background process.', True);
  end;

  {
  Set this method as the OnClick event handler of Button1.  It
  performs a calculation without yielding to other messages or
  idle time.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, J, X, Y: Word;
  begin
    StatusMsg(
      Form1, Canvas, 'The synchronous handler is starting', False);
    I := 0;
    J := 0;
    while I < 10 do
    begin
      Randomize;
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    StatusMsg(
      Form1, Canvas, 'The synchronous handler is done', False);
  end;

  {
  Set this method as the OnClick event handler of Button2.  It
  performs a calculation but calls HandleMessage to allow idle
  time or asynchronous message processing.
  }
  procedure TForm1.Button2Click(Sender: TObject);
  var
    I, J, X, Y: Word;
  begin
   StatusMsg(
      Form1, Canvas, 'The asynchronous handler is starting', False);
    I := 0;
    J := 0;
    while I < 10 do
    begin
      Randomize;
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
      { yield to OnIdle or other messages }
      Application.HandleMessage;
    end;
    StatusMsg(
      Form1, Canvas, 'The asynchronous handler is done', False);
  end;
```

See Also

OnIdle (see page 1998)

ProcessMessages (see page 1954)

UpdateAction (see page 1965)

7.1.1.4.12 TApplication.HelpCommand Method

Provides quick access to any of the Help commands in the WinHelp API (application programming interface).

Pascal

```
function HelpCommand(Command: Integer; Data: Longint): Boolean;
```

C++

```
__fastcall Boolean HelpCommand(int Command, Longint Data);
```

Description

Use HelpCommand to send a command to WinHelp. Before sending the help command to the Windows help engine, HelpCommand generates an OnHelp (see page 1995) event on the active form or on TApplication (see page 1924). Finally, the command is forwarded to WinHelp if there is no OnHelp (see page 1995) event handler or if the OnHelp (see page 1995) event handler indicates that WinHelp should be called.

For information about Windows help commands and the data passed to them, see the WinHelp topic in the Windows Help system.

See Also

HelpContext (see page 1941)

HelpFile (see page 1974)

HelpJump (see page 1942)

OnHelp (see page 1995)

OnHelp (see page 2217)

7.1.1.4.13 TApplication.HelpContext Method

Displays a specified help topic.

Pascal

```
function HelpContext(Context: THelpContext): Boolean;
```

C++

```
__fastcall Boolean HelpContext(THelpContext Context);
```

Description

Use HelpContext to bring up the help topic specified by the Context parameter from the file specified in the CurrentHelpFile (see page 1970) property. HelpContext generates an OnHelp (see page 1995) event (either on the active form or on the application object itself). If there is no OnHelp (see page 1995) event handler, or if the event handler indicates that WinHelp should be called, then HelpContext invokes the Windows help engine, passing it the context ID.

For example, if the Context value is 714, the HelpContext method displays the screen with the context help ID of 714 in the help file specified by CurrentHelpFile (see page 1970).

HelpContext returns false if CurrentHelpFile (see page 1970) is an empty string, meaning the application has no help file assigned. In all other cases, HelpContext returns true.

See Also

HelpCommand (see page 1941)

HelpContext

HelpFile (see page 1974)

HelpJump (see page 1942)

OnHelp (see page 1995)

CurrentHelpFile (see page 1970)

HelpKeyword (see page 1942)

Help and Hint Properties

7.1.1.4.14 TApplication.HelpJump Method

Displays a specified help topic.

Pascal

```
function HelpJump(const JumpID: string): Boolean;
```

C++

```
__fastcall Boolean HelpJump(const UnicodeString JumpID);
```

Description

Call HelpJump to bring up a topic identified by its symbolic name. HelpJump displays the topic identified by the JumpID parameter from the file specified in the Current HelpFile (see page 1974) property. HelpJump generates an OnHelp (see page 1995) event (either on the active form or on the application object itself). If there is no OnHelp (see page 1995) event handler, or if the event handler indicates that WinHelp should be called, then HelpJump invokes the Windows help engine, passing it the topic name.

HelpJump returns false if CurrentHelpFile (see page 1970) is an empty string, meaning the application has no help file assigned. In all other cases, HelpJump returns true.

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpFile (see page 1974)

OnHelp (see page 1995)

7.1.1.4.15 TApplication.HelpKeyword Method

Displays a specified help topic.

Pascal

```
function HelpKeyword(const Keyword: string): Boolean;
```

C++

```
__fastcall Boolean HelpKeyword(const UnicodeString Keyword);
```

Description

Use HelpKeyword to bring up the help topic specified by the Keyword parameter from the file specified in the CurrentHelpFile (see page 1970) property. HelpKeyword invokes the current help system, passing it the string specified by Keyword. If the current system is WinHelp, this displays the topic that has an Alink of Keyword.

HelpKeyword returns false if the current help system does not support the keyword lookups.

See Also

HelpCommand (see page 1941)

HelpKeyword

HelpFile (see page 1974)

HelpJump (see page 1942)

OnHelp (see page 1995)

CurrentHelpFile (see page 1970)

7.1.1.4.16 TApplication.HelpShowTableOfContents Method

Shows the Table of Contents within the help system.

Pascal

```
function HelpShowTableOfContents: Boolean;
```

C++

```
__fastcall Boolean HelpShowTableOfContents();
```

Description

Use HelpShowTableOfContents to bring up the table of contents for the help system specified in the CurrentHelpFile (see page 1970) property. This method is preferred to calling "Application.HelpSystem.ShowTableOfContents", which cannot be used as the first call into the help system.

7.1.1.4.17 TApplication.HideHint Method

Hides the current hint.

Pascal

```
procedure HideHint;
```

C++

```
__fastcall HideHint();
```

Description

Applications rarely, if ever, call HideHint directly. HideHint is used internally to hide the hint window at the appropriate times according to predetermined conditions and timer values. For example, CancelHint (see page 1931) calls HideHint.

See Also

CancelHint (see page 1931)

HintPause (see page 1980)

HintShortPause (see page 1981)

7.1.1.4.18 TApplication.HintMouseMessage Method

Regulates the display of hint windows.

Pascal

```
procedure HintMouseMessage(Control: TControl; var Message: TMessage);
```

C++

```
__fastcall HintMouseMessage(TControl * Control, TMessage Message);
```

Description

This method is called automatically when the mouse passes over a control with a Hint (see page 1976). Applications rarely, if ever, call HintMouseMessage directly. HintMouseMessage is used internally to manage the display of the hint windows.

See Also

CancelHint (see page 1931)

HintPause (see page 1980)

HintShortPause (see page 1981)

7.1.1.4.19 TApplication.HookMainWindow Method

Enables a native Windows dialog box to receive messages sent to the application's main window.

Pascal

```
procedure HookMainWindow(Hook: TWindowHook);
```

C++

```
__fastcall HookMainWindow(TWindowHook Hook);
```

Description

Use HookMainWindow to ensure that a native Windows dialog box behaves correctly as a child of the application, not as a stand-alone window. For example, switching among applications with Alt+Tab treats the application as a single task after calling HookMainWindow, rather than treating the native Windows dialog box as a separate task.

When the window identified by the Handle (see page 1974) property receives relevant dialog messages, it passes them to the dialog procedure passed as the Hook parameter.

There is no problem with leaving a dialog box hooked into the main window, even for extended periods. However, should the dialog box close, call the UnhookMainWindow (see page 1964) method to release the hook.

See Also

Handle (see page 1974)

UnhookMainWindow (see page 1964)

7.1.1.4.20 TApplication.HookSynchronizeWakeup Method

Pascal

```
procedure HookSynchronizeWakeup;
```

C++

```
__fastcall HookSynchronizeWakeup();
```

Description

This is HookSynchronizeWakeup, a member of class TApplication.

7.1.1.4.21 TApplication.Idle Method

Pascal

```
procedure Idle(const Msg: TMsg);
```

C++

```
__fastcall Idle(const TMsg Msg);
```

Description

This is Idle, a member of class TApplication.

7.1.1.4.22 TApplication.Initialize Method

Provides an opportunity to initialize subsystems.

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

Initialize is the first method called by the project source file. It calls the InitProc procedure pointer. By default, the call to Initialize for the application does nothing because the default InitProc pointer is nil (Delphi) or NULL (C++) .

To use Initialize, the InitProc pointer must be predefined. This can be accomplished in one of two ways:

In Delphi, you can include a unit that assigns a procedure to InitProc in its initialization section, such as the ComObj unit. You can make this assignment in the initialization section of any of your units.

In both Delphi and C++, you can create a custom initialization procedure that assigns a value to the InitProc pointer, and add a call to this procedure to the project source prior to the call to Initialize. (In Delphi, you can add it to the initialization section of the unit in which it is declared. In C++, you can use the pragma startup directive in that unit.)

Warning: Only one instance of InitProc can be defined in an application. If more than one unit assigns a value to InitProc, only the last assignment will work. You can, however, call the previous value of InitProc from an initialization procedure, so that all initialization procedures are executed.

For projects that do not assign a value to InitProc, the call to Initialize can be safely deleted from the project source.

Note: Although Initialize is the first method called in the main project source code, it is not the first code that is executed in a GUI application. For example, in Delphi, the application first executes the initialization section of all the units used by the application.

C++ Examples:

```
/*
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*_proj.cpp) file. To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.
```

```
  You will need to set up your project with the following steps
  before using the code example:
  o          Add four additional forms to a default project.
  o          Place a TProgressBar on Form5
  o          Take the Project|Options|Forms menu option and place
                 Form5 on the available forms list.
  o          Change the code of your project (*_proj.cpp) file to
                 look like the example.
  */
  #include <vcl\vcl.h>
  #include "Unit5.h"
  #pragma hdrstop
  //---------------------------------------------------------------------------
  USEFORM("TAppCreateForm.cpp", Form1);
  USERES("Project1.res");
  USEFORM("Unit2.cpp", Form2);
  USEFORM("Unit3.cpp", Form3);
  USEFORM("Unit4.cpp", Form4);
  USEFORM("Unit5.cpp", Form5);
  //---------------------------------------------------------------------------
  WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
  {
    try
    {
      Application->Initialize();
      SetApplicationMainFormOnTaskBar(Application, true);
      Application->CreateForm(__classid(TForm5), &Form5);
      Form5->ProgressBar1->Max = 100;
      Form5->Show();   // show a splash screen contain ProgressBar control
      Form5->Update(); // force display of Form5
      Application->CreateForm(__classid(TForm1), &Form1);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form1 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
      Application->CreateForm(__classid(TForm2), &Form2);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form2 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
      Application->CreateForm(__classid(TForm3), &Form3);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form3 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
      Application->CreateForm(__classid(TForm4), &Form4);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form4 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
  //  delete Form5;
      Application->Run();
    }
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
    return 0;
  }
```

Delphi Examples:

```
  {
  This example shows progress of loading forms as an
  application starts up. The code example is placed in the
  project source (*.dpr) file.  To see project source, right
  click on the executable file in the Project Manager and
```

```
  select the View Source menu item.  You will need to set up
  your project with the following steps before using the codeexample:    Add four additional
  forms to a default project.
      Place a TProgressBar on Form5
      Take the Project|Options|Forms menu option and place
            Form5 on the available forms list.
      Change the code of your project (*.dpr) file to look
            like the example.
  }
  begin
    Application.Initialize;
    with TForm5.Create(nil) do
    try
      Application.MainFormOnTaskbar := True;
      ProgressBar1.Max := 100;
      Show;  // show a splash screen contain ProgressBar control
      Update; // force display of Form5
      Application.CreateForm(TForm1, Form1);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form1 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm2, Form2);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form2 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm3, Form3);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form3 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm4, Form4);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form4 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
    finally
      Free;
    end;
    Application.Run;
```

See Also

InitProc

7.1.1.4.23 TApplication.IsDlgMsg Method

Pascal

```
function IsDlgMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsDlgMsg(TMsg Msg);
```

Description

This is IsDlgMsg, a member of class TApplication.

7.1.1.4.24 TApplication.IsHintMsg Method

Pascal

```
function IsHintMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsHintMsg(TMsg Msg);
```

Description

This is IsHintMsg, a member of class TApplication.

7.1.1.4.25 TApplication.IsKeyMsg Method

Pascal

```
function IsKeyMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsKeyMsg(TMsg Msg);
```

Description

This is IsKeyMsg, a member of class TApplication.

7.1.1.4.26 TApplication.IsMDIMsg Method

Pascal

```
function IsMDIMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsMDIMsg(TMsg Msg);
```

Description

This is IsMDIMsg, a member of class TApplication.

7.1.1.4.27 TApplication.IsPreProcessMessage Method

Pascal

```
function IsPreProcessMessage(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsPreProcessMessage(TMsg Msg);
```

Description

This is IsPreProcessMessage, a member of class TApplication.

7.1.1.4.28 TApplication.IsRightToLeft Method

Specifies whether the application adjusts its interface for Middle Eastern locales.

Pascal

```
function IsRightToLeft: Boolean;
```

C++

```
__fastcall Boolean IsRightToLeft();
```

Description

Use IsRightToLeft to determine whether the application makes any adjustments of control alignment, text direction, or scroll bar placement. IsRightToLeft returns true if the application's BiDiMode (see page 1970) indicates that any of these should be

reversed and the application is running on a system with a Middle Eastern locale.

See Also

BiDiMode (see page 1970)

userighttoleftreading (see page 1966)

7.1.1.4.29 TApplication.IsShortCut Method

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

This is IsShortCut, a member of class TApplication.

7.1.1.4.30 TApplication.MessageBox Method

Displays a specified message to the user.

Pascal

```
function MessageBox(const Text: PChar; const Caption: PChar; Flags: Longint = MB_OK): Integer;
```

C++

```
__fastcall int MessageBox(const const char * Text, const const char * Caption, Longint Flags =
MB_OK);
```

Description

Use MessageBox to display a generic dialog box a message and one or more buttons. Caption is the caption of the dialog box and is optional.

MessageBox is an encapsulation of the Windows API MessageBox function. TApplication (see page 1924)'s encapsulation of MessageBox automatically supplies the missing window handle parameter needed for the Windows API function.

The value of the Text parameter is the message, which can be longer than 255 characters if necessary. Long messages are automatically wrapped in the message box.

The value of the Caption parameter is the caption that appears in the title bar of the dialog box. Captions can be longer than 255 characters, but don't wrap. A long caption results in a wide message box.

The Flags parameter specifies what buttons appear on the message box and the behavior (possible return values). The following table lists the possible values. These values can be combined to obtain the desired effect.

Value	Meaning
MB_ABORTRETRYIGNORE	The message box contains three push buttons: Abort, Retry, and Ignore.
MB_OK	The message box contains one push button: OK. This is the default.
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL	The message box contains two push buttons: Retry and Cancel.
MB_YESNO	The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.

MessageBox returns 0 if there isn't enough memory to create the message box. Otherwise it returns one of the following values:

ValueNumeric valueMeaning

Value	Numeric value	Meaning
IDOK	1	The user chose the OK button.
IDCANCEL	2	The user chose the Cancel button.
IDABORT	3	The user chose the Abort button.
IDRETRY	4	The user chose the Retry button.
IDIGNORE	5	The user chose the Ignore button.
IDYES	6	The user chose the Yes button.
IDNO	7	The user chose the No button.

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

MessageDlgPosHelp

ShowMessage

MessageDlg

7.1.1.4.31 TApplication.Minimize Method

Shrinks an application to the Windows task bar.

Pascal

```
procedure Minimize;
```

C++

```
__fastcall Minimize();
```

Description

Call Minimize to minimize the application. When the user minimizes the application, the Minimize method is automatically called.

Note: Don't confuse the Minimize method, which minimizes an entire application, with minimizing a form or window. To minimize, maximize, or restore a window or form, change the value of its WindowState property.

C++ Examples:

```
/*
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
*/
void __fastcall TForm1::AppDeactivate(TObject *Sender)
{
  Application->Minimize();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnDeactivate = AppDeactivate;
}
```

Delphi Examples:

```
{
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnDeactivate := AppDeactivate;
end;
procedure TForm1.AppDeactivate(Sender: TObject);
begin
  Application.Minimize;
end;
```

See Also

Icon (see page 1981)

OnMinimize (see page 2004)

Restore (see page 1957)

WindowState (see page 2195)

7.1.1.4.32 TApplication.ModalFinished Method

Track closing of modal form.

Pascal

```
procedure ModalFinished;
```

C++

```
__fastcall ModalFinished();
```

Description

Call ModalFinished to indicate that a modal form is being closed. ModalFinished is called by the ShowModal method of TCustomForm (see page 2103).

ModalFinished calls the OnModalEnd (see page 2006) event handler, if defined.

See Also

OnModalEnd (see page 2006)

ModalStarted (see page 1952)

ShowModal (see page 2139)

7.1.1.4.33 TApplication.ModalStarted Method

Track opening of modal form.

Pascal

```
procedure ModalStarted;
```

C++

```
__fastcall ModalStarted();
```

Description

Call ModalStarted to indicate that a modal form is being opened. ModalStarted is called by the ShowModal method of TCustomForm (see page 2103).

ModalStarted calls the OnModalStart event handler, if defined.

See Also

OnModalStart

ModalFinished (see page 1952)

ShowModal (see page 2139)

7.1.1.4.34 TApplication.NormalizeAllTopMosts Method

Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.

Pascal

```
procedure NormalizeAllTopMosts;
```

C++

```
__fastcall NormalizeAllTopMosts();
```

Description

Use NormalizeAllTopMosts to allow other windows appear on top of a topmost form. Normally, topmost forms remain on top of other windows, and so may obscure other windows.

NormalizeAllTopMosts behaves like the NormalizeTopMosts (see page 1953) method, except that it also includes the application's Main Window.

See Also

FormStyle (see page 2162)

NormalizeTopMosts (see page 1953)

RestoreTopMosts (see page 1958)

MainForm (see page 1983)

7.1.1.4.35 TApplication.NormalizeTopMosts Method

Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.

Pascal

```
procedure NormalizeTopMosts;
```

C++

```
__fastcall NormalizeTopMosts();
```

Description

Use NormalizeTopMosts to allow a message box or dialog box that is displayed using the Windows API functions (such as MessageBox (see page 1949) and MessageDlg) directly, appear on top of a topmost form. Otherwise the topmost form remains on top, and may obscure the message box.

Calling NormalizeTopMosts is not necessary when using functions to display message boxes.

To return the forms designated as fsStayOnTop to be topmost again, call RestoreTopMosts (see page 1958).

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
```

```
to float to the top.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

FormStyle (see page 2162)

RestoreTopMosts (see page 1958)

NormalizeAllTopMosts (see page 1952)

MessageBox (see page 1949)

MessageDlg

7.1.1.4.36 TApplication.ProcessMessages Method

Interrupts the execution of an application so that it can process the message queue.

Pascal

```
procedure ProcessMessages;
```

C++

```
__fastcall ProcessMessages();
```

Description

Call ProcessMessages to permit the application to process messages that are currently in the message queue. ProcessMessages cycles the Windows message loop until it is empty, and then returns control to the application.

Note: Neglecting message processing affects only the application calling ProcessMessages, not other applications. In lengthy operations, calling ProcessMessages periodically allows the application to respond to paint and other messages.

Note: ProcessMessages does not allow the application to go idle, whereas HandleMessage (see page 1937) does.

C++ Examples:

```
/*
This example uses two buttons that are long enough to
accommodate lengthy captions on a form. When the user clicks
the button with the caption Ignore Messages, the code begins
to generate a long series of random numbers. If the user
tries to resize the form while the handler is running,
nothing happens until the handler is finished. When the user
clicks the button with the caption Process Messages, more
random numbers are generated, but the application can still
respond to a series of events, such as resizing the form
and printing text.
Note: How quickly these event handlers run depends on the
microprocessor of your computer. A message appears on the
form informing you when the handler has finished executing.
*/
```

```
int magicnumber = 20;

void __fastcall TForm1::FormCreate(TObject* Sender)
{
  Button1->Caption = "Ignore Messages";
  Button2->Caption = "Handle Message";
}

void __fastcall TForm1::Button1Click(TObject* Sender)
{
  int x, y;
  Canvas->TextOut(10, 10, "The Button1Click handler has started");
  Application->ProcessMessages(); // Just to get out the
  Sleep(100);
  for (int i = 0; i < magicnumber; i++)
 {
    Randomize();
    for (int j = 0; j < magicnumber; j++)
    {
      Sleep(10);
      y = random(j);
    }
    x = random(i);
  }
  Canvas->TextOut(10, 10, "The Button1Click handler is finished   ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int x, y;
  Canvas->TextOut(10, 10, "The Button2Click handler has started");
  for (int i = 0; i < magicnumber; i++)
  {
    Randomize();
    for (int j = 0; j < magicnumber; j++)
    {
      y = random(j);
      Sleep(10);
      Application->ProcessMessages();
    }
    x = random(i);
  }
  Canvas->TextOut(10, 10, "The Button2Click handler is finished   ");
}
```

Delphi Examples:

```
{
This example uses two buttons that are long enough to
accommodate lengthy captions on a form. When the user clicks
the button with the caption Ignore Messages, the code begins
to generate a long series of random numbers. If the user
tries to resize the form while the handler is running,
nothing happens until the handler is finished. When the user
clicks the button with the caption Process Messages, more
random numbers are generated, but the application can still
respond to a series of events, such as resizing the form
and printing text.
Note: How quickly these event handlers run depends on the
microprocessor of your computer. A message appears on the
form informing you when the handler has finished executing.
}
const magicnumber = 500;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Ignore Messages';
```

```
    Button2.Caption := 'Process Messages';
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, J: Integer;
    X, Y: Word;
  begin
    I := 0;
    J := 0;
    Canvas.TextOut(10, 10, 'The Button1Click handler has started');
    Application.ProcessMessages; // Just to get the message out.
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Sleep(10);
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    Canvas.TextOut(10, 10, 'The Button1Click handler is finished   ');
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  var
    I, J: Integer;
    X, Y: Word;
  begin
    I := 0;
    J := 0;
    Canvas.TextOut(10, 10, 'The Button2Click handler has started');
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Y := Random(J);
        Inc(J);
        Sleep(10);
        Application.ProcessMessages;
      end;
      X := Random(I);
      Inc(I);
    end;
    Canvas.TextOut(10, 10, 'The Button2Click handler is finished   ');
  end;
```

See Also

HandleMessage (see page 1937)

OnMessage (see page 2003)

7.1.1.4.37 TApplication.RemovePopupForm Method

Pascal

```
procedure RemovePopupForm(APopupForm: TCustomForm);
```

C++

```
__fastcall RemovePopupForm(TCustomForm APopupForm);
```

Description

This is RemovePopupForm, a member of class TApplication.

7.1.1.4.38 TApplication.Restore Method

Restores a minimized application to its normal size.

Pascal

```
procedure Restore;
```

C++

```
__fastcall Restore();
```

Description

Use Restore to restore the application to its previous size before it was minimized. When the user restores the application to normal size, Restore is automatically called.

Note: Don't confuse the Restore method, which restores the entire application, with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of its WindowState property.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
```

```
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

WindowState (see page 2195)

7.1.1.4.39 TApplication.RestoreTopMosts Method

Restores forms designated as fsStayOnTop to be topmost again.

Pascal

```
procedure RestoreTopMosts;
```

C++

```
__fastcall RestoreTopMosts();
```

Description

Use RestoreTopMosts to return forms that were originally designated as topmost forms (FormStyle is fsStayOnTop), then temporarily changed to be non-topmost forms with the NormalizeTopMosts (see page 1953) or NormalizeAllTopMosts (see page 1952) method, to a topmost position.

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

NormalizeTopMosts (see page 1953)

NormalizeAllTopMosts (see page 1952)

7.1.1.4.40 TApplication.Run Method

Executes the application.

Pascal

```
procedure Run;
```

C++

```
__fastcall Run();
```

Description

Do not call Run. When creating a new project, The IDE automatically creates a main program block in the project file that calls the Run method. When the application is executed, the application's Run method is called.

Run contains the application's main message loop. The call to Run does not return until the application's message loop terminates.

C++ Examples:

```
/*
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*_proj.cpp) file. To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.
You will need to set up your project with the following steps
before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place
               Form5 on the available forms list.
o          Change the code of your project (*_proj.cpp) file to
               look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
```

```
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.  You will need to set up
your project with the following steps before using the codeexample:    Add four additional
forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
```

```
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

Create (see page 1928)

CreateForm (see page 1932)

CreateHandle (see page 1934)

MainForm (see page 1983)

Terminate (see page 1962)

7.1.1.4.41 TApplication.ShowException Method

Displays a message box for exceptions that are not caught by application code.

Pascal

```
procedure ShowException(E: Exception);
```

C++

```
__fastcall ShowException(Exception E);
```

Description

The HandleException (see page 1935) method calls ShowException by default if no handler is specified for the OnException (see page 1993) event. If an exception handler filters exceptions, ShowException is called for those exceptions that are not filtered out.

To specify how exceptions are handled for an application, write a handler for the OnException (see page 1993) event.

Note: Calling ShowException is rarely necessary in a component-based application, since the default exception handler calls ShowException automatically.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;

procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

HandleException (see page 1935)

OnException (see page 1993)

7.1.1.4.42 TApplication.Terminate Method

Ends application execution.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to end the application programmatically. By calling Terminate rather than freeing the application object, you allow the application to shut down in an orderly fashion.

Terminate calls the Windows API PostQuitMessage function to perform an orderly shutdown of the application. Terminate is not immediate.

Terminate is called automatically on a WM_QUIT (see page 5204) message and when the main form closes.

C++ Examples:

```
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
```

```
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;

procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

Terminated (see page 1988)

7.1.1.4.43 TApplication.UnhookMainWindow Method

Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 1944) method.

Pascal

```
procedure UnhookMainWindow(Hook: TWindowHook);
```

C++

```
__fastcall UnhookMainWindow(TWindowHook Hook);
```

Description

Use UnhookMainWindow to release the hooked window. Specify the dialog procedure as the value of the Hook parameter.

The TWindowHook (see page 2611) type is the parameter type used for UnhookMainWindow. It is a method-pointer type used for the dialog-procedures of non-VCL dialog boxes. The dialog procedure is similar to a window procedure for a window, in that it processes messages for the dialog box, but its syntax is somewhat different.

For more information about Windows hooks see the SetWindowsEx and UnhookWindowsEx functions in the Windows Help File.

See Also

HookMainWindow (see page 1944)

7.1.1.4.44 TApplication.UnhookSynchronizeWakeup Method

Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 1944).

Pascal

```
procedure UnhookSynchronizeWakeup;
```

C++

```
__fastcall UnhookSynchronizeWakeup();
```

Description

Applications rarely, if ever, call UnhookSynchronizeWakeup directly. UnhookSynchronizeWakeup is called from the TApplication (see page 1924) destructor. It resets the WakeMainThread variable to nil (Delphi) or NULL (C++) if that variable was set by the HookSynchronizeWakeup (see page 1944) method.

Warning: You should not modify WakeMainThread for GUI applications because a working handler has been assigned. Console applications may assign a handler to WakeMainThread if the application needs to be notified of a thread synchronization call. If your console application assigns a handler, this is an optimization that only accelerates deliver of notification that a synchronization request is available. You must use CheckSynchronize to perforrm the synchronization itself.

See Also

hooksynchronizewakeup (see page 1944)

WakeMainThread

Synchronize

7.1.1.4.45 TApplication.UpdateAction Method

Generates an OnActionUpdate (see page 1991) event.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; reintroduce;
```

C++

```
__fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

When the application is idle, it makes a series of calls to update the properties (such as whether it is enabled, checked, and so on) of every action that is linked to a visible control or menu item. First, that action list that contains the action generates an OnUpdate event. If the action list does not handle the OnUpdate event, then the action is routed to the Application (see page 2612) object's UpdateAction method, which invokes the OnActionUpdate (see page 1991) event handler. (The application's UpdateAction method responds to all unhandled actions in the application.) If the OnActionUpdate (see page 1991) event handler does not update the action, then it is routed to the action's OnUpdate event handler. If that does not update the action, the active control's UpdateAction method is called, followed (if necessary) by the active form's UpdateAction method.

Do not call UpdateAction. It is called automatically when the application is idle. The Action parameter specifies the action to be updated. UpdateAction returns true if the action's properties are updated and require no further processing. Otherwise, UpdateAction returns false, and the default action processing continues.

See Also

OnActionUpdate (see page 1991)

ExecuteAction (see page 1935)

UpdateTarget

HandlesTarget

InitiateAction

7.1.1.4.46 TApplication.UseRightToLeftAlignment Method

Specifies whether the application object is in a right-to-left alignment mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Use UseRightToLeftAlignment to determine whether the application's controls alignment is reversed so that they are aligned from right to left. UseRightToLeftAlignment returns true for Middle Eastern locales if the BiDiMode (see page 1970) property is bdRightToLeft. Otherwise, it returns false.

See Also

BiDiMode

7.1.1.4.47 TApplication.UseRightToLeftReading Method

Specifies whether the application object is in a right-to-left text mode.

Pascal

```
function UseRightToLeftReading: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftReading();
```

Description

Call UseRightToLeftReading to determine which style and text drawing flags (used in ExTextOut or DrawText) to set for the application's controls.

UseRightToLeftReading returns true for for Middle Eastern locales if the BiDiMode (see page 1970) property is not bdLeftToRight. Otherwise, it returns false.

See Also

BiDiMode

7.1.1.4.48 TApplication.UseRightToLeftScrollBar Method

Specifies whether the vertical scroll bar appears on the left side controls.

Pascal

```
function UseRightToLeftScrollBar: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftScrollBar();
```

Description

Call UseRightToLeftScrollBar to determine whether the vertical scroll bars appear on the left sides of the application's controls. When UseRightToLeftScrollBar returns true, the vertical scroll bars appear on the left sides of the controls. When UseRightToLeftScrollBar returns false, the vertical scroll bars appear on the right.

UseRightToLeftScrollBar returns true for Middle Eastern locales if BiDiMode (see page 1970) is bdRightToLeft or

bdRightToLeftNoAlign.

See Also

BiDiMode

7.1.1.5 TApplication Properties

The properties of the TApplication class are listed here.

Public Properties

	Name	Description
	ActionUpdateDelay (see page 1968)	Delays the call to DoActionIdle.
R	Active (see page 1968)	Specifies whether the application is active and has focus.
R	ActiveFormHandle (see page 1968)	Gives the Windows handle of the currently active form.
	AllowTesting (see page 1969)	Represents information for the IDE.
	AutoDragDocking (see page 1969)	Indicates whether dragged windows are automatically docked.
	BiDiKeyboard (see page 1969)	Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.
	BiDiMode (see page 1970)	Indicates the layout of the application when running under Middle Eastern versions of Windows.
R	CurrentHelpFile (see page 1970)	Indicates the current help file.
	DefaultFont (see page 1971)	This is DefaultFont, a member of class TApplication.
	DialogHandle (see page 1971)	Provides a mechanism for using non-VCL dialog boxes in an application.
	EnumAllWindowsOnActivateHint (see page 1972)	This is EnumAllWindowsOnActivateHint, a member of class TApplication.
R	ExeName (see page 1972)	Contains the file name of the application's executable file including path information.
	Handle (see page 1974)	Provides access to the window handle of the main form (window) of the application.
	HelpFile (see page 1974)	Specifies the name of the default file the application uses to display Help.
R	HelpSystem (see page 1976)	Provides access to the interface for working with the help system.
	Hint (see page 1976)	Specifies the text string that appears in the Help Hint box.
	HintColor (see page 1979)	Determines the color of the hint boxes for the Help Hints for the application.
	HintHidePause (see page 1980)	Specifies the time interval to wait before hiding the Help Hint (see page 1976) if the mouse has not moved from the control or menu item.
	HintPause (see page 1980)	Specifies the time interval that passes before the control's Help Hint (see page 1976) appears when the user places the mouse pointer on a control or menu item.
	HintShortCuts (see page 1980)	Specifies whether the application's pop-up hints include shortcut information.
	HintShortPause (see page 1981)	Specifies the time period to wait before bringing up a hint if another hint has already been shown.
	Icon (see page 1981)	Determines which icon represents the application on the Windows task bar.
R	MainForm (see page 1983)	Identifies which form in the application is the main window.
R	MainFormHandle (see page 1983)	Gives the Windows handle of the application main form.
	MainFormOnTaskBar (see page 1983)	Controls how Windows taskbar buttons are handled by VCL.
R	ModalLevel (see page 1985)	This is ModalLevel, a member of class TApplication.
	ModalPopupMode (see page 1985)	This is ModalPopupMode, a member of class TApplication.
	NonBiDiKeyboard (see page 1985)	Specifies the name of the keyboard map that should be used unless text reads from right to left.
R	PopupControlWnd (see page 1986)	This is PopupControlWnd, a member of class TApplication.
	ShowHint (see page 1986)	Determines whether Help Hints are enabled or disabled for the entire application.
	ShowMainForm (see page 1987)	Determines whether the application shows its main form on startup.
R	Terminated (see page 1988)	Reports whether the application is in the process of shutting down.
	Title (see page 1988)	Contains the text that appears below the icon representing the application when it is minimized.
	UpdateFormatSettings (see page 1989)	Specifies whether format settings are updated automatically when the user alters the system configuration.

	UpdateMetricSettings (see page 1989)	Specifies whether the settings used for the hint window font and the icon title font are updated.

7.1.1.5.1 TApplication.ActionUpdateDelay Property

Delays the call to DoActionIdle.

Pascal

```
property ActionUpdateDelay: Integer;
```

C++

```
__property int ActionUpdateDelay;
```

Description

Use ActionUpdateDelay to delay the call to DoActionIdle. The following example demonstrates the usefulness of this property:

1. Create (see page 1928) an MDI project with a main form, a MDI Child and a TAction class.
2. Link the TAction OnUpdate event to a method.
3. Run (see page 1959) the program and maximize the MDI child.

If you open Windows Task Manager, you can see that CPU usage for the MDI application is nearly 100%. However, if you set Application.ActionUpdateDelay to a non-zero value, the CPU usage for the application is significantly lower than 100%.

See Also

Building a VCL Forms MDI Application Using a Wizard

7.1.1.5.2 TApplication.Active Property

Specifies whether the application is active and has focus.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Active is true while the application is active and false if it is not. Active is set to true in the constructor.

An application is active if the form or application has focus. An application becomes inactive when a window from a different application is about to become activated. When the application closes, Active is set to false in the destructor.

Note: Active is read-only.

See Also

OnActivate (see page 1992)

OnDeactivate (see page 1992)

7.1.1.5.3 TApplication.ActiveFormHandle Property

Gives the Windows handle of the currently active form.

Pascal

```
property ActiveFormHandle: HWND;
```

C++

```
__property HWND ActiveFormHandle;
```

Description

ActiveFormHandle gives the Windows handle of the currently active form.

See Also

MainFormHandle (see page 1983)

7.1.1.5.4 TApplication.AllowTesting Property

Represents information for the IDE.

Pascal

```
property AllowTesting: Boolean;
```

C++

```
__property Boolean AllowTesting;
```

Description

Do not use AllowTesting; it is for internal use only.

7.1.1.5.5 TApplication.AutoDragDocking Property

Indicates whether dragged windows are automatically docked.

Pascal

```
property AutoDragDocking: Boolean;
```

C++

```
__property Boolean AutoDragDocking;
```

Description

Set AutoDragDocking to enable drag-and-dock behavior when windows are dragged. When AutoDragDocking is true, dragged windows are automatically docked when the mouse is released over a docking site. When AutoDragDocking is false, dragged windows are only docked when the Ctrl key is pressed.

7.1.1.5.6 TApplication.BiDiKeyboard Property

Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.

Pascal

```
property BiDiKeyboard: string;
```

C++

```
__property UnicodeString BiDiKeyboard;
```

Description

This name is a string composed of the hexadecimal value of the language identifier (low word) and a device identifier (high word). For example, Saudi Arabian Arabic has a language identifier of 0x0401, so its primary layout is named "00000401".

This keyboard layout is only used when the application runs in a middle eastern locale and the value of BiDiMode (see page 1970) indicates that text should read from right to left.

See Also

BiDiMode (see page 1970)

NonBiDiKeyboard (see page 1985)

7.1.1.5.7 TApplication.BiDiMode Property

Indicates the layout of the application when running under Middle Eastern versions of Windows.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to specify the bi-directional mode for the application. The bi-directional mode controls the direction in which text appears (left-to-right or right-to-left), the placement of vertical scroll bars, and the default alignment of text when the application runs in Middle Eastern locales.

The application's BiDiMode is used in the ShowMessage procedure and the MessageBox (see page 1949) method. It also serves as the ParentBiDiMode for forms and for popup menus that are not activated by a control.

See Also

ParentBiDiMode

ShowMessage

MessageBox (see page 1949)

7.1.1.5.8 TApplication.CurrentHelpFile Property

Indicates the current help file.

Pascal

```
property CurrentHelpFile: string;
```

C++

```
__property UnicodeString CurrentHelpFile;
```

Description

Read CurrentHelpFile to determine the name of the help file that is currently used by the application when it executes a help command (via the HelpCommand (see page 1941) method) or brings up context sensitive help (via the HelpContext (see page 1941) or HelpJump (see page 1942) method). If CurrentHelpFile is an empty string, the application ignores attempts to display Help.

If the active form has an associated help file, this is the value of CurrentHelpFile. Otherwise, CurrentHelpFile reflects the value of the application's HelpFile (see page 1974) property.

See Also

PropertyName

ActiveCustomForm (see page 2452)

HelpCommand (see page 1941)

HelpJump (see page 1942)

HelpSystem (see page 1976)

HelpContext (see page 1941)

7.1.1.5.9 TApplication.DefaultFont Property

Pascal

```
property DefaultFont: TFont;
```

C++

```
__property TFont DefaultFont;
```

Description

This is DefaultFont, a member of class TApplication.

7.1.1.5.10 TApplication.DialogHandle Property

Provides a mechanism for using non-VCL dialog boxes in an application.

Pascal

```
property DialogHandle: HWnd;
```

C++

```
__property HWnd DialogHandle;
```

Description

Use DialogHandle when displaying a modeless dialog box that was created using the CreateDialog API function. Assigning the handle of the modeless dialog box to DialogHandle allows the dialog to see messages from the application's message loop.

Assign the handle of the modeless dialog box to DialogHandle when it receives an activation message (WM_NCACTIVATE (see page 5189)) and set DialogHandle to zero when the dialog box receives a deactivation message.

Delphi Examples:

```
{
Here is replacement of the window procedure for Delphi's
encapsulation of the Windows search-and-replace common dialog
box. Click on Button2 to activate an open dialog and send
a WM_NCACTIVATE to FindReplaceWndProc.
}
function FindReplaceWndProc(Wnd: HWND; Msg, WParam, LParam: Longint): Longint; stdcall;

  function CallDefWndProc: Longint;
  begin
    Result := CallWindowProc(Pointer(GetProp(Wnd,
      MakeIntAtom(WndProcPtrAtom))), Wnd, Msg, WParam, LParam);
  end;

begin
  case Msg of
    WM_DESTROY:
      if Application.DialogHandle = Wnd then Application.DialogHandle := 0;
    WM_NCACTIVATE:
      if WParam <> 0 then
      begin
        if Application.DialogHandle = 0 then Application.DialogHandle := Wnd;
      end else
      begin
```

```
          if Application.DialogHandle = Wnd then Application.DialogHandle := 0;
        end;
      WM_NCDESTROY:
        begin
          Result := CallDefWndProc;
          RemoveProp(Wnd, MakeIntAtom(WndProcPtrAtom));
          Exit;
        end;
    end;
    Result := CallDefWndProc;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    ReplaceDialog1.Execute;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    OpenDialog1.Execute;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    AtomText: array[0..31] of Char;
  begin
    WndProcPtrAtom := GlobalAddAtom(StrFmt(AtomText,
      'WndProcPtr%.8X%.8X', [HInstance, GetCurrentThreadID]));
    SetWindowLong(Application.Handle, GWL_WNDPROC, Longint(@FindReplaceWndProc));;
  end;
```

See Also

Handle (see page 1974)

7.1.1.5.11 TApplication.EnumAllWindowsOnActivateHint Property

Pascal

```
property EnumAllWindowsOnActivateHint: Boolean;
```

C++

```
__property Boolean EnumAllWindowsOnActivateHint;
```

Description

This is EnumAllWindowsOnActivateHint, a member of class TApplication.

7.1.1.5.12 TApplication.ExeName Property

Contains the file name of the application's executable file including path information.

Pascal

```
property ExeName: string;
```

C++

```
__property UnicodeString ExeName;
```

Description

Use ExeName to obtain the name of the executable file for the application. ExeName is the fully-qualified name, including the path to the application's executable.

The name of the application is the root name of the project with an .EXE extension. By default, this name is PROJECT1.EXE. To change the ExeName, save the project by the desired new root name and rebuild the application. ExeName will reflect the change to the project file name.

ExeName is a read-only property.

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  temp1 := Edit4.Text;
```

```
    temp2 := ExtractFileName(Edit1.Text);
    NewFileName := temp1 + temp2;
    OldFileName :=
      Edit3.Text + ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
```

See Also

ParamStr

Title (see page 1988)

7.1.1.5.13 TApplication.Handle Property

Provides access to the window handle of the main form (window) of the application.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Use Handle when calling Windows API functions that require a parent window handle. For example, a DLL that displays its own top-level pop-up windows needs a parent window to display its windows in the application. Using the Handle property makes such windows part of the application, so that they are minimized, restored, enabled, and disabled with the application.

Note: When writing a DLL that uses VCL forms, assign the window handle of the host EXE's main window to the DLL's Application.HandleApplication->Handle property. This makes the DLL's form part of the host application. Never assign to the Handle property in an EXE.

See Also

DialogHandle (see page 1971)

MainForm (see page 1983)

7.1.1.5.14 TApplication.HelpFile Property

Specifies the name of the default file the application uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property UnicodeString HelpFile;
```

Description

Use HelpFile to specify a default help file for applications that use the Help system. Either assign a file name to the HelpFile property at runtime, or specify a Help File on the Application (see page 2612) page of the Project|Options dialog box at design time.

The Help system specified by HelpSystem (see page 1976) displays help topics from the file specified by CurrentHelpFile (see page 1970). HelpFile supplies the value of CurrentHelpFile (see page 1970) unless there is a help file associated with the active form.

By default, HelpFile is a null string, and the application's Help method ignores attempts to display Help unless the active form has an associated help file.

C++ Examples:

```
/*
The following code changes the Help file depending on the
active form, before the help viewer is called. AppHelp is
assigned to the OnHelp event handler of Application in the
OnCreate event of Form1.
*/
bool __fastcall TForm1::AppHelp(Word Command, int Data, bool &CallHelp)
{
  Application->HelpFile = Screen->ActiveForm->Name + ".hlp";
  CallHelp = true;
  return true;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHelp = AppHelp;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  const DATANOTFOUND = 714;
  Application->HelpContext(DATANOTFOUND);
}
```

Delphi Examples:

```
{
The following code changes the Help file depending on the
active form, before the help viewer is called. AppHelp is
assigned to the OnHelp event handler of Application in the
OnCreate event of Form1.
}
function TForm1.AppHelp(
  Command: Word; Data: Longint; var CallHelp: Boolean) : Boolean;
var helpstring : String;
begin
  helpstring := Screen.ActiveForm.Name + '.hlp';
  MessageDlg(
    'Using ' + helpstring + ' as the help file.',
    mtInformation, [mbOK], 0);
  Application.HelpFile := helpstring;
  CallHelp := True;
end;

procedure TForm1.Button1Click(Sender: TObject);
const DATANOTFOUND = 714;
begin
```

```
  Application.HelpContext(DATANOTFOUND);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnHelp := AppHelp;
end;
```

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpJump (see page 1942)

CurrentHelpFile (see page 1970)

HelpSystem (see page 1976)

7.1.1.5.15 TApplication.HelpSystem Property

Provides access to the interface for working with the help system.

Pascal

```
property HelpSystem: IHelpSystem;
```

C++

```
__property IHelpSystem HelpSystem;
```

Description

TApplication (see page 1924) uses IHelpSystem (see page 3231) to interact with a Help system manager. This interface includes methods for displaying help topics based on a keyword or topic ID.

Applications can use the HelpSystem property to interact directly with the Help system manager, or they can use the application's help-oriented methods such as HelpCommand (see page 1941) and HelpJump (see page 1942).

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpJump (see page 1942)

CurrentHelpFile (see page 1970)

HelpFile (see page 1974)

ShowTableOfContents (see page 3233)

7.1.1.5.16 TApplication.Hint Property

Specifies the text string that appears in the Help Hint box.

Pascal

```
property Hint: string;
```

C++

```
__property UnicodeString Hint;
```

Description

TApplication (see page 1924)'s Hint property is assigned the Hint value of the control or menu item that the mouse is currently moving over. It can also be assigned a string value that informs the user of some action, error, or other type of information. Therefore, use Hint either to:

Transfer hints from controls to another area of display, such as a status bar, using the OnHint (see page 1997) event handler. This reads the Hint property.

Indicate the status of the application while it is processing some action. This sets the Hint property.

Help Hints appear when the OnHint (see page 1997) event occurs. Therefore, if TApplication (see page 1924)'s Hint property is assigned to the status bar's caption, for example, the caption of the status bar control displays the current string value of the Hint property during an OnHint (see page 1997) event.

There are two parts to the Hint string: short and long. Short hints are used by pop-up tool tips. Long hints are used by the status bar, separated by the | character. Use the global functions GetShortHint and GetLongHint, from the Controls unit to extract the long and short hints from a hint string.

Note: When setting Hint to a message informing the user of an event occurring in the application, remember that, by default, the Hint string is reset to a control's Hint when the mouse cursor moves over a control.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
/*
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
*/
```

```
/*
Here is the code to add to the unit file.  It provides the
implementation of the DisplayHint method and assigns it to
the Application in the form's OnCreate event handler.
Note:   It is not actually necessary to write an OnHint event
handler to display hints on the status bar. This can be
accomplished more simply by setting the status bar's AutoHint
property.
*/
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
}

// Here is the form's OnCreate event handler.
// It assigns the application's OnHint event handler at
// runtime because the Application is not available in the
// Object Inspector
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHint = DisplayHint;
}
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
{
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    StatusBar1: TStatusBar;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
```

```
    public
      procedure DisplayHint(Sender: TObject);
    end;
  var
    Form1: TForm1;
  implementation
  {$R *.dfm}

  { Here is the implementation of the OnHint event handler }
  { It displays the application's current hint in the status bar }
  procedure TForm1.DisplayHint(Sender: TObject);
  begin
    StatusBar1.SimpleText := GetLongHint(Application.Hint);
  end;
  { Here is the form's OnCreate event handler. }
  { It assign's the application's OnHint event handler at runtime }
  { because the Application is not available in the Object Inspector }
  { at design time }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnHint := DisplayHint;
  end;
```

See Also

GetLongHint

GetShortHint

HintColor (see page 1979)

OnHint (see page 1997)

ShowHint (see page 1986)

THintWindow

7.1.1.5.17 TApplication.HintColor Property

Determines the color of the hint boxes for the Help Hints for the application.

Pascal

```
property HintColor: TColor;
```

C++

```
__property TColor HintColor;
```

Description

Use HintColor to specify the hint box color. A default color value of clInfoBk (see page 2909) is set for the HintColor property in the constructor when the application is created. To change the HintColor, assign it a new TColor (see page 2878) value at runtime.

Tip: To further customize the hint window, create a custom descendant of THintWindow and assign it to the global HintWindowClass (see page 2615) variable.

See Also

Hint (see page 1976)

THintWindow

HintWindowClass (see page 2615)

7.1.1.5.18 TApplication.HintHidePause Property

Specifies the time interval to wait before hiding the Help Hint (see page 1976) if the mouse has not moved from the control or menu item.

Pascal

```
property HintHidePause: Integer;
```

C++

```
__property int HintHidePause;
```

Description

Use HintHidePause to specify a wait time in milliseconds, which is different from the default value of 2500 ms or 2 1/2 seconds that is set in the constructor. The Help Hint (see page 1976) for the control or menu item is specified in the Hint (see page 1976) property.

Note: The default is a predefined constant value that is several times longer than HintPause (see page 1980).

See Also

Hint (see page 1976)

HintPause (see page 1980)

HintShortPause (see page 1981)

7.1.1.5.19 TApplication.HintPause Property

Specifies the time interval that passes before the control's Help Hint (see page 1976) appears when the user places the mouse pointer on a control or menu item.

Pascal

```
property HintPause: Integer;
```

C++

```
__property int HintPause;
```

Description

Use HintPause to change the default pause time of 500 ms or 1/2 second that is set in the constructor. When assigning a value to HintPause, specify the interval in milliseconds. The Help Hint (see page 1976) displayed at the end of the pause is specified in the Hint (see page 1976) property.

See Also

HintHidePause (see page 1980)

Hint (see page 1976)

HintShortPause (see page 1981)

7.1.1.5.20 TApplication.HintShortCuts Property

Specifies whether the application's pop-up hints include shortcut information.

Pascal

```
property HintShortCuts: Boolean;
```

C++

```
__property Boolean HintShortCuts;
```

Description

Use HintShortCuts to specify whether the application's pop-up hints include shortcut information. If HintShortCuts is true, the shortcut information is displayed. For example, if HintShortCuts is true, the hint for the Cut command might be "Cut (Ctrl+X)" instead of "Cut".

7.1.1.5.21 TApplication.HintShortPause Property

Specifies the time period to wait before bringing up a hint if another hint has already been shown.

Pascal

```
property HintShortPause: Integer;
```

C++

```
__property int HintShortPause;
```

Description

Use HintShortPause to reduce the flicker when moving the mouse quickly over a set of buttons that all have help hints on. Specify a value in milliseconds that is different from the default value of 50 ms that is set in the constructor.

See Also

HintHidePause (see page 1980)

Hint (see page 1976)

HintPause (see page 1980)

7.1.1.5.22 TApplication.Icon Property

Determines which icon represents the application on the Windows task bar.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use Icon to assign a specific icon to represent the application. Set the Icon property by either:

Assigning it a TIcon (see page 2768) at runtime.

Loading an icon on the Application (see page 2612) page of the Project|Options dialog box at design time.

Icon represents the application when it is minimized and appears either on the Windows desktop or task bar.

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

See Also

Minimize (see page 1951)

OnMinimize (see page 2004)

Restore (see page 1957)

Title (see page 1988)

7.1.1.5.23 TApplication.MainForm Property

Identifies which form in the application is the main window.

Pascal

```
property MainForm: TForm;
```

C++

```
__property TForm * MainForm;
```

Description

Use MainForm to determine the form that acts as the application's main window. The main form is the first form created in the main body of the application. When the main form closes, the application terminates.

When a new project is created, Form1 automatically becomes the value of the MainForm property. To assign a different form to the MainForm property, use the Forms (see page 1921) page of the Project|Options dialog box at design time. MainForm cannot be modified at runtime; it is read-only.

See Also

Create (see page 1928)

CreateForm (see page 1932)

Handle (see page 1974)

Terminate (see page 1962)

7.1.1.5.24 TApplication.MainFormHandle Property

Gives the Windows handle of the application main form.

Pascal

```
property MainFormHandle: HWND;
```

C++

```
__property HWND MainFormHandle;
```

Description

MainFormHandle gives the Windows handle of the application main form.

See Also

ActiveFormHandle (see page 1968)

7.1.1.5.25 TApplication.MainFormOnTaskBar Property

Controls how Windows taskbar buttons are handled by VCL.

Pascal

```
property MainFormOnTaskBar: Boolean;
```

C++

```
__property Boolean MainFormOnTaskBar;
```

Description

The MainFormOnTaskBar property controls how Windows taskbar buttons are handled by VCL.

If the property is true, a taskbar button represents the application's main form and displays its caption. If false, a taskbar button represents the application's (hidden) main window and bears the application's Title (see page 1988).

MainFormOnTaskBar must be true to use Windows Vista Aero effects. These include live taskbar thumbnails, Dynamic Windows, Windows Flip, and Windows Flip 3D.

If a change to default behavior is required, MainFormOnTaskBar should be set in the .dpr file after Application.Initialize (see page 1945) and before main form creation. The MainFormOnTaskBar setting is intended to persist for the life of the application; later runtime changes of this property could result in unexpected behavior.

MainFormOnTaskBar defaults to `True` for applications created in Delphi 2007 and later products and `False` for earlier products.

The property can be applied to older applications. Note that it affects the Z-order of your MainForm (see page 1983) in case your application depends on this.

To update existing VCL applications, add the following line to the project's .dpr file after `Application.Initialize (see page 1945);`:

```
Application.MainFormOnTaskbar := True;
```

This line is automatically added to new projects.

For further information, see http://www.microsoft.com/windows/products/windowsvista/features/experiences/aero.mspx.

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.  You will need to set up
your project with the following steps before using the codeexample:    Add four additional
forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
```

```
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

TApplication (see page 1924)

7.1.1.5.26 TApplication.ModalLevel Property

Pascal

```
property ModalLevel: Integer;
```

C++

```
__property int ModalLevel;
```

Description

This is ModalLevel, a member of class TApplication.

7.1.1.5.27 TApplication.ModalPopupMode Property

Pascal

```
property ModalPopupMode: TPopupMode;
```

C++

```
__property TPopupMode ModalPopupMode;
```

Description

This is ModalPopupMode, a member of class TApplication.

7.1.1.5.28 TApplication.NonBiDiKeyboard Property

Specifies the name of the keyboard map that should be used unless text reads from right to left.

Pascal

```
property NonBiDiKeyboard: string;
```

C++

```
__property UnicodeString NonBiDiKeyboard;
```

Description

Use NonBiDiKeyboard to specify the keyboard map that should be used. This name is a string composed of the hexadecimal value of the language identifier (low word) and a device identifier (high word). For example, U.S. English has a language

identifier of 0x0409, so the primary U.S. English layout is named "00000409". Variants of U.S. English layout (such as the Dvorak layout) are named "00010409", "00020409", and so on.

If the application is running in a middle eastern locale and the value of BiDiMode (see page 1970) indicates that text should read from right to left, the application uses BiDiKeyboard (see page 1969) instead.

See Also

BiDiMode (see page 1970)

BiDiKeyboard (see page 1969)

7.1.1.5.29 TApplication.PopupControlWnd Property

Pascal

```
property PopupControlWnd: HWND;
```

C++

```
__property HWND PopupControlWnd;
```

Description

This is PopupControlWnd, a member of class TApplication.

7.1.1.5.30 TApplication.ShowHint Property

Determines whether Help Hints are enabled or disabled for the entire application.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

Use ShowHints to choose when to display Help Hints. If ShowHint is true, Help Hints are enabled; if ShowHint is false, Help Hints are disabled. The default value is true. Help Hints are specified in the Hint (see page 1976) property.

Setting ShowHint for the application to false disables all Help Hints, regardless of the value of the ShowHint properties for individual controls.

Tip: You can customize the hint window by creating a custom descendant of THintWindow and assigning it to the global HintWindowClass (see page 2615) variable.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
```

```
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 1976)

OnHint (see page 1997)

ParentShowHint

ShowHint

THintWindow

7.1.1.5.31 TApplication.ShowMainForm Property

Determines whether the application shows its main form on startup.

Pascal

```
property ShowMainForm: Boolean;
```

C++

```
__property Boolean ShowMainForm;
```

Description

Use ShowMainForm to control whether and when the application shows its main form.

The default value of true is set in the constructor. When ShowMainForm is true, the main window shows automatically when the application starts. The MainForm (see page 1983) property specifies which form is the main form.

To hide the main form at startup, set ShowMainForm to false in the main project file before the call to the Run (see page 1959) method. Make sure that the Visible property of the form is also false.

See Also

Create (see page 1928)

MainForm (see page 1983)

Run (see page 1959)

Visible

Hiding the Main Form

7.1.1.5.32 TApplication.Terminated Property

Reports whether the application is in the process of shutting down.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Terminated is used when calling the ProcessMessages (see page 1954) method so that the application doesn't perform unnecessary actions after the shut-down process has been triggered. ProcessMessages (see page 1954) sets Terminated to true when it receives a WM_QUIT (see page 5204) message.

An application usually receives a WM_QUIT (see page 5204) message because the main window of the application has closed, or because the Terminate (see page 1962) method has been called.

Note: For applications using calculation-intensive loops, call ProcessMessages (see page 1954) periodically, and also check Terminated to determine whether to abort the calculation and allow the application to terminate.

See Also

ProcessMessages (see page 1954)

Terminate (see page 1962)

7.1.1.5.33 TApplication.Title Property

Contains the text that appears below the icon representing the application when it is minimized.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Use the Title property to modify the name of the text string appearing below the icon of the application. The default value is taken from the DLL or EXE file name at runtime.

Set the Title at runtime, or enter the value of the Title property on the Application (see page 2612) page of the Project | Options dialog box.

C++ Examples:

```
/*
This code displays the name of your project in an edit box:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->Text = Application->Title;
}
```

Delphi Examples:

```
{
This code displays the name of your project in an edit box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := Application.Title;
end;
```

See Also

ExeName (see page 1972)

Icon (see page 1981)

7.1.1.5.34 TApplication.UpdateFormatSettings Property

Specifies whether format settings are updated automatically when the user alters the system configuration.

Pascal

```
property UpdateFormatSettings: Boolean;
```

C++

```
__property Boolean UpdateFormatSettings;
```

Description

Use UpdateFormatSettings to control automatic updating of format settings. The default of true is set in the constructor. UpdateFormatSettings is checked when the application receives a WM_WININICHANGE (see page 5219) message.

Using the default format settings is recommended. These settings are initialized to the Windows local settings.

See Also

variables

7.1.1.5.35 TApplication.UpdateMetricSettings Property

Specifies whether the settings used for the hint window font and the icon title font are updated.

Pascal

```
property UpdateMetricSettings: Boolean;
```

C++

```
__property Boolean UpdateMetricSettings;
```

Description

Use UpdateMetricSettings to specify whether or not the hint window font and the icon title font on the desk top should be updated to reflect the system settings when those settings are changed.

The initial value of true is set in the constructor.

7.1.1.6 TApplication Events

The events of the TApplication class are listed here.

Public Events

	Name	Description
	OnActionExecute (see page 1990)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1991)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1992)	Occurs when an application becomes active.
	OnDeactivate (see page 1992)	Occurs when an application becomes inactive.
	OnException (see page 1993)	Occurs when an unhandled exception occurs in the application.
	OnGetActiveFormHandle (see page 1995)	This is OnGetActiveFormHandle, a member of class TApplication.
	OnGetMainFormHandle (see page 1995)	This is OnGetMainFormHandle, a member of class TApplication.
	OnHelp (see page 1995)	Occurs when the application receives a request for help.
	OnHint (see page 1997)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 1976).
	OnIdle (see page 1998)	Occurs when an application becomes idle.
	OnMessage (see page 2003)	Occurs when the application receives a Windows message.
	OnMinimize (see page 2004)	Occurs when an application is minimized.
	OnModalBegin (see page 2006)	Occurs when a modal form is opened.
	OnModalEnd (see page 2006)	Occurs when a modal form is closed.
	OnRestore (see page 2006)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 2007)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 2007)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 2008)	Occurs when the application is about to display the hint window for a Help Hint (see page 1976).

7.1.1.6.1 TApplication.OnActionExecute Event

Occurs when an action's Execute method is called and its action list has not already handled it.

Pascal

```
property OnActionExecute: TActionEvent;
```

C++

```
__property TActionEvent OnActionExecute;
```

Description

Use the OnExecute event handler to respond when a user invokes actions whose action lists do not have OnExecute event handlers.

If the action list that contains the action does not handle it in an OnExecute event handler, then the action is routed to the Application (see page 2612) object's ExecuteAction (see page 1935) method, which invokes the OnActionExecute event handler. (The application's ExecuteAction (see page 1935) method applies to all actions in the application.)

The Handled parameter of the event handler returns false by default. If the handler handles the event, it should change Handled to true, thereby preventing further attempts to handle the action. When the event handler exits with Handled set to false, the action's OnExecute event occurs. If the action remains unhandled after that, the active control's ExecuteAction (see page 1935) method is called to allow the action to execute with an identified target. Finally, the active form's ExecuteAction (see page 1935) method is called if all other handlers do not handle the action.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnActionUpdate (see page 1991)

ExecuteAction (see page 1935)

OnExecute

OnExecute

ExecuteAction

OnActionExecute

What Happens When an Action Fires

7.1.1.6.2 TApplication.OnActionUpdate Event

Occurs when an action's Update method is called and its action list has not already handled it.

Pascal

```
property OnActionUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnActionUpdate;
```

Description

Use the OnActionUpdate event handler to update the properties of an action when the application is idle if the action's action list does not handle it in an OnUpdate event handler.

If the action list that contains the action does not update it in an OnUpdate event handler, the action is routed to the Application (see page 2612) object's UpdateAction (see page 1965) method, which invokes the OnActionUpdate event handler. (The application's UpdateAction (see page 1965) method applies to all actions in the application.)

The Handled parameter of the event handler returns false by default. If the handler updates the event, it should change Handled to true, thereby ending the processing of the action. When the event handler exits with Handled set to false, the action's OnUpdate event occurs. If the action is not updated after that, the active control's UpdateAction (see page 1965) method is called to allow the target to update the action. Finally, the active form's UpdateAction (see page 1965) method is called if all other handlers do not handle the action.

Note: OnActionUpdate occurs after the application's OnIdle (see page 1998) event.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnActionExecute (see page 1990)

UpdateAction (see page 1965)

OnUpdate

OnUpdate

UpdateAction

OnIdle (see page 1998)

OnActionUpdate

7.1.1.6.3 TApplication.OnActivate Event

Occurs when an application becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform special processing when the application becomes active.

An application becomes active when it is initially run or when focus moves from another Windows application back to any window of the application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

Active (see page 1968)

OnDeactivate (see page 1992)

OnActivate

7.1.1.6.4 TApplication.OnDeactivate Event

Occurs when an application becomes inactive.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any special processing that should occur immediately before the application is deactivated. The OnDeactivate event occurs when the user switches from the application to another Windows application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
*/
void __fastcall TForm1::AppDeactivate(TObject *Sender)
{
  Application->Minimize();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnDeactivate = AppDeactivate;
}
```

Delphi Examples:

```
{
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnDeactivate := AppDeactivate;
end;
procedure TForm1.AppDeactivate(Sender: TObject);
begin
  Application.Minimize;
end;
```

See Also

Active (see page 1968)

OnActivate (see page 1992)

OnDeactivate

7.1.1.6.5 TApplication.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TExceptionEvent;
```

C++

```
__property TExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException (see page 1935) method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run (see page 1959) method do not generate OnException events.

If an exception passes through the tryblocks in the application code, the application automatically calls the HandleException (see page 1935) method. Unless the exception object is EAbort, HandleException (see page 1935) calls the OnException

handler, if one exists. Otherwise, it calls ShowException (see page 1961) to display a message box indicating an error occurred.

TExceptionEvent (see page 2599) is the type of the OnException event. It points to a method that handles exceptions in the application. The Sender parameter is the object that raised the exception, and E is the exception object.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;

procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

HandleException (see page 1935)

ShowException (see page 1961)

OnException

7.1.1.6.6 TApplication.OnGetActiveFormHandle Event

Pascal

```
property OnGetActiveFormHandle: TGetHandleEvent;
```

C++

```
__property TGetHandleEvent OnGetActiveFormHandle;
```

Description

This is OnGetActiveFormHandle, a member of class TApplication.

7.1.1.6.7 TApplication.OnGetMainFormHandle Event

Pascal

```
property OnGetMainFormHandle: TGetHandleEvent;
```

C++

```
__property TGetHandleEvent OnGetMainFormHandle;
```

Description

This is OnGetMainFormHandle, a member of class TApplication.

7.1.1.6.8 TApplication.OnHelp Event

Occurs when the application receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext (see page 1941) and the HelpJump (see page 1942) methods automatically trigger the OnHelp event.

Set CallHelp to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. All application help methods go through OnHelp. The application calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

The event handler returns true if it succeeds, false if it fails.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code changes the Help file depending on the
active form, before the help viewer is called. AppHelp is
```

```
  assigned to the OnHelp event handler of Application in the
  OnCreate event of Form1.
  */
  bool __fastcall TForm1::AppHelp(Word Command, int Data, bool &CallHelp)
  {
    Application->HelpFile = Screen->ActiveForm->Name + ".hlp";
    CallHelp = true;
    return true;
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Application->OnHelp = AppHelp;
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    const DATANOTFOUND = 714;
    Application->HelpContext(DATANOTFOUND);
  }
```

Delphi Examples:

```
  {
  The following code changes the Help file depending on the
  active form, before the help viewer is called. AppHelp is
  assigned to the OnHelp event handler of Application in the
  OnCreate event of Form1.
  }
  function TForm1.AppHelp(
    Command: Word; Data: Longint; var CallHelp: Boolean) : Boolean;
  var helpstring : String;
  begin
    helpstring := Screen.ActiveForm.Name + '.hlp';
    MessageDlg(
      'Using ' + helpstring + ' as the help file.',
      mtInformation, [mbOK], 0);
    Application.HelpFile := helpstring;
    CallHelp := True;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  const DATANOTFOUND = 714;
  begin
    Application.HelpContext(DATANOTFOUND);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnHelp := AppHelp;
  end;
```

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpFile (see page 1974)

HelpJump (see page 1942)

OnHelp

InvokeHelp

7.1.1.6.9 TApplication.OnHint Event

Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 1976).

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write an OnHint event handler to perform special processing when the mouse pauses over a control or menu item whose Hint (see page 1976) property is not an empty string.

A common use of the OnHint event is to display the value of a control or menu item's Hint (see page 1976) property in another control. Note that it is not necessary to use an OnHint event handler to display hints on a status bar (TStatusBar): TStatusBar will automatically display hints if you set its AutoHint property to true.

The Hint (see page 1976) property of a control can specify both a short Help Hint (see page 1976) and a (usually) longer hint that appears elsewhere because of code in an OnHint event handler.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

Tip: Component writers can respond to the automatically-fired THintAction action in a component's ExecuteAction (see page 1935) method rather than relying on an OnHint event handler.

C++ Examples:

```
/*
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
*/

/*
Here is the code to add to the unit file.  It provides the
implementation of the DisplayHint method and assigns it to
the Application in the form's OnCreate event handler.
Note:   It is not actually necessary to write an OnHint event
handler to display hints on the status bar. This can be
accomplished more simply by setting the status bar's AutoHint
property.
*/
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
}

// Here is the form's OnCreate event handler.
// It assigns the application's OnHint event handler at
// runtime because the Application is not available in the
// Object Inspector
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  Application->OnHint = DisplayHint;
}
```

Delphi Examples:

```
{
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    StatusBar1: TStatusBar;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure DisplayHint(Sender: TObject);
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

{ Here is the implementation of the OnHint event handler }
{ It displays the application's current hint in the status bar }
procedure TForm1.DisplayHint(Sender: TObject);
begin
  StatusBar1.SimpleText := GetLongHint(Application.Hint);
end;
{ Here is the form's OnCreate event handler. }
{ It assign's the application's OnHint event handler at runtime }
{ because the Application is not available in the Object Inspector }
{ at design time }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnHint := DisplayHint;
end;
```

See Also

GetLongHint

GetShortHint

Hint (see page 1976)

ActivateHint (see page 1930)

OnHint

AutoHint

ExecuteAction

THintAction

7.1.1.6.10 TApplication.OnIdle Event

Occurs when an application becomes idle.

Pascal

```
property OnIdle: TIdleEvent;
```

C++

```
__property TIdleEvent OnIdle;
```

Description

Write an OnIdle event handler to perform special processing when an application is idle. An application is idle when it is not processing code. For example, an application is idle when it is waiting for input from the user.

OnIdle is called only once, as the application transitions into an idle state. It is not called continuously unless the Done parameter is set to false. Applications that set Done to false consume an inordinate amount of CPU time, which affects overall system performance.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
*/
// global variables to show the order of events
int  XPos, YPos, Delta;

// This is a utility procedure to display messages
// add this at the beginning of the unit

void StatusMsg(TForm *Form, TCanvas *Canvas, char *Message, bool Bkg)
{
  if (!Bkg)
    Canvas->Font->Style = TFontStyles() << fsBold; // foreground messages are bold
  Canvas->TextOut(XPos, YPos, Message);
  if (!Bkg)
    Canvas->Font->Style = TFontStyles();
  // change Xpos and YPos to prepare for the next message
  YPos += Delta;
  if (YPos >= Form->ClientHeight - 10)
  {
    YPos = 10;
    XPos += 180;
  }
  if (XPos >= Form->ClientWidth - 100)
  {
    if (Canvas->Font->Color == clRed)
      Canvas->Font->Color = clBlack;
    else
      Canvas->Font->Color = clRed;
    XPos = 10;
  }
}

/*
This is the OnCreate event handler for Form1.  It
initializes global values and sets the application's OnIdle
event handler
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  Button1->Caption = "Do not yield";
  Button2->Caption  = "Handle Message";
  Application->OnIdle = MyIdleHandler;
  XPos = 10;
  YPos = 10;
  if (Canvas->Font->Height > 0)
    Delta = Canvas->Font->Height + 1;
  else
    Delta = 1 - Canvas->Font->Height;
}

/*
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some background
processing for the application.  This one simply writes a
message to let you know when it's there.
*/
void __fastcall TForm1::MyIdleHandler(TObject *Sender, bool &Done)
{
  StatusMsg(Form1, Canvas, "This represents a background process.", true);
}

/*
This is the OnClick event handler for Button1.  It
simulates a lengthy process
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button1Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
      Y = random(J);
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button1Click handler is done", false);
}

/*
This is the OnClick event handler for button2.  It
simulates a lengthy process that allows other messages to be
processed.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button2Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
    {
      Y = random(J);
    }
    // Yield to OnIdle or other messages
    Application->HandleMessage();
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button2Click handler is done", false);
}
```

Delphi Examples:

```
{
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
}
var
{ global variables to show the order of events. }
  XPos, YPos, Delta: integer;

{ This is a utility procedure to display messages }
{ add this at the beginning of the implementation section }
procedure StatusMsg(
  MyForm : TForm1; Canvas : TCanvas; Message : string; Bkg : Boolean);
begin
  if not bkg then
    Canvas.Font.Style := [fsBold]; {foreground messages are bold }
  Canvas.TextOut(XPos, YPos, Message);
  if not bkg then
    Canvas.Font.Style := [];
  { change Xpos and YPos to prepare for the next message }
  Ypos := Ypos + Delta;
  if YPos >= MyForm.ClientHeight - 10 then
  begin
    YPos := 10;
    Xpos := Xpos + 180;
  end;
  if (Xpos >= MyForm.ClientWidth - 100) then
  begin
    if (Canvas.Font.Color = clRed) then
      Canvas.Font.Color := clBlack
    else
      Canvas.Font.Color := clRed;
    Xpos := 10;
  end;
end;

{This is the Form's OnCreate event handler.  }
{ It initializes global variables and sets the  the OnIdle
event handler }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Do not yield';
  Button2.Caption := 'Handle Message';
  Application.OnIdle:= MyIdleHandler;
  XPos := 10;
  YPos := 10;
  Delta := Abs(Canvas.Font.Height) + 1;
 end;

{
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some
background processing for the application.  This one simply
writes a message to let you know when it's there.
}
procedure TForm1.MyIdleHandler(Sender: TObject; var Done: Boolean);
begin
  StatusMsg(
    Form1, Canvas, 'This represents a background process.', True);
end;

{
Set this method as the OnClick event handler of Button1.  It
performs a calculation without yielding to other messages or
```

```
  idle time.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, J, X, Y: Word;
  begin
    StatusMsg(
      Form1, Canvas, 'The synchronous handler is starting', False);
    I := 0;
    J := 0;
    while I < 10 do
    begin
      Randomize;
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    StatusMsg(
      Form1, Canvas, 'The synchronous handler is done', False);
  end;

  {
  Set this method as the OnClick event handler of Button2.  It
  performs a calculation but calls HandleMessage to allow idle
  time or asynchronous message processing.
  }
  procedure TForm1.Button2Click(Sender: TObject);
  var
    I, J, X, Y: Word;
  begin
   StatusMsg(
      Form1, Canvas, 'The asynchronous handler is starting', False);
    I := 0;
    J := 0;
    while I < 10 do
    begin
      Randomize;
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
      { yield to OnIdle or other messages }
      Application.HandleMessage;
    end;
    StatusMsg(
      Form1, Canvas, 'The asynchronous handler is done', False);
  end;
```

See Also

OnIdle

CheckSynchronize

OnUpdate

7.1.1.6.11 TApplication.OnMessage Event

Occurs when the application receives a Windows message.

Pascal

```
property OnMessage: TMessageEvent;
```

C++

```
__property TMessageEvent OnMessage;
```

Description

Use OnMessage to trap any or all Windows messages posted to all windows in the application. The OnMessage event occurs when an application receives a Windows message. An OnMessage event handler allows an application to respond to messages other than those declared in the events for TApplication (see page 1924). If the application doesn't have a specific handler for an incoming message, the message is dispatched to the window for which it was intended, and Windows handles the message.

Note: OnMessage only receives messages that are posted to the message queue, not those sent directly with the Windows API SendMessage function.

Warning: Caution:Thousands of messages per second flow though this event. Be careful when coding the handler, because it can affect the performance of the entire application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code handles a custom message that the
application sends to itself when a file is ready for reading.
*/
const WM_FILEREADY = WM_USER + 2000;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  const PAnsiChar Path = "../overview.rtf";
  PostMessage(Application->Handle, WM_FILEREADY, 0, int(Path));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMessage = AppMessage;
}

void __fastcall TForm1::AppMessage(tagMSG &Msg, bool &Handled)
{
  if (Msg.message == WM_FILEREADY)
  {
    Memo1->Lines->LoadFromFile(AnsiString((char *)Msg.lParam));
    Handled = true;
  }
/*
For all other messages, Handled remains False so that other
message handlers can respond.
*/
}
```

Delphi Examples:

```
{
The following code handles a custom message that the
application sends to itself when a file is ready for reading.
}
const WM_FILEREADY = WM_USER + 2000;
procedure TForm1.Button1Click(Sender: TObject);
const Path: PChar = 'OverView.RTF';
begin
  PostMessage(Application.Handle, WM_FILEREADY, 0, integer(Path));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMessage := AppMessage;
end;

procedure TForm1.AppMessage(var Msg: TMsg; var Handled: Boolean);
var msgStr: String;
begin
  if Msg.hwnd = Application.Handle then
  begin
    if Msg.message = WM_FILEREADY then
    begin
      msgStr:= PChar(Msg.lParam);
      Memo1.Lines.LoadFromFile(msgStr);
      Handled := True;
    end;
  end;
  { for all other messages, Handled remains False }
  { so that other message handlers can respond }
end;
```

See Also

HandleMessage (see page 1937)

ProcessMessages (see page 1954)

OnMessage

7.1.1.6.12 TApplication.OnMinimize Event

Occurs when an application is minimized.

Pascal

```
property OnMinimize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMinimize;
```

Description

Write an OnMinimize event handler to perform special processing when the application is minimized. The application is minimized, either because the user minimizes the main window, or because of a call to the Minimize (see page 1951) method. The Icon (see page 1981) property determines the icon that represents the minimized application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

Icon (see page 1981)

Minimize (see page 1951)

OnRestore (see page 2006)

Restore (see page 1957)

OnMinimize

7.1.1.6.13 TApplication.OnModalBegin Event

Occurs when a modal form is opened.

Pascal

```
property OnModalBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalBegin;
```

Description

Write an OnModalBegin event handler to perform special processing when a modal form is opened.

See Also

OnModalEnd (see page 2006)

ModalStarted (see page 1952)

7.1.1.6.14 TApplication.OnModalEnd Event

Occurs when a modal form is closed.

Pascal

```
property OnModalEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalEnd;
```

Description

Write an OnModalEnd event handler to perform special processing when a modal form is closed.

See Also

OnModalBegin (see page 2006)

ModalStarted (see page 1952)

7.1.1.6.15 TApplication.OnRestore Event

Occurs when the previously minimized application is restored to its normal size.

Pascal

```
property OnRestore: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRestore;
```

Description

Write an OnRestore event handler to perform special processing when the application is restored from the minimized state in which it appears as an icon. An application is restored either because the user restores the application, or because the application calls the Restore (see page 1957) method.

Note: Don't confuse restoring an application with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of the WindowState property.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

Icon (see page 1981)

Minimize (see page 1951)

Restore (see page 1957)

OnRestore

WindowState (see page 2195)

7.1.1.6.16 TApplication.OnSettingChange Event

Occurs when Windows notifies the application that a system-wide setting has changed.

Pascal

```
property OnSettingChange: TSettingChangeEvent;
```

C++

```
__property TSettingChangeEvent OnSettingChange;
```

Description

Use OnSettingChange to respond when Windows informs the application that a system-wide setting or policy has changed.

7.1.1.6.17 TApplication.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form or its controls handle them. When the user presses a key, the application can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut lets the application implement additional shortcuts.

If the OnShortCut event handler responds to the keystroke, set its Handled parameter to true. This prevents the keystroke from being passed on to menus or actions associated with the application. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnShortCut (see page 2235)

OnKeyDown

OnShortCut

7.1.1.6.18 TApplication.OnShowHint Event

Occurs when the application is about to display the hint window for a Help Hint (see page 1976).

Pascal

```
property OnShowHint: TShowHintEvent;
```

C++

```
__property TShowHintEvent OnShowHint;
```

Description

Write an OnShowHint event handler to change the appearance and behavior of Help Hints.

In the event handler, set the HintStr parameter to change the text of the Help Hint (see page 1976). To obtain the text of a hint for a particular control, call the GetLongHint or GetShortHint function, and assign the result to HintStr.

Use the CanShow parameter to permit or prevent the Help Hint (see page 1976) from displaying.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
*/
void __fastcall TForm1::DoShowHint(UnicodeString &HintStr, bool &CanShow, THintInfo &HintInfo)
{
  if (HintInfo.HintControl == SpeedButton3)
  {
    HintInfo.HintColor = clAqua;// Changes only for this hint
    HintInfo.HintMaxWidth = 120; // Hint text word wraps if width is greater than 120
    HintInfo.HintPos.x += SpeedButton3->Width; // Move hint to right edge
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->ShowHint = true;
  Application->OnShowHint = DoShowHint;
}
```

Delphi Examples:

```
{
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
```

```
  implementation part of the unit. Finally, in the OnCreate
  event handler for the form, assign the method to the
  OnShowHint event of the application.
  }

  procedure TForm1.DoShowHint(var HintStr: String; var CanShow: Boolean; var HintInfo:
  THintInfo);
  begin
    if HintInfo.HintControl = SpeedButton3 then
    begin
      with HintInfo do
      begin
        HintColor := clAqua;{ Changes only for this hint }
        HintMaxWidth := 120;{Hint text word wraps if width is greater than 120 }
        Inc(HintPos.X, SpeedButton3.Width); { Move hint to right edge }
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.ShowHint := True;
    Application.OnShowHint := DoShowHint;
  end;
```

See Also

GetLongHint

GetShortHint

HintHidePause (see page 1980)

OnHint (see page 1997)

OnShowHint

7.1.2 TControlScrollBar Class

TControlScrollBar displays a horizontal or vertical scroll bar.

Class Hierarchy

Pascal

```
TControlScrollBar = class(TPersistent);
```

C++

```
class TControlScrollBar : public TPersistent;
```

File

Forms

Description

Use TControlScrollBar to control the scroll bar in a scrolling windowed control such as TForm (see page 2241) or TScrollBox (see page 2483). TControlScrollBar introduces properties that control the size of the scrollable region and the position to which that region is scrolled, and when the scroll bar appears.

See Also

HorzScrollBar

TForm (see page 2241)

TScrollBox (see page 2483)

VertScrollBar

TScrollBar

7.1.2.1 TControlScrollBar Members

The following tables list the members exposed by TControlScrollBar.

Public Methods

	Name	Description
	Assign (see page 2011)	Copies the contents of a TControlScrollBar (see page 2009) instance to another TControlScrollBar (see page 2009).
	ChangeBiDiPosition (see page 2011)	Changes the position of a horizontal scroll bar to match the current bi-directional mode.
	IsScrollBarVisible (see page 2011)	Returns true if the scroll bar is visible.

Public Properties

	Name	Description
	Kind (see page 2015)	Indicates whether the scroll bar is horizontal or vertical.
	ScrollPos (see page 2020)	Indicates the position of the thumb tab.

Published Properties

	Name	Description
	ButtonSize (see page 2012)	Specifies the sizes of the buttons in the scroll bar.
	Color (see page 2013)	Specifies the scroll bar's color.
	Increment (see page 2013)	Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.
	Margin (see page 2015)	Determines when a scroll bar is generated.
	ParentColor (see page 2016)	Specifies whether the scroll bar should.
	Position (see page 2017)	Specifies the position of the thumb tab on the scroll bar.
	Range (see page 2018)	Determines how far the scrolling region of the associated control can move.
	Size (see page 2021)	Specifies the scroll bar's size.
	Smooth (see page 2021)	Specifies whether the amount the associated control scrolls is dynamically computed.
	Style (see page 2022)	Specifies the scroll bar's style.
	ThumbSize (see page 2022)	This is ThumbSize, a member of class TControlScrollBar.
	Tracking (see page 2022)	Determines whether the form or scroll box moves before the thumb tab is released.
	Visible (see page 2022)	Determines whether the scroll bar appears.

7.1.2.2 TControlScrollBar Methods

The methods of the TControlScrollBar class are listed here.

Public Methods

	Name	Description
	Assign (see page 2011)	Copies the contents of a TControlScrollBar (see page 2009) instance to another TControlScrollBar (see page 2009).

	ChangeBiDiPosition (see page 2011)	Changes the position of a horizontal scroll bar to match the current bi-directional mode.
	IsScrollBarVisible (see page 2011)	Returns true if the scroll bar is visible.

7.1.2.2.1 TControlScrollBar.Assign Method

Copies the contents of a TControlScrollBar (see page 2009) instance to another TControlScrollBar (see page 2009).

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the Visible (see page 2022), Range (see page 2018), Position (see page 2017), and Increment (see page 2013) properties from the source object to the TControlScrollBar (see page 2009) instance. If the Source parameter is not another scroll bar, Assign calls the TPersistent Assign method, which allows the source object to copy properties if it has an AssignTo method that knows how to copy to a control scroll bar.

See Also

Assign

AssignTo

Position (see page 2017)

Visible (see page 2022)

Range (see page 2018)

7.1.2.2.2 TControlScrollBar.ChangeBiDiPosition Method

Changes the position of a horizontal scroll bar to match the current bi-directional mode.

Pascal

```
procedure ChangeBiDiPosition;
```

C++

```
__fastcall ChangeBiDiPosition();
```

Description

Use ChangeBiDiPosition to change the position of a horizontal scroll bar to reflect the current bi-directional mode of its associated scrolling windowed control.

For example, TScrollingWinControl (see page 2554) calls ChangeBiDiPosition from its DoFlipChildren method to determine whether the scroll bar should be repositioned along with the child controls.

See Also

BiDiMode

DoFlipChildren (see page 2560)

7.1.2.2.3 TControlScrollBar.IsScrollBarVisible Method

Returns true if the scroll bar is visible.

Pascal

```
function IsScrollBarVisible: Boolean;
```

C++

```
__fastcall Boolean IsScrollBarVisible();
```

Description

Call IsScrollBarVisible to determine whether the scroll bar is visible. IsScrollBarVisible returns true if the Visible (see page 2022) property is true and the associated scrolling windowed control contains child controls that lie within Margin (see page 2015) pixels from the edge.

See Also

Visible (see page 2022)

Margin (see page 2015)

7.1.2.3 TControlScrollBar Properties

The properties of the TControlScrollBar class are listed here.

Public Properties

	Name	Description
R	Kind (see page 2015)	Indicates whether the scroll bar is horizontal or vertical.
R	ScrollPos (see page 2020)	Indicates the position of the thumb tab.

Published Properties

	Name	Description
	ButtonSize (see page 2012)	Specifies the sizes of the buttons in the scroll bar.
	Color (see page 2013)	Specifies the scroll bar's color.
	Increment (see page 2013)	Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.
	Margin (see page 2015)	Determines when a scroll bar is generated.
	ParentColor (see page 2016)	Specifies whether the scroll bar should.
	Position (see page 2017)	Specifies the position of the thumb tab on the scroll bar.
	Range (see page 2018)	Determines how far the scrolling region of the associated control can move.
	Size (see page 2021)	Specifies the scroll bar's size.
	Smooth (see page 2021)	Specifies whether the amount the associated control scrolls is dynamically computed.
	Style (see page 2022)	Specifies the scroll bar's style.
	ThumbSize (see page 2022)	This is ThumbSize, a member of class TControlScrollBar.
	Tracking (see page 2022)	Determines whether the form or scroll box moves before the thumb tab is released.
	Visible (see page 2022)	Determines whether the scroll bar appears.

7.1.2.3.1 TControlScrollBar.ButtonSize Property

Specifies the sizes of the buttons in the scroll bar.

Pascal

```
property ButtonSize: Integer;
```

C++

```
__property int ButtonSize;
```

Description

Use ButtonSize to specify the sizes of the buttons in the scroll bar. If the scroll bar is horizontal, ButtonSize is the width of the arrow buttons in pixels. If the scroll bar is vertical, ButtonSize is the height of the arrow buttons in pixels.

See Also

Size (see page 2021)

Kind (see page 2015)

thumbsize (see page 2022)

7.1.2.3.2 TControlScrollBar.Color Property

Specifies the scroll bar's color.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to specify the scroll bar's color. When you explicitly assign a value to Color, the ParentColor (see page 2016) property is automatically set to false.

If the scroll bar's ParentColor (see page 2016) property is true, then the Color property reflects the Windows button highlight color (clBtnHighlight (see page 2900)), and changing this color in the Windows control panel automatically updates the value of Color.

See Also

ParentColor (see page 2016)

7.1.2.3.3 TControlScrollBar.Increment Property

Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.

Pascal

```
property Increment: TScrollBarInc;
```

C++

```
__property TScrollBarInc Increment;
```

Description

Use Increment to control how the scroll bar affects the scrollable region of the control. If Increment is set to 1, each click scrolls by one position; if Increment is set to 2, each click scrolls by two positions; and so forth.

Note: Do not use the Increment property when Smooth (see page 2021) is set to true. When Smooth (see page 2021) is true, every time the range or visibility of the scroll bar changes, the value of Increment is dynamically recomputed.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

Smooth (see page 2021)

7.1.2.3.4 TControlScrollBar.Kind Property

Indicates whether the scroll bar is horizontal or vertical.

Pascal

```
property Kind: TScrollBarKind;
```

C++

```
__property TScrollBarKind Kind;
```

Description

Kind indicates whether the scroll bar has a vertical or horizontal orientation. This value is initialized in the constructor, when the scroll bar is created by a scrolling windowed control for its VertScrollBar or HorzScrollBar property.

See Also

VertScrollBar (see page 2563)

HorzScrollBar (see page 2562)

Create

7.1.2.3.5 TControlScrollBar.Margin Property

Determines when a scroll bar is generated.

Pascal

```
property Margin: Word;
```

C++

```
__property Word Margin;
```

Description

The Margin property determines the minimum number of pixels that must separate each control from the edge of the control that uses the scroll bar. At runtime, whenever a child control is less than Margin pixels from the edge and Visible (see page 2022) is set to true, a scroll bar appears.

C++ Examples:

```
/*
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  Button1->Left = Button1->Left + 25;
  Label1->Caption = IntToStr(HorzScrollBar->Range);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  HorzScrollBar->Margin = 25;
  HorzScrollBar->Increment = 10;
}
```

Delphi Examples:

```
{
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with HorzScrollBar do
  begin
    Margin:= 25;
    Increment := 10;
    Visible := True;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Left := Button1.Left + 25;
  Label1.Caption := IntToStr(HorzScrollBar.Range);
end;
```

See Also

Visible (see page 2022)

IsScrollBarVisible (see page 2011)

7.1.2.3.6 TControlScrollBar.ParentColor Property

Specifies whether the scroll bar should.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

Use ParentColor to indicate that the scroll bar should always reflect the Windows button highlight color. Using ParentColor differs from setting the Color (see page 2013) property to clBtnHighlight (see page 2900) in that when ParentColor is true, changing the button highlight color in the Windows control panel automatically updates the value of Color (see page 2013).

When the value of the Color (see page 2013) property is changed, its ParentColor property is automatically set to false.

See Also

Color (see page 2013)

TColor (see page 2878)

7.1.2.3.7 TControlScrollBar.Position Property

Specifies the position of the thumb tab on the scroll bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Use Position to programmatically scroll the client area of the associated control. The value of Position changes (from 0 to Range (see page 2018)) at runtime as the user scrolls the scroll bar.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

Range (see page 2018)

ScrollPos (see page 2020)

7.1.2.3.8 TControlScrollBar.Range Property

Determines how far the scrolling region of the associated control can move.

Pascal

```
property Range: Integer;
```

C++

```
__property int Range;
```

Description

Range represents the virtual size (in pixels) of the associated control's client area. For example, if the Range of a form's horizontal scroll bar is set to 500, and the width of the form is 200, the scroll bar's Position (see page 2017) can vary from 0 to 300.

If the Range of a horizontal scroll bar is less than the width of the form or scroll box, no horizontal scroll bar appears. If the Range of a vertical scroll bar is less than the height of the form or scroll box, no vertical scroll bar appears.

The value of Range is set automatically, but can be overridden in the Object Inspector or in program code.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Button1->Left = Button1->Left + 25;
  Label1->Caption = IntToStr(HorzScrollBar->Range);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  HorzScrollBar->Margin = 25;
  HorzScrollBar->Increment = 10;
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
```

```
      { This means that the form's logical size is twice as big }
      { as the physical window. }
      { Note that Range must always be larger than the ClientWidth }
      Range := 600;
      Position := Range - ClientWidth; { start form out fully scrolled }
      Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
      Visible := True;  { Show the scrollbar }
    end;
  end;
  {
  This example uses a button and a label on a form and needs
  the AutoScroll property set on the form. Place the label
  near the left side of the form, and place the button
  somewhere near the middle of the form. When the user runs
  the application, a horizontal scroll bar does not appear,
  because no control on the form is close enough to the right
  edge.  Each time the user clicks the button, the button moves
  25 pixels to the right, and the calculated Range value is
  reported in the caption of the label. Repeatedly clicking the
  button eventually moves the button close enough to the edge
  of the form (within the Margin amount) so that a horizontal
  scroll bar appears.  Note that the Range property is
  continuously updated as the button moves to the right.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with HorzScrollBar do
    begin
      Margin:= 25;
      Increment := 10;
      Visible := True;
    end;
  end;
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Button1.Left := Button1.Left + 25;
    Label1.Caption := IntToStr(HorzScrollBar.Range);
  end;
  {
  The following example demonstrates form scrolling with the
  PgUp and PgDn keys.
  }
  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
  const
    PageDelta = 10;
  begin
    With VertScrollbar do
      if Key = VK_NEXT then
        Position := Position + PageDelta
      else if Key = VK_PRIOR then
        Position := Position - PageDelta;
  end;
```

See Also

Position (see page 2017)

Width

Height

7.1.2.3.9 TControlScrollBar.ScrollPos Property

Indicates the position of the thumb tab.

Pascal

```
property ScrollPos: Integer;
```

C++

```
__property int ScrollPos;
```

Description

The read-only ScrollPos property returns the current value of Position (see page 2017) when the scroll bar is visible. If the scroll bar is not visible, ScrollPos returns 0.

See Also

Position (see page 2017)

Visible (see page 2022)

7.1.2.3.10 TControlScrollBar.Size Property

Specifies the scroll bar's size.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size to specify the width of the scroll bar in pixels.

Note: The length of the scroll bar is determined by the width (for horizontal scroll bars) or height (for vertical scroll bars) of the associated scrolling windowed control.

See Also

ButtonSize (see page 2012)

thumbsize (see page 2022)

7.1.2.3.11 TControlScrollBar.Smooth Property

Specifies whether the amount the associated control scrolls is dynamically computed.

Pascal

```
property Smooth: Boolean;
```

C++

```
__property Boolean Smooth;
```

Description

Set Smooth to true to specify that the scroll bar should dynamically adjust its scrolling increment based on the size of the associated control's client area.

When Smooth is true, clicking the end arrows of the scroll bar scrolls the associated control by approximately one tenth of its client area, and clicking the scroll bar beside the thumb tab scrolls the control by the length of the client area minus the amount scrolled by the end arrows. When Smooth is false, the Increment (see page 2013) property controls the amount of scrolling when the user clicks the end arrows, and clicking beside the thumb tab scrolls the control by the entire length of the client area.

See Also

Increment (see page 2013)

Position (see page 2017)

7.1.2.3.12 TControlScrollBar.Style Property

Specifies the scroll bar's style.

Pascal

```
property Style: TScrollBarStyle;
```

C++

```
__property TScrollBarStyle Style;
```

Description

Use Style to specify the scroll bar's style. The style determines whether the scroll bar appears three-dimensional, flat, or uses hot tracking.

7.1.2.3.13 TControlScrollBar.ThumbSize Property

Pascal

```
property ThumbSize: Integer;
```

C++

```
__property int ThumbSize;
```

Description

This is ThumbSize, a member of class TControlScrollBar.

7.1.2.3.14 TControlScrollBar.Tracking Property

Determines whether the form or scroll box moves before the thumb tab is released.

Pascal

```
property Tracking: Boolean;
```

C++

```
__property Boolean Tracking;
```

Description

Use Tracking to control whether the scroll bar and associated control repaint while the thumb tab is dragged with the mouse. If Tracking is true, the image moves as the user drags the thumb tab. If Tracking is false, the image remains static until the thumb tab is released.

Repainting while the thumb tab is dragged can provide useful feedback to the user about how far the control has scrolled, but slows performance.

7.1.2.3.15 TControlScrollBar.Visible Property

Determines whether the scroll bar appears.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

If Visible is set to true, a scroll bar appears whenever a visual component is less than Margin (see page 2015) pixels from the edge of the form or scroll box. If Visible is set to false, the scroll bar is never visible. This is useful, for example, for programmatically controlling the scroll position of a form without allowing the user to control the scroll position.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
```

```
      Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
      Visible := True;  { Show the scrollbar }
    end;
  end;
  {
  The following example demonstrates form scrolling with the
  PgUp and PgDn keys.
  }
  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
  const
    PageDelta = 10;
  begin
    With VertScrollbar do
      if Key = VK_NEXT then
        Position := Position + PageDelta
      else if Key = VK_PRIOR then
        Position := Position - PageDelta;
  end;
```

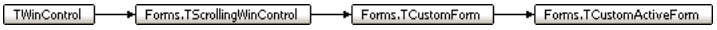
See Also

Margin (see page 2015)

7.1.3 TCustomActiveForm Class

TCustomActiveForm is the base class for Active (see page 2145) Forms (see page 1921).

Class Hierarchy



Pascal

```
TCustomActiveForm = class(TCustomForm);
```

C++

```
class TCustomActiveForm : public TCustomForm;
```

File

Forms

Description

TCustomActiveForm is used in code that is generated automatically by the ActiveForm wizard. For more information, see Generating an ActiveX control based on a VCL form.

7.1.3.1 TCustomActiveForm Members

The following tables list the members exposed by TCustomActiveForm.

Protected Events

	Name	Description
	OnCanResize (see page 2202)	Occurs when an attempt is made to resize the control.
	OnClose (see page 2205)	Occurs when the form closes.
	OnCloseQuery (see page 2206)	Occurs when close is attempted.
	OnHelp (see page 2217)	Occurs when the form receives a request for help.

	OnHide (see page 2217)	Occurs when the form is hidden (that is, when its Visible (see page 2192) property is set to false).
	OnResize (see page 2235)	Occurs immediately after the control is resized.
	OnShortCut (see page 2235)	Occurs when the user presses a key (before the OnKeyDown (see page 2219) event).
	OnShow (see page 2236)	Occurs when the form is shown (that is, when its Visible (see page 2192) property is set to true).

Protected Fields

	Name	Description
	FFormState (see page 2110)	This is FFormState, a member of class TCustomForm.

Protected Methods

	Name	Description
	Activate (see page 2111)	Activation event dispatcher.
	ActiveChanged (see page 2112)	Active (see page 2145) state event dispatcher.
V	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
V	AlignControls (see page 2113)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
V	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
V	BeginAutoDrag (see page 2113)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
V	ChangeScale (see page 2114)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
V	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CloseModal (see page 2115)	This is CloseModal, a member of class TCustomForm.
V	CreateParams (see page 2029)	Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 2033) property.
V	CreateWindowHandle (see page 2116)	Creates a form window given a set of window-creation parameters.
V	CreateWnd (see page 2117)	Creates the form window.
V	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	Deactivate (see page 2117)	Generates an OnDeactivate (see page 2209) event.
V	DefineProperties (see page 2118)	Designates methods for storing a form's unpublished data on a stream such as a form file.
V	DestroyHandle (see page 2119)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 2119)	Destroys the window created in the CreateWindowHandle (see page 2116) method.
	DoClose (see page 2120)	Form closing event dispatcher.
V	DoCreate (see page 2120)	Form creation event dispatcher.
V	DoDestroy (see page 2121)	Form destruction event dispatcher.
V	DoDock (see page 2121)	Docking event dispatcher.
V	DoFlipChildren (see page 2560)	Flips the control's children.
	DoHide (see page 2122)	Form hide event dispatcher.
	DoShow (see page 2122)	Form show event dispatcher.
	get_ActiveMDIChild (see page 2123)	This is get_ActiveMDIChild, a member of class TCustomForm.
	get_MDIChildCount (see page 2123)	This is get_MDIChildCount, a member of class TCustomForm.
	get_MDIChildren (see page 2123)	This is get_MDIChildren, a member of class TCustomForm.
	GetBorderIconStyles (see page 2123)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 2124)	This is GetBorderStyles, a member of class TCustomForm.
V	GetClientRect (see page 2124)	Returns the value of the ClientRect property.
V	GetFloating (see page 2125)	Returns the value of the Floating property.
	GetOwnerWindow (see page 2127)	This is GetOwnerWindow, a member of class TCustomForm.
	HandleCreateException (see page 2127)	Handles exceptions that are raised by the OnCreate (see page 2207) event handler.
	InitializeNewForm (see page 2128)	This is InitializeNewForm, a member of class TCustomForm.
V	Loaded (see page 2129)	Performs fix-ups when the form is first loaded into memory.

	Notification (see page 2130)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 2131)	Generates an OnPaint (see page 2233) event.
	PaintWindow (see page 2131)	Renders the image of the form.
	PaletteChanged (see page 2132)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	QueryInterface (see page 2132)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2133)	Reads the form's published properties, stored data, and owned components from a reader object.
	RequestAlign (see page 2134)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	Resizing (see page 2134)	Makes any required adjustments when the form changes size.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	SetChildOrder (see page 2135)	Changes the order in which Child appears when GetChildren (see page 2124) is called.
	SetParent (see page 2136)	Sets the Parent property.
	SetParentBiDiMode (see page 2136)	Sets the ParentBiDiMode property.
	UpdateActions (see page 2140)	Updates all actions associated with the form.
	UpdateWindowState (see page 2141)	Updates the WindowState (see page 2195) property to reflect the current state of the form.
	ValidateRename (see page 2141)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 2141)	Responds just before the form changes the value of its Visible (see page 2192) property.
	WndProc (see page 2142)	Provides specific message responses for the form.

Protected Properties

	Name	Description
	ActiveMDIChild (see page 2146)	Specifies the MDI child that has focus.
	AlphaBlend (see page 2147)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2147)	Specifies the degree of translucency on a translucent form.
	BorderIcons (see page 2148)	Specifies which icons appear on the title bar of the form.
	ClientHandle (see page 2154)	Provides access to the handle of the form's internal MDI client window.
	ClientHeight (see page 2154)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 2155)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 2158)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2158)	Specifies the monitor on which the form appears.
	FormStyle (see page 2162)	Determines the form's style.
	Icon (see page 2167)	Specifies the icon that appears when the form is minimized.
	MDIChildCount (see page 2172)	Specifies the number of open MDI child forms.
	MDIChildren (see page 2173)	Provides indexed access to all MDI child forms.
	ObjectMenuItem (see page 2177)	Represents an OLE object menu item that reacts to selections of OLE objects.
	ParentFont (see page 2179)	Determines where a control looks for its font information.
	PopupChildren (see page 2179)	This is PopupChildren, a member of class TCustomForm.
	Position (see page 2185)	Represents the size and placement of the form.
	TileMode (see page 2188)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 2190)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2191)	Indicates the color on the form that appears transparent when TransparentColor (see page 2190) is true.
	WindowMenu (see page 2194)	Specifies the Window menu for an MDI parent form.

Public Methods

	Name	Description
	AfterConstruction (see page 2112)	Construction event dispatcher.
	BeforeDestruction (see page 2113)	Destruction event dispatcher.
	Close (see page 2114)	Closes the form.
	CloseQuery (see page 2115)	Close (see page 2114) attempt event dispatcher.

	Create (see page 2029)	Creates a new TCustomActiveForm (see page 2024) object.
	CreateNew (see page 2108)	Creates and initializes a new form.
	DefaultHandler (see page 2118)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 2118)	Removes focus from a control on the form.
	Destroy (see page 2109)	Removes the form from memory.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	Dock (see page 2120)	Handles automatic docking of the form into other windows.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	FocusControl (see page 2122)	Sets focus to a control on the form.
	GetChildren (see page 2124)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetFormImage (see page 2125)	Returns a bitmap of the form.
	Hide (see page 2127)	Hides the form.
	IsShortCut (see page 2128)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 2129)	Ensures that the form is fully visible on a specified monitor.
	MouseWheelHandler (see page 2130)	Directs mouse wheel messages to the appropriate control.
	Print (see page 2132)	Prints the form.
	RecreateAsPopup (see page 2133)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 2133)	Destroys the form and frees its associated memory.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 2134)	Cancels modes on the form.
	set_PopupParent (see page 2135)	This is set_PopupParent, a member of class TCustomForm.
	SetFocus (see page 2135)	Sets focus to the form.
	SetFocusedControl (see page 2136)	Sets focus to a control on the form.
	Show (see page 2137)	Shows the form.
	ShowModal (see page 2139)	Shows a form as a modal dialog.
	WantChildKey (see page 2030)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 2144)	Designates the action associated with the control.
R	Active (see page 2145)	Specifies whether the form has focus.
	ActiveOleControl (see page 2146)	Specifies the OLE control on the form that reacts to changes in focus.
	BorderStyle (see page 2149)	Specifies the appearance and behavior of the form border.
R	Canvas (see page 2150)	Provides access to the drawing area of the form.
	Designer (see page 2159)	Specifies the designer interface for the form.
	DropTarget (see page 2159)	Specifies whether the form is the target of a drag-and-drop operation.
R	FormState (see page 2161)	Indicates transitional state information about the form.
	GlassFrame (see page 2164)	This is GlassFrame, a member of class TCustomForm.
	HelpFile (see page 2165)	Specifies the name of the file the form uses to display Help.
	Menu (see page 2174)	Specifies the form's main menu.
	ModalResult (see page 2176)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 2177)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 2178)	Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.
	PopupMode (see page 2185)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 2185)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 2187)	Specifies whether form snaps to edge of screen.
	SnapBuffer (see page 2188)	Specifies distance for screen snap.
	Visible (see page 2054)	Indicates whether the form is visible.
	WindowState (see page 2195)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 2058)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 2064)	Occurs when the user clicks the control.
	OnContextPopup (see page 2066)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 2066)	Occurs when the form is created.
	OnDblClick (see page 2068)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2068)	Occurs when the form loses focus.
	OnDestroy (see page 2069)	Occurs when the form is destroyed.
	OnDragDrop (see page 2070)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2073)	Occurs when the user drags an object over a control.
	OnKeyDown (see page 2076)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2080)	Occurs when a key is pressed.
	OnKeyUp (see page 2081)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2083)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2084)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2086)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2086)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2086)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2089)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 2091)	Occurs when the form is redrawn.

Published Properties

	Name	Description
	ActiveControl (see page 2031)	Specifies the control that has focus on the form.
	Anchors (see page 2032)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2033)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2033)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AxBorderStyle (see page 2033)	Sets the active form's border style.
	BorderWidth (see page 2034)	Specifies the width of the control's border.
	Caption (see page 2034)	Specifies a text string that identifies the control to the user.
	Color (see page 2037)	Specifies the background color of the control.
	Constraints (see page 2038)	Specifies the size constraints for the control.
	Font (see page 2038)	Controls the attributes of text written on or in the control.
	Height (see page 2039)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2040)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 2041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 2171)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	OldCreateOrder (see page 2045)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 2045)	Specifies the padding of a control.
	PixelsPerInch (see page 2046)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 2046)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 2051)	Represents the proportions of a printed form.

	Scaled (see page 2052)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 2052)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Top (see page 2189)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 2054)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 2055)	Specifies the horizontal size of the control or form in pixels.

7.1.3.2 TCustomActiveForm.Create Constructor

Creates a new TCustomActiveForm (see page 2024) object.

Pascal

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

C++

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

Description

Create is the TCustomActiveForm (see page 2024) constructor.

AOwner specifies a component, typically the global application object, that becomes the value of the Owner property. The owner is responsible for initiating streaming of the active form and for freeing it.

See Also

Owner

ParentWindow

7.1.3.3 TCustomActiveForm Methods

The methods of the TCustomActiveForm class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2029)	Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 2033) property.

Public Methods

	Name	Description
	WantChildKey (see page 2030)	Indicates whether the form processes keyboard input for a control it owns.

7.1.3.3.1 TCustomActiveForm.CreateParams Method

Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 2033) property.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams performs the following tasks:

Calls the inherited CreateParams method.

Removes the WS_CAPTION flag from the Style property of Params.

Adds flags to the Style and ExStyle properties of Params according to the value of the AxBorderStyle (see page 2033) property, as shown in the following table:

AxBorderStyleStyleExStyle

AxBorderStyle (see page 2033)	Style	ExStyle
afbNone	Nothing	Nothing
afbSingle	WS_BORDER	Nothing
afbSunken	Nothing	WS_EX_CLIENTEDGE
afbRaised	WS_DLGFRAME	WS_EX_WINDOWEDGE

Note: The creation parameters are not updated when the form is in design mode.

See Also

CreateParams2

7.1.3.3.2 TCustomActiveForm.WantChildKey Method

Indicates whether the form processes keyboard input for a control it owns.

Pascal

```
function WantChildKey(Child: TControl; var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean WantChildKey(TControl * Child, TMessage Message);
```

Description

Do not call WantChildKey. It is used internally to ensure that keyboard events sent to child objects are handled correctly.

See Also

WantChildKey (see page 2142)

7.1.3.4 TCustomActiveForm Properties

The properties of the TCustomActiveForm class are listed here.

Public Properties

	Name	Description
	Visible (see page 2054)	Indicates whether the form is visible.

Published Properties

	Name	Description
	ActiveControl (see page 2031)	Specifies the control that has focus on the form.
	Anchors (see page 2032)	Specifies how the control is anchored to its parent.

	AutoScroll (see page 2033)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2033)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AxBorderStyle (see page 2033)	Sets the active form's border style.
	BorderWidth (see page 2034)	Specifies the width of the control's border.
	Caption (see page 2034)	Specifies a text string that identifies the control to the user.
	Color (see page 2037)	Specifies the background color of the control.
	Constraints (see page 2038)	Specifies the size constraints for the control.
	Font (see page 2038)	Controls the attributes of text written on or in the control.
	Height (see page 2039)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2040)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 2041)	Specifies whether the form should receive keyboard events before the active control.
	OldCreateOrder (see page 2045)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 2045)	Specifies the padding of a control.
	PixelsPerInch (see page 2046)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 2046)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 2051)	Represents the proportions of a printed form.
	Scaled (see page 2052)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 2052)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	VertScrollBar (see page 2054)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 2055)	Specifies the horizontal size of the control or form in pixels.

7.1.3.4.1 TCustomActiveForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
```

```
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 2146)

OnExit

OnEnter

Active (see page 2145)

7.1.3.4.2 TCustomActiveForm.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.3.4.3 TCustomActiveForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.3.4.4 TCustomActiveForm.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

7.1.3.4.5 TCustomActiveForm.AxBorderStyle Property

Sets the active form's border style.

Pascal

```
property AxBorderStyle: TActiveFormBorderStyle;
```

C++

```
__property TActiveFormBorderStyle AxBorderStyle;
```

Description

AxBorderStyle takes one of the following values:

ValuesMeaning

Values	Meaning	
afbNone	No border appears around the active form.	
afbSingle	A single line border appears around the active form.	
afbSunken	The active form gets a sunken border.	
afbRaised	The active form gets a raised border and no caption.	

See Also

BorderStyle (see page 2149)

7.1.3.4.6 TCustomActiveForm.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth;
```

C++

```
__property BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter

7.1.3.4.7 TCustomActiveForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
```

```
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

7.1.3.4.8 TCustomActiveForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

7.1.3.4.9 TCustomActiveForm.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

7.1.3.4.10 TCustomActiveForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

7.1.3.4.11 TCustomActiveForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
```

```
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

7.1.3.4.12 TCustomActiveForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 2561)

VertScrollBar (see page 2563)

TControlScrollBar (see page 2009)

7.1.3.4.13 TCustomActiveForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

ActiveControl (see page 2145)

OnKeyDown

OnKeyUp

OnKeyPress

7.1.3.4.14 TCustomActiveForm.OldCreateOrder Property

Specifies when OnCreate and OnDestroy events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate event occurs after all constructors are finished and the OnDestroy event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate event occurred when the TCustomForm constructor executed and the OnDestroy event occurred when the TCustomForm destructor executed. Applications that require the OnCreate event and OnDestroy events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 2207)

OnDestroy (see page 2209)

BeforeDestruction (see page 2113)

AfterConstruction (see page 2112)

7.1.3.4.15 TCustomActiveForm.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

7.1.3.4.16 TCustomActiveForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 2187)

7.1.3.4.17 TCustomActiveForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
```

```
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
```

```
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
```

```
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

7.1.3.4.18 TCustomActiveForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 2132)

7.1.3.4.19 TCustomActiveForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 2179)

Considerations When Dynamically Resizing Forms and Controls

7.1.3.4.20 TCustomActiveForm.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

7.1.3.4.21 TCustomActiveForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

TControlScrollBar (see page 2009)

7.1.3.4.22 TCustomActiveForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show and ShowModal methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront

Show (see page 2137)

ShowModal (see page 2139)

7.1.3.4.23 TCustomActiveForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
```

```
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

ClientWidth

Height

Left

7.1.3.5 TCustomActiveForm Events

The events of the TCustomActiveForm class are listed here.

Published Events

	Name	Description
	OnActivate (see page 2058)	Occurs when the form becomes active.
	OnClick (see page 2064)	Occurs when the user clicks the control.
	OnContextPopup (see page 2066)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 2066)	Occurs when the form is created.
	OnDblClick (see page 2068)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2068)	Occurs when the form loses focus.
	OnDestroy (see page 2069)	Occurs when the form is destroyed.
	OnDragDrop (see page 2070)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2073)	Occurs when the user drags an object over a control.
	OnKeyDown (see page 2076)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2080)	Occurs when a key is pressed.
	OnKeyUp (see page 2081)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2083)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2084)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2086)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2086)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2086)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2089)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 2091)	Occurs when the form is redrawn.

7.1.3.5.1 TCustomActiveForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication), not the form, occurs when Windows switches control from another application.

C++ Examples:

```
/*
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
*/

TForm1 *Form1;
TTable *Customers, *Blobs;
Boolean Form1Done = False;

void CreateCustSource()
{
  Customers = new TTable(Form1); // The owner will clean this up.
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
```

```
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  Form1->CustomerDS->DataSet = Customers;
  Customers->Active = True;
}

void CreateBlobSource()
{
  int i;
  Blobs = new TTable(Form1);
  Blobs->DatabaseName = "DBDEMOS";
  Blobs->TableType = ttParadox;
  Blobs->TableName = "MyBlobInfo";
  Blobs->Active = False;
    // Don't overwrite an existing table
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
  if (Blobs->Exists)
  {
    Blobs->Close();
    Blobs->DeleteTable();
  };
  // The Table component must not be active }
  // First, describe the type of table and give }
  // it a name }
  // Next, describe the fields in the table }
  Blobs->FieldDefs->Clear();
  TFieldDef *newDef = Blobs->FieldDefs->AddFieldDef();
  newDef->Name = "Field1";
  newDef->DataType = ftInteger;
  newDef->Required = True;
  newDef = Blobs->FieldDefs->AddFieldDef();
  newDef->Name = "Field2";
  newDef->DataType = ftBlob;
  newDef->Size = 30;
  // Next, describe any indexes }
  Blobs->IndexDefs->Clear();
  // The 1st index has no name because it is
  // a Paradox primary key }
  TIndexDef *newIDef = Blobs->IndexDefs->AddIndexDef();
  newIDef->Name = "";
  newIDef->Fields = "Field1";
  newIDef->Options << ixPrimary;
/*
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
*/
  // Call the CreateTable method to create the table }
  Blobs->CreateTable();
  Blobs->Active = True;
  for (int i = 1; i <= 20; i++)
    Blobs->AppendRecord(ARRAYOFCONST((i*111, IntToStr(i*222))));
  Form1->BlobDS->DataSet = Blobs;
  Blobs->Active = True;
};
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CreateCustSource();
  CreateBlobSource();
  for (int i = 0; i < ComponentCount; i++)
  {
    if (Components[i]->ClassNameIs("TDataSource"))
      ListBox1->Items->AddObject(Components[i]->Name,
         dynamic_cast<TObject *>(Components[i]));
  }
}

void __fastcall TForm1::ListBox1MouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift,
          int X, int Y)
{
  Form1Done = True;
}
/*
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it. Also, set the visible property
of Form2 to true.
*/
void __fastcall TForm2::FormActivate(TObject *Sender)
{
  if (Form1Done)
  {
    DBGrid2->DataSource =
      dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
    DBNavigator2->DataSource =
      dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
  };
}
```

Delphi Examples:

```
{
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
```

```
    begin
      if (Components[I] is TDataSource) then
        ListBox1.Items.AddObject(Components[I].Name,
          Components[I] as TObject);
    end;
  end;

  procedure CreateCustSource();
  begin
    Customers:= TTable.Create(Form1);
    with Customers do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'MyCustInfo';
      ReadOnly:= False;

  //    Don't overwrite an existing table
  //    Looking for c:/ProgramFiles/Common Files/Codegear Shared/Data/dbdemos.gdb
      if (not Customers.Exists) then CreateMyTable
      else
      begin
        if (Customers.Exists AND
          (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
  mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
        begin
          Customers.Close;
          Customers.DeleteTable;
          CreateMyTable;
        end;
      end;
    end;
    Form1.CustomerDS.DataSet:= Customers;
    Customers.Active:= True;
  end;

  procedure CreateMyTable();
  var
    i: Integer;
  begin
  //      The Table component must not be active
  //      First, describe the type of table and give
  //      it a name
  //      Next, describe the fields in the table
    with Customers do
    begin
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
```

```
        end;
        with AddFieldDef do
        begin
          Name := 'Field4';
          DataType := ftString;
          Size := 30;
        end;
      end;

//    Take out this part to remove indexing
//    Next, describe any indexes
      with IndexDefs do
      begin
        Clear;
//        The 1st index has no name because it is
//        a Paradox primary key
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;

//      Call the CreateTable method to create the table
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;

procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
    begin
      if (Blobs.Exists) then
      begin
        Blobs.Close;
        Blobs.DeleteTable;
      end;
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
```

```
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftBlob;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
{
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
}
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Blobs.Active:= True;
      for i := 1 to 20 do
        Blobs.AppendRecord([i*111, IntToStr(i*222)]);
    end;
  end;
  Form1.BlobDS.DataSet:= Blobs;
  Blobs.Active:= True;
end;

procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Form1Done := True;
end;

initialization

Form1Done := False;
{
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it. Also, set the visible property
of Form2 to true.
}
procedure TForm2.FormActivate(Sender: TObject);
begin
  if (Form1Done) then
```

```
  begin
    DBGrid2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    DBNavigator2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
  end;
end;
```

See Also

FormStyle (see page 2162)

OnDeactivate (see page 2209)

OnActivate (see page 1992)

7.1.3.5.2 TCustomActiveForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

7.1.3.5.3 TCustomActiveForm.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

7.1.3.5.4 TCustomActiveForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm's constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should

be freed by the OnDestroy event.

When a form is being created and its Visible property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint

Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);  // These tabsheets are cleaned up by their
owner (PageControl1).
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet #' + IntToStr(i);
        Brush.Color := colorPalette[i];
      end;
  end;

  procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
  begin
    PageControl1.SelectNextPage(Button = btNext);
  end;
```

See Also

OnDestroy (see page 2209)

OnActivate (see page 2196)

OnPaint (see page 2233)

OnShow (see page 2236)

OldCreateOrder (see page 2178)

7.1.3.5.5 TCustomActiveForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

7.1.3.5.6 TCustomActiveForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication object's OnDeactivate event.

See Also

OnActivate (see page 2196)

OnDeactivate (see page 1992)

7.1.3.5.7 TCustomActiveForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate event.

C++ Examples:

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
*/
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 2207)

System::TObject::Free

Destroy (see page 2109)

OldCreateOrder (see page 2178)

7.1.3.5.8 TCustomActiveForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}
```

```
void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
```

```
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.3.5.9 TCustomActiveForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
```

```
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
  This code is an OnDragOver event handler that won't allow a
  label control to be dropped on a panel control and stops the
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.3.5.10 TCustomActiveForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
```

```
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

7.1.3.5.11 TCustomActiveForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

7.1.3.5.12 TCustomActiveForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

7.1.3.5.13 TCustomActiveForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.3.5.14 TCustomActiveForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.3.5.15 TCustomActiveForm.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

7.1.3.5.16 TCustomActiveForm.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

7.1.3.5.17 TCustomActiveForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

7.1.3.5.18 TCustomActiveForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

7.1.3.5.19 TCustomActiveForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _TheGraphicCleaner(TheGraphic = new
Graphics::TBitmap());  // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
```

Delphi Examples:

```
{
```

```
  The following code loads a background bitmap onto the Canvas
  of the main form in the OnPaint event handler.
  }
  procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
  begin
    Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    TheGraphic := TBitmap.Create; { Create the bitmap object }
    TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    (TheGraphic as TObject).Free;
  end;
```

See Also

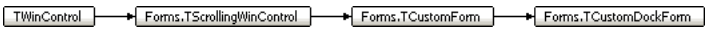
Canvas (see page 2150)

ClipRect (see page 2718)

7.1.4 TCustomDockForm Class

TCustomDockForm is the default floating dock site class for TControl objects.

Class Hierarchy



Pascal

```
TCustomDockForm = class(TCustomForm);
```

C++

```
class TCustomDockForm : public TCustomForm;
```

File

Forms

Description

When a control is created, TCustomDockForm is the type assigned to its FloatingDockSiteClass property. You can use TCustomDockForm as the basis for your own floating dock site class, and then assign the new class to the a control's FloatingDockSiteClass property.

See Also

TControl

7.1.4.1 TCustomDockForm Members

The following tables list the members exposed by TCustomDockForm.

Protected Events

	Name	Description
	OnActivate (see page 2196)	Occurs when the form becomes active.
	OnCanResize (see page 2202)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2203)	Occurs when the user clicks the control.
	OnClose (see page 2205)	Occurs when the form closes.
	OnCloseQuery (see page 2206)	Occurs when close is attempted.
	OnCreate (see page 2207)	Occurs when the form is created.
	OnDblClick (see page 2209)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2209)	Occurs when the form loses focus.
	OnDestroy (see page 2209)	Occurs when the form is destroyed.
	OnDragDrop (see page 2211)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2214)	Occurs when the user drags an object over a control.
	OnHelp (see page 2217)	Occurs when the form receives a request for help.
	OnHide (see page 2217)	Occurs when the form is hidden (that is, when its Visible (see page 2192) property is set to false).
	OnKeyDown (see page 2219)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2223)	Occurs when a key is pressed.
	OnKeyUp (see page 2224)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2226)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2227)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 2229)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2231)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 2233)	Occurs when the form is redrawn.
	OnResize (see page 2235)	Occurs immediately after the control is resized.
	OnShortCut (see page 2235)	Occurs when the user presses a key (before the OnKeyDown (see page 2219) event).
	OnShow (see page 2236)	Occurs when the form is shown (that is, when its Visible (see page 2192) property is set to true).

Protected Fields

	Name	Description
	FFormState (see page 2110)	This is FFormState, a member of class TCustomForm.

Protected Methods

	Name	Description
	Activate (see page 2111)	Activation event dispatcher.
	ActiveChanged (see page 2112)	Active (see page 2145) state event dispatcher.
V	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
V	AlignControls (see page 2113)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
V	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
V	BeginAutoDrag (see page 2113)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
V	ChangeScale (see page 2114)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
V	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CloseModal (see page 2115)	This is CloseModal, a member of class TCustomForm.
V	CreateParams (see page 2116)	Initializes the window-creation parameter record when the form window is created.
V	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.

	CreateWindowHandle (see page 2116)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 2117)	Creates the form window.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	Deactivate (see page 2117)	Generates an OnDeactivate (see page 2209) event.
	DefineProperties (see page 2118)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DestroyHandle (see page 2119)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 2119)	Destroys the window created in the CreateWindowHandle (see page 2116) method.
	DoAddDockClient (see page 2097)	Performs actions when a control is docked to the windowed control.
	DoClose (see page 2120)	Form closing event dispatcher.
	DoCreate (see page 2120)	Form creation event dispatcher.
	DoDestroy (see page 2121)	Form destruction event dispatcher.
	DoDock (see page 2121)	Docking event dispatcher.
	DoFlipChildren (see page 2560)	Flips the control's children.
	DoHide (see page 2122)	Form hide event dispatcher.
	DoRemoveDockClient (see page 2098)	Performs actions when a control is undocked from the dock form.
	DoShow (see page 2122)	Form show event dispatcher.
	get_ActiveMDIChild (see page 2123)	This is get_ActiveMDIChild, a member of class TCustomForm.
	get_MDIChildCount (see page 2123)	This is get_MDIChildCount, a member of class TCustomForm.
	get_MDIChildren (see page 2123)	This is get_MDIChildren, a member of class TCustomForm.
	GetBorderIconStyles (see page 2123)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 2124)	This is GetBorderStyles, a member of class TCustomForm.
	GetClientRect (see page 2124)	Returns the value of the ClientRect property.
	GetFloating (see page 2125)	Returns the value of the Floating property.
	GetOwnerWindow (see page 2127)	This is GetOwnerWindow, a member of class TCustomForm.
	GetSiteInfo (see page 2098)	Returns docking information for the dock form.
	HandleCreateException (see page 2127)	Handles exceptions that are raised by the OnCreate (see page 2207) event handler.
	InitializeNewForm (see page 2128)	This is InitializeNewForm, a member of class TCustomForm.
	Loaded (see page 2099)	Specifies actions to perform when the dock form is loaded.
	Notification (see page 2130)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 2131)	Generates an OnPaint (see page 2233) event.
	PaintWindow (see page 2131)	Renders the image of the form.
	PaletteChanged (see page 2132)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	QueryInterface (see page 2132)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2133)	Reads the form's published properties, stored data, and owned components from a reader object.
	RequestAlign (see page 2134)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	Resizing (see page 2134)	Makes any required adjustments when the form changes size.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	SetChildOrder (see page 2135)	Changes the order in which Child appears when GetChildren (see page 2124) is called.
	SetParent (see page 2136)	Sets the Parent property.
	SetParentBiDiMode (see page 2136)	Sets the ParentBiDiMode property.
	UpdateActions (see page 2140)	Updates all actions associated with the form.
	UpdateWindowState (see page 2141)	Updates the WindowState (see page 2195) property to reflect the current state of the form.
	ValidateRename (see page 2141)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 2141)	Responds just before the form changes the value of its Visible (see page 2192) property.
	WndProc (see page 2142)	Provides specific message responses for the form.

Protected Properties

	Name	Description
R	ActiveMDIChild (see page 2146)	Specifies the MDI child that has focus.
	AlphaBlend (see page 2147)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2147)	Specifies the degree of translucency on a translucent form.
	BorderIcons (see page 2148)	Specifies which icons appear on the title bar of the form.
R	ClientHandle (see page 2154)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 2154)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 2155)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 2158)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2158)	Specifies the monitor on which the form appears.
	Height (see page 2164)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2166)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 2167)	Specifies the icon that appears when the form is minimized.
R	MDIChildCount (see page 2172)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 2173)	Provides indexed access to all MDI child forms.
	ObjectMenuItem (see page 2177)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 2178)	Specifies when OnCreate (see page 2207) and OnDestroy (see page 2209) events occur.
	ParentFont (see page 2179)	Determines where a control looks for its font information.
R	PopupChildren (see page 2179)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 2180)	Identifies the pop-up menu associated with the control.
	Position (see page 2185)	Represents the size and placement of the form.
	PrintScale (see page 2186)	Represents the proportions of a printed form.
	Scaled (see page 2187)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 2179) property.
	TileMode (see page 2188)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 2190)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2191)	Indicates the color on the form that appears transparent when TransparentColor (see page 2190) is true.
	VertScrollBar (see page 2191)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 2192)	Indicates whether the form is visible.
	Width (see page 2193)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 2194)	Specifies the Window menu for an MDI parent form.

Public Methods

	Name	Description
V	AfterConstruction (see page 2112)	Construction event dispatcher.
V	BeforeDestruction (see page 2113)	Destruction event dispatcher.
	Close (see page 2114)	Closes the form.
V	CloseQuery (see page 2115)	Close (see page 2114) attempt event dispatcher.
V	Create (see page 2097)	Creates an instance of TCustomDockForm (see page 2092).
V	CreateNew (see page 2108)	Creates and initializes a new form.
V	DefaultHandler (see page 2118)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 2118)	Removes focus from a control on the form.
V	Destroy (see page 2109)	Removes the form from memory.
V	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
V	Dock (see page 2120)	Handles automatic docking of the form into other windows.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	FocusControl (see page 2122)	Sets focus to a control on the form.
V	GetChildren (see page 2124)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetFormImage (see page 2125)	Returns a bitmap of the form.

	Hide (see page 2127)	Hides the form.
	IsShortCut (see page 2128)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 2129)	Ensures that the form is fully visible on a specified monitor.
V	MouseWheelHandler (see page 2130)	Directs mouse wheel messages to the appropriate control.
	Print (see page 2132)	Prints the form.
	RecreateAsPopup (see page 2133)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 2133)	Destroys the form and frees its associated memory.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 2134)	Cancels modes on the form.
	set_PopupParent (see page 2135)	This is set_PopupParent, a member of class TCustomForm.
V	SetFocus (see page 2135)	Sets focus to the form.
V	SetFocusedControl (see page 2136)	Sets focus to a control on the form.
	Show (see page 2137)	Shows the form.
V	ShowModal (see page 2139)	Shows a form as a modal dialog.
V	WantChildKey (see page 2142)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 2144)	Designates the action associated with the control.
R	Active (see page 2145)	Specifies whether the form has focus.
	ActiveControl (see page 2145)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 2146)	Specifies the OLE control on the form that reacts to changes in focus.
	AutoScroll (see page 2099)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderStyle (see page 2100)	Specifies the appearance and behavior of the form border.
R	Canvas (see page 2150)	Provides access to the drawing area of the form.
	Caption (see page 2151)	Specifies a text string that identifies the control to the user.
	Color (see page 2157)	Specifies the background color of the control.
	Designer (see page 2159)	Specifies the designer interface for the form.
	DropTarget (see page 2159)	Specifies whether the form is the target of a drag-and-drop operation.
	Font (see page 2160)	Controls the attributes of text written on or in the control.
R	FormState (see page 2161)	Indicates transitional state information about the form.
	FormStyle (see page 2100)	Determines the form's style.
	GlassFrame (see page 2164)	This is GlassFrame, a member of class TCustomForm.
	HelpFile (see page 2165)	Specifies the name of the file the form uses to display Help.
	KeyPreview (see page 2167)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 2174)	Specifies the form's main menu.
	ModalResult (see page 2176)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 2177)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 2178)	Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.
	PopupMode (see page 2185)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 2185)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 2187)	Specifies whether form snaps to edge of screen.
	SnapBuffer (see page 2188)	Specifies distance for screen snap.
	WindowState (see page 2195)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 2171)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	PixelsPerInch (see page 2103)	Represents the proportion of the font on the system on which the form was designed.
	Top (see page 2189)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.

7.1.4.2 TCustomDockForm.Create Constructor

Creates an instance of TCustomDockForm (see page 2092).

Pascal

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

C++

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

Description

Typically, applications do not create instances of TCustomDockForm (see page 2092). Instead, this class is instantiated automatically as needed by controls that specify it as the value of their FloatingDockSiteClass property.

AOwner is the dock form's owner.

See Also

FloatingDockSiteClass

Floating

7.1.4.3 TCustomDockForm Methods

The methods of the TCustomDockForm class are listed here.

Protected Methods

	Name	Description
	DoAddDockClient (see page 2097)	Performs actions when a control is docked to the windowed control.
	DoRemoveDockClient (see page 2098)	Performs actions when a control is undocked from the dock form.
	GetSiteInfo (see page 2098)	Returns docking information for the dock form.
	Loaded (see page 2099)	Specifies actions to perform when the dock form is loaded.

7.1.4.3.1 TCustomDockForm.DoAddDockClient Method

Performs actions when a control is docked to the windowed control.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient is called automatically when a client control is docked to the windowed control. As implemented in TWinControl, DoAddDockClient sets the client's Parent property to the control.

Client is the control that has been docked.

ARect describes the boundaries of the region where Client has been docked.

Override DoAddDockClient to perform actions when a control is docked to the windowed control.

Note: DoAddDockClient is called in response to a message sent by the DockDrop method, before the OnDockDrop event.

See Also

Parent

DockClients

ManualDock

DoRemoveDockClient

DockDrop

OnDockDrop

7.1.4.3.2 TCustomDockForm.DoRemoveDockClient Method

Performs actions when a control is undocked from the dock form.

Pascal

```
procedure DoRemoveDockClient(Client: TControl); override;
```

C++

```
virtual __fastcall DoRemoveDockClient(TControl * Client);
```

Description

Client is the control being undocked.

DoRemoveDockClient calls the inherited DoRemoveDockClient method. Then, if the dock form contains no more clients, DoRemoveDockClient calls the Release (see page 2133) method.

See Also

Release (see page 2133)

doadddockclient (see page 2097)

7.1.4.3.3 TCustomDockForm.GetSiteInfo Method

Returns docking information for the dock form.

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

GetSiteInfo overrides the inherited method, setting CanDock to true if the dock form contains one or more clients.

See Also

DockClientCount

7.1.4.3.4 TCustomDockForm.Loaded Method

Specifies actions to perform when the dock form is loaded.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded calls the inherited method. Then Loaded sets the dock form's active control, if possible.

7.1.4.4 TCustomDockForm Properties

The properties of the TCustomDockForm class are listed here.

Public Properties

	Name	Description
	AutoScroll (see page 2099)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderStyle (see page 2100)	Specifies the appearance and behavior of the form border.
	FormStyle (see page 2100)	Determines the form's style.

Published Properties

	Name	Description
	PixelsPerInch (see page 2103)	Represents the proportion of the font on the system on which the form was designed.

7.1.4.4.1 TCustomDockForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.4.4.2 TCustomDockForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 2148)

7.1.4.4.3 TCustomDockForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.

fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example ensures that the main form of the application
is an MDI parent:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  if(FormStyle != fsMDIForm)
          FormStyle = fsMDIForm;
  if(FormStyle == fsMDIForm)
          Edit1->Text = "MDI form";
  else
    //This line never executes
    Edit1->Text = "Not an MDI form";
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example ensures that the main form of the application
is an MDI parent:
}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

constructor TForm1.Create(AOwner : TComponent);

begin
  inherited Create(AOwner);
  if FormStyle <> fsMDIForm then
          FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
          Edit1.Text := 'MDI form'
  else
    //This line never executes
    Edit1.Text := 'Not an MDI form';
end;
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

ArrangeIcons (see page 2247)

Tile (see page 2249)

Cascade (see page 2248)

Windows Overview

7.1.4.4.4 TCustomDockForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 2187)

7.1.5 TCustomForm Class

TCustomForm is the base class from which to derive a window such as a form or dialog.

Class Hierarchy

Pascal

```
TCustomForm = class(TScrollingWinControl);
```

C++

```
class TCustomForm : public TScrollingWinControl;
```

File

Forms

Description

Derive from TCustomForm to create a custom window. The TCustomForm descendant can contain other objects, such as TButton, TCheckBox, and TComboBox objects.

See Also

TPropertyPage

TActiveForm

TForm (see page 2241)

7.1.5.1 TCustomForm Members

The following tables list the members exposed by TCustomForm.

Protected Events

	Name	Description
	OnActivate (see page 2196)	Occurs when the form becomes active.
	OnCanResize (see page 2202)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2203)	Occurs when the user clicks the control.
	OnClose (see page 2205)	Occurs when the form closes.
	OnCloseQuery (see page 2206)	Occurs when close is attempted.
	OnCreate (see page 2207)	Occurs when the form is created.
	OnDblClick (see page 2209)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2209)	Occurs when the form loses focus.
	OnDestroy (see page 2209)	Occurs when the form is destroyed.
	OnDragDrop (see page 2211)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2214)	Occurs when the user drags an object over a control.
	OnHelp (see page 2217)	Occurs when the form receives a request for help.
	OnHide (see page 2217)	Occurs when the form is hidden (that is, when its Visible (see page 2192) property is set to false).
	OnKeyDown (see page 2219)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2223)	Occurs when a key is pressed.
	OnKeyUp (see page 2224)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2226)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2227)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 2229)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2231)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 2233)	Occurs when the form is redrawn.
	OnResize (see page 2235)	Occurs immediately after the control is resized.
	OnShortCut (see page 2235)	Occurs when the user presses a key (before the OnKeyDown (see page 2219) event).
	OnShow (see page 2236)	Occurs when the form is shown (that is, when its Visible (see page 2192) property is set to true).

Protected Fields

	Name	Description
	FFormState (see page 2110)	This is FFormState, a member of class TCustomForm.

Protected Methods

	Name	Description
	Activate (see page 2111)	Activation event dispatcher.
	ActiveChanged (see page 2112)	Active (see page 2145) state event dispatcher.
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2113)	Aligns all child controls of the form that fall within a specified rectangle.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 2113)	Responds when the user starts dragging the form if DragMode is dmAutomatic.

	ChangeScale (see page 2114)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	CloseModal (see page 2115)	This is CloseModal, a member of class TCustomForm.
	CreateParams (see page 2116)	Initializes the window-creation parameter record when the form window is created.
	CreateWindowHandle (see page 2116)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 2117)	Creates the form window.
	Deactivate (see page 2117)	Generates an OnDeactivate (see page 2209) event.
	DefineProperties (see page 2118)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DestroyHandle (see page 2119)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 2119)	Destroys the window created in the CreateWindowHandle (see page 2116) method.
	DoClose (see page 2120)	Form closing event dispatcher.
	DoCreate (see page 2120)	Form creation event dispatcher.
	DoDestroy (see page 2121)	Form destruction event dispatcher.
	DoDock (see page 2121)	Docking event dispatcher.
	DoFlipChildren (see page 2560)	Flips the control's children.
	DoHide (see page 2122)	Form hide event dispatcher.
	DoShow (see page 2122)	Form show event dispatcher.
	get_ActiveMDIChild (see page 2123)	This is get_ActiveMDIChild, a member of class TCustomForm.
	get_MDIChildCount (see page 2123)	This is get_MDIChildCount, a member of class TCustomForm.
	get_MDIChildren (see page 2123)	This is get_MDIChildren, a member of class TCustomForm.
	GetBorderIconStyles (see page 2123)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 2124)	This is GetBorderStyles, a member of class TCustomForm.
	GetClientRect (see page 2124)	Returns the value of the ClientRect property.
	GetFloating (see page 2125)	Returns the value of the Floating property.
	GetOwnerWindow (see page 2127)	This is GetOwnerWindow, a member of class TCustomForm.
	HandleCreateException (see page 2127)	Handles exceptions that are raised by the OnCreate (see page 2207) event handler.
	InitializeNewForm (see page 2128)	This is InitializeNewForm, a member of class TCustomForm.
	Loaded (see page 2129)	Performs fix-ups when the form is first loaded into memory.
	Notification (see page 2130)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 2131)	Generates an OnPaint (see page 2233) event.
	PaintWindow (see page 2131)	Renders the image of the form.
	PaletteChanged (see page 2132)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	QueryInterface (see page 2132)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2133)	Reads the form's published properties, stored data, and owned components from a reader object.
	RequestAlign (see page 2134)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	Resizing (see page 2134)	Makes any required adjustments when the form changes size.
	SetChildOrder (see page 2135)	Changes the order in which Child appears when GetChildren (see page 2124) is called.
	SetParent (see page 2136)	Sets the Parent property.
	SetParentBiDiMode (see page 2136)	Sets the ParentBiDiMode property.
	UpdateActions (see page 2140)	Updates all actions associated with the form.
	UpdateWindowState (see page 2141)	Updates the WindowState (see page 2195) property to reflect the current state of the form.
	ValidateRename (see page 2141)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 2141)	Responds just before the form changes the value of its Visible (see page 2192) property.
	WndProc (see page 2142)	Provides specific message responses for the form.

Protected Properties

	Name	Description
R	ActiveMDIChild (see page 2146)	Specifies the MDI child that has focus.
	AlphaBlend (see page 2147)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2147)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 2148)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderIcons (see page 2148)	Specifies which icons appear on the title bar of the form.
R	ClientHandle (see page 2154)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 2154)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 2155)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 2158)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2158)	Specifies the monitor on which the form appears.
	FormStyle (see page 2162)	Determines the form's style.
	Height (see page 2164)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2166)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 2167)	Specifies the icon that appears when the form is minimized.
R	MDIChildCount (see page 2172)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 2173)	Provides indexed access to all MDI child forms.
	ObjectMenuItem (see page 2177)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 2178)	Specifies when OnCreate (see page 2207) and OnDestroy (see page 2209) events occur.
	ParentFont (see page 2179)	Determines where a control looks for its font information.
	PixelsPerInch (see page 2179)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 2179)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 2180)	Identifies the pop-up menu associated with the control.
	Position (see page 2185)	Represents the size and placement of the form.
	PrintScale (see page 2186)	Represents the proportions of a printed form.
	Scaled (see page 2187)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 2179) property.
	TileMode (see page 2188)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 2190)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2191)	Indicates the color on the form that appears transparent when TransparentColor (see page 2190) is true.
	VertScrollBar (see page 2191)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 2192)	Indicates whether the form is visible.
	Width (see page 2193)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 2194)	Specifies the Window menu for an MDI parent form.

Public Methods

	Name	Description
V	AfterConstruction (see page 2112)	Construction event dispatcher.
V	BeforeDestruction (see page 2113)	Destruction event dispatcher.
	Close (see page 2114)	Closes the form.
V	CloseQuery (see page 2115)	Close (see page 2114) attempt event dispatcher.
V	Create (see page 2108)	Creates and initializes a new TForm (see page 2241) object.
V	CreateNew (see page 2108)	Creates and initializes a new form.
V	DefaultHandler (see page 2118)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 2118)	Removes focus from a control on the form.
V	Destroy (see page 2109)	Removes the form from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
V	Dock (see page 2120)	Handles automatic docking of the form into other windows.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	FocusControl (see page 2122)	Sets focus to a control on the form.

	GetChildren (see page 2124)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetFormImage (see page 2125)	Returns a bitmap of the form.
	Hide (see page 2127)	Hides the form.
	IsShortCut (see page 2128)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 2129)	Ensures that the form is fully visible on a specified monitor.
	MouseWheelHandler (see page 2130)	Directs mouse wheel messages to the appropriate control.
	Print (see page 2132)	Prints the form.
	RecreateAsPopup (see page 2133)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 2133)	Destroys the form and frees its associated memory.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 2134)	Cancels modes on the form.
	set_PopupParent (see page 2135)	This is set_PopupParent, a member of class TCustomForm.
	SetFocus (see page 2135)	Sets focus to the form.
	SetFocusedControl (see page 2136)	Sets focus to a control on the form.
	Show (see page 2137)	Shows the form.
	ShowModal (see page 2139)	Shows a form as a modal dialog.
	WantChildKey (see page 2142)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 2144)	Designates the action associated with the control.
R	Active (see page 2145)	Specifies whether the form has focus.
	ActiveControl (see page 2145)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 2146)	Specifies the OLE control on the form that reacts to changes in focus.
	BorderStyle (see page 2149)	Specifies the appearance and behavior of the form border.
R	Canvas (see page 2150)	Provides access to the drawing area of the form.
	Caption (see page 2151)	Specifies a text string that identifies the control to the user.
	Color (see page 2157)	Specifies the background color of the control.
	Designer (see page 2159)	Specifies the designer interface for the form.
	DropTarget (see page 2159)	Specifies whether the form is the target of a drag-and-drop operation.
	Font (see page 2160)	Controls the attributes of text written on or in the control.
R	FormState (see page 2161)	Indicates transitional state information about the form.
	GlassFrame (see page 2164)	This is GlassFrame, a member of class TCustomForm.
	HelpFile (see page 2165)	Specifies the name of the file the form uses to display Help.
	KeyPreview (see page 2167)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 2174)	Specifies the form's main menu.
	ModalResult (see page 2176)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 2177)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 2178)	Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.
	PopupMode (see page 2185)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 2185)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 2187)	Specifies whether form snaps to edge of screen.
	SnapBuffer (see page 2188)	Specifies distance for screen snap.
	WindowState (see page 2195)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	Left (see page 2171)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Top (see page 2189)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

7.1.5.2 TCustomForm.Create Constructor

Creates and initializes a new TForm (see page 2241) object.

Pascal

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

C++

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

Description

Use Create to instantiate a TForm (see page 2241) object or descendant of TForm (see page 2241) at runtime. AOwner is the owner of the TCustomForm (see page 2103) object.

For descendants of TCustomForm (see page 2103) that are not TForm (see page 2241) objects, use CreateNew (see page 2108) instead. Calling Create for a TCustomForm (see page 2103) descendant that is not a TForm (see page 2241) will raise an exception.

See Also

CreateNew (see page 2108)

TForm (see page 2241)

Owner

7.1.5.3 TCustomForm.CreateNew Constructor

Creates and initializes a new form.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); virtual;
```

C++

```
virtual __fastcall TCustomForm(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 2108) to create a form without using the associated .DFM file to initialize it. Always use CreateNew if the TCustomForm (see page 2103) descendant is not a TForm (see page 2241) object or a descendant of TForm (see page 2241).

CreateNew bypasses the streaming in of the previously-associated .DFM file. If the form contains visual components, therefore, you must stream in an external .DFM to bind the visual components with their classes. If the newly created form has an external .DFM file, then you can follow the call to CreateNew with a call to InitInheritedComponent. If you need to create the .dfm file for the new form instance, bracket the call to CreateNew with calls to WriteComponentResFile and ReadComponentResFile. The following code sequence

1. streams out an external .DFM file;2. creates a new form disassociated from any .DFM resource file;3. streams in the external .DFM file and binds it to this new form.

```
WriteComponentResFile('Temp.dfm', Form1);
...
```

Form2 := TForm.CreateNew(Application (see page 2612));

```
ReadComponentResFile('Temp.dfm', Form2);
```

Warning: Using CreateNew instead of Create (see page 2108) can cause unpredictable results because most forms are written assuming their controls will be created from the .DFM file.

CreateNew is called by the Create (see page 2108) method.

The first argument, AOwner, is the owner of the form (usually the application or another form is the owner). The second argument, Dummy, defaults to 0 and is not used by this constructor. The system passes 0 by default if this constructor is passed only the first argument, AOwner.

See Also

Create (see page 2108)

Owner

ReadComponentResFile

WriteComponentResFile

7.1.5.4 TCustomForm.Destroy Destructor

Removes the form from memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomForm();
```

Description

Do not call Destroy. Instead, call Release (see page 2133), which indirectly produces a call to the Free method, which in turn calls Destroy.

See Also

OnDestroy (see page 2209)

Release (see page 2133)

System::TObject::Free

7.1.5.5 TCustomForm Fields

The fields of the TCustomForm class are listed here.

Protected Fields

	Name	Description
	FFormState (see page 2110)	This is FFormState, a member of class TCustomForm.

7.1.5.5.1 TCustomForm.FFormState Field

Pascal

```
FFormState: TFormState;
```

C++

```
TFormState FFormState;
```

Description

This is FFormState, a member of class TCustomForm.

7.1.5.6 TCustomForm Methods

The methods of the TCustomForm class are listed here.

Protected Methods

	Name	Description
	Activate (see page 2111)	Activation event dispatcher.
	ActiveChanged (see page 2112)	Active (see page 2145) state event dispatcher.
V	AlignControls (see page 2113)	Aligns all child controls of the form that fall within a specified rectangle.
V	BeginAutoDrag (see page 2113)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
V	ChangeScale (see page 2114)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	CloseModal (see page 2115)	This is CloseModal, a member of class TCustomForm.
V	CreateParams (see page 2116)	Initializes the window-creation parameter record when the form window is created.
V	CreateWindowHandle (see page 2116)	Creates a form window given a set of window-creation parameters.
V	CreateWnd (see page 2117)	Creates the form window.
	Deactivate (see page 2117)	Generates an OnDeactivate (see page 2209) event.
V	DefineProperties (see page 2118)	Designates methods for storing a form's unpublished data on a stream such as a form file.
V	DestroyHandle (see page 2119)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 2119)	Destroys the window created in the CreateWindowHandle (see page 2116) method.
	DoClose (see page 2120)	Form closing event dispatcher.
V	DoCreate (see page 2120)	Form creation event dispatcher.
V	DoDestroy (see page 2121)	Form destruction event dispatcher.
V	DoDock (see page 2121)	Docking event dispatcher.
	DoHide (see page 2122)	Form hide event dispatcher.
	DoShow (see page 2122)	Form show event dispatcher.
	get_ActiveMDIChild (see page 2123)	This is get_ActiveMDIChild, a member of class TCustomForm.
	get_MDIChildCount (see page 2123)	This is get_MDIChildCount, a member of class TCustomForm.
	get_MDIChildren (see page 2123)	This is get_MDIChildren, a member of class TCustomForm.
	GetBorderIconStyles (see page 2123)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 2124)	This is GetBorderStyles, a member of class TCustomForm.
V	GetClientRect (see page 2124)	Returns the value of the ClientRect property.
V	GetFloating (see page 2125)	Returns the value of the Floating property.
	GetOwnerWindow (see page 2127)	This is GetOwnerWindow, a member of class TCustomForm.
	HandleCreateException (see page 2127)	Handles exceptions that are raised by the OnCreate (see page 2207) event handler.
	InitializeNewForm (see page 2128)	This is InitializeNewForm, a member of class TCustomForm.
V	Loaded (see page 2129)	Performs fix-ups when the form is first loaded into memory.
V	Notification (see page 2130)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 2131)	Generates an OnPaint (see page 2233) event.

	PaintWindow (see page 2131)	Renders the image of the form.
	PaletteChanged (see page 2132)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	QueryInterface (see page 2132)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2133)	Reads the form's published properties, stored data, and owned components from a reader object.
	RequestAlign (see page 2134)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	Resizing (see page 2134)	Makes any required adjustments when the form changes size.
	SetChildOrder (see page 2135)	Changes the order in which Child appears when GetChildren (see page 2124) is called.
	SetParent (see page 2136)	Sets the Parent property.
	SetParentBiDiMode (see page 2136)	Sets the ParentBiDiMode property.
	UpdateActions (see page 2140)	Updates all actions associated with the form.
	UpdateWindowState (see page 2141)	Updates the WindowState (see page 2195) property to reflect the current state of the form.
	ValidateRename (see page 2141)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 2141)	Responds just before the form changes the value of its Visible (see page 2192) property.
	WndProc (see page 2142)	Provides specific message responses for the form.

Public Methods

	Name	Description
	AfterConstruction (see page 2112)	Construction event dispatcher.
	BeforeDestruction (see page 2113)	Destruction event dispatcher.
	Close (see page 2114)	Closes the form.
	CloseQuery (see page 2115)	Close (see page 2114) attempt event dispatcher.
	DefaultHandler (see page 2118)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 2118)	Removes focus from a control on the form.
	Dock (see page 2120)	Handles automatic docking of the form into other windows.
	FocusControl (see page 2122)	Sets focus to a control on the form.
	GetChildren (see page 2124)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetFormImage (see page 2125)	Returns a bitmap of the form.
	Hide (see page 2127)	Hides the form.
	IsShortCut (see page 2128)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 2129)	Ensures that the form is fully visible on a specified monitor.
	MouseWheelHandler (see page 2130)	Directs mouse wheel messages to the appropriate control.
	Print (see page 2132)	Prints the form.
	RecreateAsPopup (see page 2133)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 2133)	Destroys the form and frees its associated memory.
	SendCancelMode (see page 2134)	Cancels modes on the form.
	set_PopupParent (see page 2135)	This is set_PopupParent, a member of class TCustomForm.
	SetFocus (see page 2135)	Sets focus to the form.
	SetFocusedControl (see page 2136)	Sets focus to a control on the form.
	Show (see page 2137)	Shows the form.
	ShowModal (see page 2139)	Shows a form as a modal dialog.
	WantChildKey (see page 2142)	Indicates whether the form processes keyboard input for a control it owns.

7.1.5.6.1 TCustomForm.Activate Method

Activation event dispatcher.

Pascal

```
procedure Activate; dynamic;
```

C++

```
__fastcall Activate();
```

Description

Activation is the event dispatcher for form activations.

The form component calls Activate in response to the CM_ACTIVATE message, sent when a form gets the focus either because of a change in active windows within an application or because of the application becoming active. As defined in TCustomForm (see page 2103), Activate simply calls the OnActivate (see page 2196) event handler.

See Also

Deactivate (see page 2117)

OnActivate (see page 2196)

7.1.5.6.2 TCustomForm.ActiveChanged Method

Active (see page 2145) state event dispatcher.

Pascal

```
procedure ActiveChanged; dynamic;
```

C++

```
__fastcall ActiveChanged();
```

Description

ActiveChanged is the event dispatcher for changes in the form's active status. As implemented in TCustomForm (see page 2103), ActiveChanged does nothing. Override this method in a descendant class to implement event handling or other response to a change in the active form.

See Also

ActiveControl (see page 2145)

7.1.5.6.3 TCustomForm.AfterConstruction Method

Construction event dispatcher.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is the dispatcher for the event that occurs immediately after the form object's construction. As implemented in TCustomForm (see page 2103), AfterConstruction calls the OnCreate (see page 2207) event handler, provided the OldCreateOrder (see page 2178) property is false.

See Also

BeforeDestruction (see page 2113)

OnCreate (see page 2207)

OldCreateOrder (see page 2178)

7.1.5.6.4 TCustomForm.AlignControls Method

Aligns all child controls of the form that fall within a specified rectangle.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of the form. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly aligned controls.

See Also

Align

7.1.5.6.5 TCustomForm.BeforeDestruction Method

Destruction event dispatcher.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is the dispatcher for the event that occurs immediately before the form object's destruction. As implemented in TCustomForm (see page 2103), BeforeDestruction calls the OnDestroy (see page 2209) event handler, provided the OldCreateOrder (see page 2178) property is false.

See Also

AfterConstruction (see page 2112)

OnDestroy (see page 2209)

7.1.5.6.6 TCustomForm.BeginAutoDrag Method

Responds when the user starts dragging the form if DragMode is dmAutomatic.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag suppresses the default behavior of the BeginAutoDrag method. This is because forms do not change the mouse cursor when dragged.

See Also

DragMode

Floating

7.1.5.6.7 TCustomForm.ChangeScale Method

Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

TCustomForm (see page 2103) overrides ChangeScale to rescale any scroll bars and child controls.

See Also

ChangeScale

7.1.5.6.8 TCustomForm.Close Method

Closes the form.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to close a form.

Attempts to close a form can be aborted by the CloseQuery (see page 2115) event dispatcher or the OnCloseQuery (see page 2206) event handler.

Note: When the main form of the application closes, the application terminates.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
*/
void __fastcall TForm1::CloseChildrenClick(TObject *Sender)
{
  for(int i = MDIChildCount - 1; i >= 0; i--)
    MDIChildren[i]->Close();
}
```

```
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}

void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
  }
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

CloseQuery (see page 2115)

OnCloseQuery (see page 2206)

7.1.5.6.9 TCustomForm.CloseModal Method

Pascal

```
procedure CloseModal;
```

C++

```
__fastcall CloseModal();
```

Description

This is CloseModal, a member of class TCustomForm.

7.1.5.6.10 TCustomForm.CloseQuery Method

Close (see page 2114) attempt event dispatcher.

Pascal

```
function CloseQuery: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CloseQuery();
```

Description

CloseQuery is called automatically when an attempt is made to close the form. CloseQuery can allow the form to close by returning true, or prevent the form from closing by returning false.

As implemented in TCustomForm (see page 2103), CloseQuery polls any MDI children by calling their CloseQuery methods. If no child form aborts the close, CloseQuery then calls the OnCloseQuery (see page 2206) event handler, if it exists, to determine if the close should be allowed. If no such event handler exists, CloseQuery returns true.

See Also

Close (see page 2114)

OnCloseQuery (see page 2206)

7.1.5.6.11 TCustomForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2117) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 2116). TCustomForm (see page 2103) overrides CreateParams to customize the way the form creates its Windows representation after calling the inherited method. You should never need to override this method in the context of a form class.

The fields of the Params parameter become the parameters to a call to the CreateWindowEx OS function.

See Also

CreateWindowHandle (see page 2116)

CreateWnd (see page 2117)

7.1.5.6.12 TCustomForm.CreateWindowHandle Method

Creates a form window given a set of window-creation parameters.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 2117) method calls CreateWindowHandle to create the form window once it has been specified in the window-creation parameters. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. CreateWindowHandle removes the CS_HREDRAW and

CS_VREDRAW class styles from the window class.

Params holds information needed when telling Windows to create a window handle.

See Also

CreateParams (see page 2116)

CreateWnd (see page 2117)

7.1.5.6.13 TCustomForm.CreateWnd Method

Creates the form window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the form is created, and whenever a change to the form requires the window to be recreated. CreateWnd calls (indirectly) the CreateParams (see page 2116) method to obtain the window creation parameters, and then calls CreateWindowHandle (see page 2116) to direct Windows to create the underlying window for the form. It then initializes the newly-created window to reflect the property settings of the form.

See Also

CreateParams (see page 2116)

CreateWindowHandle (see page 2116)

Handle

7.1.5.6.14 TCustomForm.Deactivate Method

Generates an OnDeactivate (see page 2209) event.

Pascal

```
procedure Deactivate; dynamic;
```

C++

```
__fastcall Deactivate();
```

Description

The form component calls Deactivate in response to the CM_DEACTIVATE message, sent when a form gets the focus either because of a change in active windows within an application or because of the application becoming active.

The default Deactivate method calls any event handler attached by the user to the OnDeactivate (see page 2209) event. Override this method to block the OnDeactivate (see page 2209) event or provide additional responses to the loss of activation.

See Also

Activate (see page 2111)

OnActivate (see page 2196)

7.1.5.6.15 TCustomForm.DefaultHandler Method

Provides message handling for all messages that the form does not fully process.

Pascal

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

C++

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

Description

Override DefaultHandler to change the default message handling for the form. This is seldom necessary because messages can be handled by creating message methods.

DefaultHandler passes any otherwise-unhandled messages to the form's window procedure by calling the CallWindowProc OS function.

Note: In Delphi code, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method unless that ancestor specifies a handler for the message.

See Also

System::TObject::Dispatch

7.1.5.6.16 TCustomForm.DefineProperties Method

Designates methods for storing a form's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties calls the inherited method and calls the filer object's DefineProperty method for the PixelsPerInch (see page 2179),and TextHeight properties. TForm (see page 2241) descendants can also specify methods that read and write data other than properties by overriding the DefineProperties method.

See Also

DefineProperty

DefineBinaryProperty

7.1.5.6.17 TCustomForm.DefocusControl Method

Removes focus from a control on the form.

Pascal

```
procedure DefocusControl(Control: TWinControl; Removing: Boolean);
```

C++

```
__fastcall DefocusControl(TWinControl * Control, Boolean Removing);
```

Description

Use DefocusControl to remove focus from a control on the form. This method is used internally by the VCL.

Control is the control from which to remove focus. If Control is the form's active control, DefocusControl sets the form's ActiveControl (see page 2145) property to nil (Delphi) or NULL (C++).

Removing specifies whether to set focus to the control's parent.

See Also

SetFocusedControl (see page 2136)

ActiveControl (see page 2145)

SetFocus (see page 2135)

FocusedControl

7.1.5.6.18 TCustomForm.DestroyHandle Method

Destroys the control's window without destroying the control.

Pascal

```
procedure DestroyHandle; override;
```

C++

```
virtual __fastcall DestroyHandle();
```

Description

Call DestroyHandle to dispose of the window, but leave the control intact. The control can later recreate the window if needed. DestroyHandle is the converse operation to CreateHandle. Applications should call the high-level CreateHandle and DestroyHandle methods, rather than the lower-level methods of CreateWnd and DestroyWnd, whenever possible.

If the control has TWinControl objects as child controls, DestroyHandle calls each of their DestroyHandle methods before calling DestroyWnd to destroy its own handle.

See Also

CreateHandle

DestroyWnd

7.1.5.6.19 TCustomForm.DestroyWindowHandle Method

Destroys the window created in the CreateWindowHandle (see page 2116) method.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

DestroyWindowHandle is called automatically to destroy the window associated with a form before destroying the form.

See Also

CreateWindowHandle (see page 2116)

DestroyWnd

Destroy (see page 2109)

7.1.5.6.20 TCustomForm.Dock Method

Handles automatic docking of the form into other windows.

Pascal

```
procedure Dock(NewDockSite: TWinControl; ARect: TRect); override;
```

C++

```
virtual __fastcall Dock(TWinControl * NewDockSite, TRect ARect);
```

Description

Do not call Dock in application code. It is called automatically when the form is released on a docking site that will accept it. To dock a form programmatically, use the ManualDock method instead.

See Also

DragMode

ManualDock

7.1.5.6.21 TCustomForm.DoClose Method

Form closing event dispatcher.

Pascal

```
procedure DoClose(var Action: TCloseAction); dynamic;
```

C++

```
__fastcall DoClose(TCloseAction Action);
```

Description

DoClose is called automatically when the form is about to close. DoClose can alter closing behavior by changing the Action (see page 2144) parameter. For more information, refer to the OnClose (see page 2205) event handler.

As implemented in TCustomForm (see page 2103), DoClose simply calls the OnClose (see page 2205) event handler.

See Also

OnClose (see page 2205)

7.1.5.6.22 TCustomForm.DoCreate Method

Form creation event dispatcher.

Pascal

```
procedure DoCreate; virtual;
```

C++

```
virtual __fastcall DoCreate();
```

Description

DoCreate is called automatically after form construction is complete. The precise way DoCreate is called can be modified by setting the OldCreateOrder (see page 2178) property.

As implemented in TCustomForm (see page 2103), DoCreate simply calls the OnCreate (see page 2207) event handler.

See Also

OnCreate (see page 2207)

FormState (see page 2161)

Visible

OldCreateOrder (see page 2178)

7.1.5.6.23 TCustomForm.DoDestroy Method

Form destruction event dispatcher.

Pascal

```
procedure DoDestroy; virtual;
```

C++

```
virtual __fastcall DoDestroy();
```

Description

DoDestroy is called automatically when the TCustomForm (see page 2103) object is about to be deallocated. The precise way DoDestroy is called can be modified by setting the OldCreateOrder (see page 2178) property.

As implemented in TCustomForm (see page 2103), DoDestroy simply calls the OnDestroy (see page 2209) event handler.

See Also

OnDestroy (see page 2209)

OldCreateOrder (see page 2178)

7.1.5.6.24 TCustomForm.DoDock Method

Docking event dispatcher.

Pascal

```
procedure DoDock(NewDockSite: TWinControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall DoDock(TWinControl * NewDockSite, TRect ARect);
```

Description

DoDock supplements the inherited DoDock method by updating form borders.

See Also

DoDock

Floating

Dock (see page 2120)

ManualDock

ManualFloat

7.1.5.6.25 TCustomForm.DoHide Method

Form hide event dispatcher.

Pascal

```
procedure DoHide; dynamic;
```

C++

```
__fastcall DoHide();
```

Description

DoHide is called automatically when the form is hidden.

As implemented in TCustomForm (see page 2103), DoHide calls the OnHide (see page 2217) event handler.

See Also

OnHide (see page 2217)

DoShow (see page 2122)

Hide (see page 2127)

7.1.5.6.26 TCustomForm.DoShow Method

Form show event dispatcher.

Pascal

```
procedure DoShow; dynamic;
```

C++

```
__fastcall DoShow();
```

Description

DoShow is called automatically when the application is about to show the form. As implemented in TCustomForm (see page 2103), DoShow simply calls the OnShow (see page 2236) event handler.

See Also

DoHide (see page 2122)

OnShow (see page 2236)

7.1.5.6.27 TCustomForm.FocusControl Method

Sets focus to a control on the form.

Pascal

```
procedure FocusControl(Control: TWinControl);
```

C++

```
__fastcall FocusControl(TWinControl * Control);
```

Description

Use FocusControl to set focus to a control on the form.

Control is a TWinControl object encapsulating the control on which to set focus. FocusControl sets the ActiveControl (see page 2145) property.

See Also

TWinControl

ActiveControl (see page 2145)

7.1.5.6.28 TCustomForm.get_ActiveMDIChild Method

Pascal

```
function get_ActiveMDIChild: TForm;
```

C++

```
__fastcall TForm * get_ActiveMDIChild();
```

Description

This is get_ActiveMDIChild, a member of class TCustomForm.

7.1.5.6.29 TCustomForm.get_MDIChildCount Method

Pascal

```
function get_MDIChildCount: Integer;
```

C++

```
__fastcall int get_MDIChildCount();
```

Description

This is get_MDIChildCount, a member of class TCustomForm.

7.1.5.6.30 TCustomForm.get_MDIChildren Method

Pascal

```
function get_MDIChildren(I: Integer): TForm;
```

C++

```
__fastcall TForm * get_MDIChildren(int I);
```

Description

This is get_MDIChildren, a member of class TCustomForm.

7.1.5.6.31 TCustomForm.GetBorderIconStyles Method

Pascal

```
procedure GetBorderIconStyles(var Style: Cardinal; var ExStyle: Cardinal); dynamic;
```

C++

```
__fastcall GetBorderIconStyles(unsigned Style, unsigned ExStyle);
```

Description

This is GetBorderIconStyles, a member of class TCustomForm.

7.1.5.6.32 TCustomForm.GetBorderStyles Method

Pascal

```
procedure GetBorderStyles(var Style: Cardinal; var ExStyle: Cardinal; var ClassStyle:
Cardinal); dynamic;
```

C++

```
__fastcall GetBorderStyles(unsigned Style, unsigned ExStyle, unsigned ClassStyle);
```

Description

This is GetBorderStyles, a member of class TCustomForm.

7.1.5.6.33 TCustomForm.GetChildren Method

Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

The component streaming system calls GetChildren to request that the form execute a callback for every child component in the form. The callback must execute in creation order (the order the child components appear in the form file). A child component is defined as any component that returns this form from its GetParentComponent method.

Proc is the callback to execute for every child component.

Root specifies the top-level component (such as a form or data module) that is currently being loaded or saved.

In TCustomForm (see page 2103), GetChildren first calls the inherited method, which executes Proc for every control that lists the form as its Parent. Then, if the Root parameter identifies this form, GetChildren executes Proc for every child component that does not have a Parent.

See Also

GetParentComponent

7.1.5.6.34 TCustomForm.GetClientRect Method

Returns the value of the ClientRect property.

Pascal

```
function GetClientRect: TRect; override;
```

C++

```
virtual __fastcall TRect GetClientRect();
```

Description

GetClientRect is the protected access function for the ClientRect property. It returns the dimensions of the form window, indented by any scroll bars and menus.

Override this method to change the way the client area of the form is defined.

C++ Examples:

```
/*
The following lines of code draw the top quarter of an arc
bounded by the current window:
*/
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TRect R = GetClientRect();  // Gets the rectangular
                              // coordinates of
                              // the current window.
  Canvas->Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
}
```

Delphi Examples:

```
{
The following lines of code draw the top quarter of an arc
bounded by the current window:
}
procedure TForm1.FormPaint(Sender: TObject);
var
  R: TRect;
begin
  R := GetClientRect;   { Gets the boundaries of the current window }
  Canvas.Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
end;
```

See Also

GetClientOrigin

ClientRect

7.1.5.6.35 TCustomForm.GetFloating Method

Returns the value of the Floating property.

Pascal

```
function GetFloating: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetFloating();
```

Description

GetFloating is the protected access function for the Floating property. Override this method in descendants that implement floating windows in some class-specific way.

See Also

Floating

7.1.5.6.36 TCustomForm.GetFormImage Method

Returns a bitmap of the form.

Pascal

```
function GetFormImage: TBitmap;
```

C++

```
__fastcall TBitmap GetFormImage();
```

Description

Use the GetFormImage method to obtain a bitmap of the form.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
```

```
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
```

See Also

Print (see page 2132)

7.1.5.6.37 TCustomForm.GetOwnerWindow Method

Pascal

```
function GetOwnerWindow: HWND; dynamic;
```

C++

```
__fastcall HWND GetOwnerWindow();
```

Description

This is GetOwnerWindow, a member of class TCustomForm.

7.1.5.6.38 TCustomForm.HandleCreateException Method

Handles exceptions that are raised by the OnCreate (see page 2207) event handler.

Pascal

```
function HandleCreateException: Boolean; dynamic;
```

C++

```
__fastcall Boolean HandleCreateException();
```

Description

The form calls HandleCreateException internally if the OnCreate (see page 2207) event handler raises an exception. HandleCreateException forwards the exception on to the application object, which generates an OnException event. If there is no OnException event handler, the application displays an exception message box.

HandleCreateException returns true if it handles the exception, false otherwise.

See Also

OnCreate (see page 2207)

OnException (see page 1993)

ShowException (see page 1961)

7.1.5.6.39 TCustomForm.Hide Method

Hides the form.

Pascal

```
procedure Hide;
```

C++

```
__fastcall Hide();
```

Description

Hide sets the Visible (see page 2192) property to false.

C++ Examples:

```
/*
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Timer1->Interval = StrToInt(MaskEdit1->Text);
  Timer1->Enabled = true;
  Hide();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Visible = true;
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Interval := StrtoInt(MaskEdit1.Text);
  Timer1.Enabled := True;
  Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Visible := True;
  Timer1.Enabled := False;
end;
```

See Also

Visible (see page 2192)

7.1.5.6.40 TCustomForm.InitializeNewForm Method

Pascal

```
procedure InitializeNewForm; dynamic;
```

C++

```
__fastcall InitializeNewForm();
```

Description

This is InitializeNewForm, a member of class TCustomForm.

7.1.5.6.41 TCustomForm.IsShortCut Method

Processes shortcut keys when the form has focus.

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean; dynamic;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

IsShortCut is called automatically when the user presses a key while the form has focus. It allows the form to dispatch shortcut keys and bypass the standard keystroke processing (OnKeyDown (see page 2219), OnKeyPress (see page 2223), and OnKeyUp (see page 2224) events).

The Message parameter encapsulates the Windows message describing the keypress. If a shortcut for the key exists, IsShortCut executes the appropriate command and returns true. Otherwise IsShortCut returns false.

IsShortCut generates an OnShortCut (see page 2235) event to allow the developer to insert shortcuts for the form. If the keystroke is not handled in an OnShortCut (see page 2235) event handler, IsShortCut checks for any shortcuts associated with the form's menus and actions.

See Also

OnShortCut (see page 2235)

OnKeyDown

OnKeyPress

OnKeyUp

7.1.5.6.42 TCustomForm.Loaded Method

Performs fix-ups when the form is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded overrides the inherited method to give focus to the ActiveControl (see page 2145) if it is assigned. Before assigning focus, Loaded calls the inherited method to perform any lower-level initialization.

See Also

ActiveControl (see page 2145)

Loaded

7.1.5.6.43 TCustomForm.MakeFullyVisible Method

Ensures that the form is fully visible on a specified monitor.

Pascal

```
procedure MakeFullyVisible(AMonitor: TMonitor = nil);
```

C++

```
__fastcall MakeFullyVisible(TMonitor AMonitor = nil);
```

Description

Call MakeFullyVisible to ensure that the form does not appear split over more than one monitor in a multi-monitor application.

AMonitor is the monitor on which the form should appear. If AMonitor is nil (Delphi) or NULL (C++), MakeFullyVisible uses the Monitor (see page 2177) property.

MakeFullyVisible checks whether the form fits entirely on the specified monitor. If not, it repositions the form so that it fits, if possible.

See Also

DefaultMonitor (see page 2158)

Monitor (see page 2177)

7.1.5.6.44 TCustomForm.MouseWheelHandler Method

Directs mouse wheel messages to the appropriate control.

Pascal

```
procedure MouseWheelHandler(var Message: TMessage); override;
```

C++

```
virtual __fastcall MouseWheelHandler(TMessage Message);
```

Description

MouseWheelHandler is called automatically when the form or any windowed control contained in the form receives a message from a mouse wheel. The Message parameter encapsulates the message.

As implemented in TCustomForm (see page 2103), MouseWheelHandler performs no special processing, but simply redirects the message to the active control by calling its Perform method. By default, the Perform method generates the appropriate mouse wheel events.

See Also

ActiveControl (see page 2145)

Perform

OnMouseWheel

OnMouseWheelUp

OnMouseWheelDown

7.1.5.6.45 TCustomForm.Notification Method

Responds to notifications that objects are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, forms pass along the notification to their owned components, if any. In addition, Notification makes internal

adjustments if the main menu is added or deleted.

A form can, if needed, override Notification to make other adjustments when components are inserted or removed. In particular, if a form has object fields or properties that contains references to other components, it might check the notifications of component removals and invalidate those references as needed.

See Also

InsertComponent

7.1.5.6.46 TCustomForm.Paint Method

Generates an OnPaint (see page 2233) event.

Pascal

```
procedure Paint; dynamic;
```

C++

```
__fastcall Paint();
```

Description

The form component calls Paint in response to the WM_PAINT (see page 5198) message (sent when the window needs to be painted) before rendering all controls on the form.

The default Paint method calls any event handler attached by the user to the OnPaint (see page 2233) event. Override Paint to block the OnPaint (see page 2233) event or to draw additional features before or after the OnPaint (see page 2233) event.

See Also

OnPaint (see page 2233)

7.1.5.6.47 TCustomForm.PaintWindow Method

Renders the image of the form.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Applications can't call this protected method. It is called when the form needs to be painted.

PaintWindow locks the canvas to prevent threading issues, assigns the DC parameter as the Handle of the form's canvas, and then calls the Paint (see page 2131) method to render the image of the form.

See Also

Paint (see page 2131)

Canvas (see page 2150)

Handle (see page 2722)

Lock (see page 2694)

7.1.5.6.48 TCustomForm.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

PaletteChanged is called automatically when the system palette changes. The Foreground parameter indicates whether the palette realization should treat this control's palette as being in the foreground or the background. The return value from PaletteChanged indicates whether the form actually realized its palette, returning true only if the form has an associated palette.

Most applications should not need to alter the default behavior of PaletteChanged.

See Also

PaletteChanged

7.1.5.6.49 TCustomForm.Print Method

Prints the form.

Pascal

```
procedure Print;
```

C++

```
__fastcall Print();
```

Description

Call Print to print the form. Print uses the GetFormImage (see page 2125) method to obtain a bitmap of the form and draws that to the printer's HDC.

See Also

GetFormImage (see page 2125)

7.1.5.6.50 TCustomForm.QueryInterface Method

Returns a reference to a specified interface if that interface is supported by the form.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

For forms that support interfaces, QueryInterface calls the QueryInterface method of the interface supported by the form. QueryInterface returns a reference to the interface specified by the IID parameter, as the Obj parameter. If the form does not support the interface, the Obj parameter is nil (Delphi) or NULL (C++).

QueryInterface returns S_OK if the Obj parameter is successfully assigned.

See Also

_AddRef

_Release

Invoke

7.1.5.6.51 TCustomForm.ReadState Method

Reads the form's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the VCL streaming system that loads and saves VCL components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

TReader

7.1.5.6.52 TCustomForm.RecreateAsPopup Method

Pascal

```
procedure RecreateAsPopup(AWindowHandle: HWND);
```

C++

```
__fastcall RecreateAsPopup(HWND AWindowHandle);
```

Description

This is RecreateAsPopup, a member of class TCustomForm.

7.1.5.6.53 TCustomForm.Release Method

Destroys the form and frees its associated memory.

Pascal

```
procedure Release;
```

C++

```
__fastcall Release();
```

Description

Use Release to destroy the form and free its associated memory.

Release does not destroy the form until all event handlers of the form and event handlers of components on the form have finished executing. Release also guarantees that all messages in the form's event queue are processed before the form is released. Any event handlers for the form or its children should use Release instead of Free (Delphi) or delete (C++). Failing to

do so can cause a memory access error.

Note: Release returns immediately to the caller. It does not wait for the form to be freed before returning.

See Also

System::TObject::Free

Destroy (see page 2109)

7.1.5.6.54 TCustomForm.RequestAlign Method

Instructs the form's parent (if any) to reposition the form, enforcing its Align property.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

RequestAlign is called internally when changes are made to the size or position of the form, so that it remains properly aligned within a dock site or properly positioned on the screen.

See Also

Align

Floating

7.1.5.6.55 TCustomForm.Resizing Method

Makes any required adjustments when the form changes size.

Pascal

```
procedure Resizing(State: TWindowState); override;
```

C++

```
virtual __fastcall Resizing(TWindowState State);
```

Description

This protected method is called automatically when the form is resized, including when it is maximized, minimized, or restored to normal size. Resizing adjusts the size and position of any child controls that are aligned to the form.

The State parameter indicates the state of the form after the resize.

7.1.5.6.56 TCustomForm.SendCancelMode Method

Cancels modes on the form.

Pascal

```
procedure SendCancelMode(Sender: TControl);
```

C++

```
__fastcall SendCancelMode(TControl * Sender);
```

Description

Use SendCancelMode to release the mouse capture and cancel scroll bar input and menu processing on the form.

See Also

ActiveControl (see page 2145)

7.1.5.6.57 TCustomForm.set_PopupParent Method

Pascal

```
procedure set_PopupParent(Value: TCustomForm);
```

C++

```
__fastcall set_PopupParent(TCustomForm Value);
```

Description

This is set_PopupParent, a member of class TCustomForm.

7.1.5.6.58 TCustomForm.SetChildOrder Method

Changes the order in which Child appears when GetChildren (see page 2124) is called.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

Use SetChildOrder to change the order in which child components are loaded and saved. The Child moves as if it were an item in a list object: items previously below the child's old position move up, and those below the new position move down.

See Also

GetChildren (see page 2124)

7.1.5.6.59 TCustomForm.SetFocus Method

Sets focus to the form.

Pascal

```
procedure SetFocus; override;
```

C++

```
virtual __fastcall SetFocus();
```

Description

Use SetFocus to give the form input focus. SetFocus calls the SetFocus method of the active control, if one exists. To determine whether the form has focus, use the Active (see page 2145) property.

See Also

Active (see page 2145)

SetFocusedControl (see page 2136)

SetFocus

7.1.5.6.60 TCustomForm.SetFocusedControl Method

Sets focus to a control on the form.

Pascal

```
function SetFocusedControl(Control: TWinControl): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SetFocusedControl(TWinControl * Control);
```

Description

Use SetFocusedControl to give a control on the form input focus. SetFocusedControl returns false if the Control specified by the Control parameter was already in the process of receiving focus, true otherwise.

Note: A return value of true does not indicate the control has successfully received input focus. If the Control can't have focus (for example, if it is not visible), SetFocusedControl will still return true, indicating that an attempt was made.

See Also

Active (see page 2145)

SetFocus (see page 2135)

DefocusControl (see page 2118)

7.1.5.6.61 TCustomForm.SetParent Method

Sets the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. It overrides the inherited method to regenerate the form's window when the Parent property is changed to nil (Delphi) or NULL (C++).

See Also

Parent

7.1.5.6.62 TCustomForm.SetParentBiDiMode Method

Sets the ParentBiDiMode property.

Pascal

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

C++

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

Description

SetParentBiDiMode is the protected implementation of the ParentBiDiMode property. It overrides the inherited method to obtain

the BiDiMode property of the application if the form does not have a parent.

See Also

ParentBiDiMode

7.1.5.6.63 TCustomForm.Show Method

Shows the form.

Pascal

```
procedure Show;
```

C++

```
__fastcall Show();
```

Description

Use Show to set the form's Visible (see page 2192) property to true and to bring the form to the front of other forms on the screen.

C++ Examples:

```
/*
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*_proj.cpp) file. To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.
You will need to set up your project with the following steps
before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place
               Form5 on the available forms list.
o          Change the code of your project (*_proj.cpp) file to
               look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
```

```
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, right
click on the executable file in the Project Manager and
select the View Source menu item.  You will need to set up
your project with the following steps before using the codeexample:    Add four additional
forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
```

```
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

BringToFront

Visible (see page 2192)

7.1.5.6.64 TCustomForm.ShowModal Method

Shows a form as a modal dialog.

Pascal

```
function ShowModal: Integer; virtual;
```

C++

```
virtual __fastcall int ShowModal();
```

Description

Use ShowModal to show a form as a modal form. A modal form is one where the application can't continue to run until the form is closed. Thus, ShowModal does not return until the form closes. When the form closes, it returns the value of the ModalResult (see page 2176) property.

To close a modal form, set its ModalResult (see page 2176) property to a nonzero value.

Note: If the form contains buttons with a ModalResult (see page 2176) property set to a value other than mrNone, the form automatically closes when the user clicks one of those buttons and returns the ModalResult (see page 2176) value as the return value of ShowModal.

You can check the return value against common return values using the global IsAbortResult, IsAnAllResult, IsNegativeResult, or IsPositiveResult functions.

C++ Examples:

```
/*
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
*/
#include "Unit2.h"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyDialogBox1->ShowModal() == mrOk)
    MessageBeep(0);
}
/*
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
*/
void __fastcall TMyDialogBox1::OKButtonClick(TObject *Sender)
{
```

```
  ModalResult = mrOk;
}

void __fastcall TMyDialogBox1::CancelButtonClick(TObject *Sender)
{
  ModalResult = mrCancel;
}
```

Delphi Examples:

```
{
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MyDialogBox1.ShowModal = mrOK then
    Beep;
end;
{
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
}
procedure TMyDialogBox1.CancelButtonClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;

procedure TMyDialogBox1.OKButtonClick(Sender: TObject);
begin
  ModalResult := mrOK;
end;
```

See Also

ModalResult (see page 2176)

ModalResult

IsAnALlResult

IsNegativeResult

IsPositiveResult

StripAllFromResult

IsAbortResult

7.1.5.6.65 TCustomForm.UpdateActions Method

Updates all actions associated with the form.

Pascal

```
procedure UpdateActions; virtual;
```

C++

```
virtual __fastcall UpdateActions();
```

Description

UpdateActions is called automatically when the application is idle to allow all components in the form an opportunity to update any actions for which the component is a target. This allows actions to be checked, grayed, and so on to reflect the current state of their targets.

UpdateActions updates all actions associated with the form, its menus, and the components contained in the form.

See Also

UpdateAction

InitiateAction

7.1.5.6.66 TCustomForm.UpdateWindowState Method

Updates the WindowState (see page 2195) property to reflect the current state of the form.

Pascal

```
procedure UpdateWindowState;
```

C++

```
__fastcall UpdateWindowState();
```

Description

UpdateWindowState ensures that the WindowState (see page 2195) property reflects the current state of the form.

See Also

WindowState (see page 2195)

7.1.5.6.67 TCustomForm.ValidateRename Method

Determines if a form can rename one of its owned components.

Pascal

```
procedure ValidateRename(AComponent: TComponent; const CurName: string; const NewName:
string); override;
```

C++

```
virtual __fastcall ValidateRename(TComponent * AComponent, const UnicodeString CurName, const
UnicodeString NewName);
```

Description

The ValidateRename method checks to see if a form can rename one of its owned components, passed in AComponent, from its current name (CurName) to the string passed in NewName. If AComponent is nil (Delphi) or NULL (C++) or NewName is already the name of a component in the form's list, ValidateRename raises an EComponentError exception.

See Also

FindComponent

7.1.5.6.68 TCustomForm.VisibleChanging Method

Responds just before the form changes the value of its Visible (see page 2192) property.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

This protected method raises an exception if, for some reason, the form should not change its visible state.

See Also

Visible (see page 2192)

7.1.5.6.69 TCustomForm.WantChildKey Method

Indicates whether the form processes keyboard input for a control it owns.

Pascal

```
function WantChildKey(Child: TControl; var Message: TMessage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean WantChildKey(TControl * Child, TMessage Message);
```

Description

WantChildKey is called by any control that the form owns when the control receives keyboard input. If WantChildKey returns true, the control performs no further processing in response to the keyboard input, instead deferring to its parent form to respond to the keystroke. In its default implementation, WantChildKey always returns false; it is overridden in TActiveForm to implement certain ActiveX functionality.

7.1.5.6.70 TCustomForm.WndProc Method

Provides specific message responses for the form.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method that receives messages for a form. After calling the inherited method, WndProc performs some window activation, focus, and position message handling necessary to keep in sync with Windows. Override WndProc to change how the form responds to Windows messages.

See Also

WndProc

7.1.5.7 TCustomForm Properties

The properties of the TCustomForm class are listed here.

Protected Properties

	Name	Description
R	ActiveMDIChild (see page 2146)	Specifies the MDI child that has focus.
	AlphaBlend (see page 2147)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2147)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 2148)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderIcons (see page 2148)	Specifies which icons appear on the title bar of the form.
R	ClientHandle (see page 2154)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 2154)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 2155)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 2158)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2158)	Specifies the monitor on which the form appears.
	FormStyle (see page 2162)	Determines the form's style.
	Height (see page 2164)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 2166)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 2167)	Specifies the icon that appears when the form is minimized.
R	MDIChildCount (see page 2172)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 2173)	Provides indexed access to all MDI child forms.
	ObjectMenuItem (see page 2177)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 2178)	Specifies when OnCreate (see page 2207) and OnDestroy (see page 2209) events occur.
	ParentFont (see page 2179)	Determines where a control looks for its font information.
	PixelsPerInch (see page 2179)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 2179)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 2180)	Identifies the pop-up menu associated with the control.
	Position (see page 2185)	Represents the size and placement of the form.
	PrintScale (see page 2186)	Represents the proportions of a printed form.
	Scaled (see page 2187)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 2179) property.
	TileMode (see page 2188)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 2190)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2191)	Indicates the color on the form that appears transparent when TransparentColor (see page 2190) is true.
	VertScrollBar (see page 2191)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 2192)	Indicates whether the form is visible.
	Width (see page 2193)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 2194)	Specifies the Window menu for an MDI parent form.

Public Properties

	Name	Description
	Action (see page 2144)	Designates the action associated with the control.
R	Active (see page 2145)	Specifies whether the form has focus.
	ActiveControl (see page 2145)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 2146)	Specifies the OLE control on the form that reacts to changes in focus.
	BorderStyle (see page 2149)	Specifies the appearance and behavior of the form border.
R	Canvas (see page 2150)	Provides access to the drawing area of the form.
	Caption (see page 2151)	Specifies a text string that identifies the control to the user.
	Color (see page 2157)	Specifies the background color of the control.
	Designer (see page 2159)	Specifies the designer interface for the form.
	DropTarget (see page 2159)	Specifies whether the form is the target of a drag-and-drop operation.
	Font (see page 2160)	Controls the attributes of text written on or in the control.
R	FormState (see page 2161)	Indicates transitional state information about the form.
	GlassFrame (see page 2164)	This is GlassFrame, a member of class TCustomForm.

	HelpFile (see page 2165)	Specifies the name of the file the form uses to display Help.
	KeyPreview (see page 2167)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 2174)	Specifies the form's main menu.
	ModalResult (see page 2176)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 2177)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 2178)	Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.
	PopupMode (see page 2185)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 2185)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 2187)	Specifies whether form snaps to edge of screen.
	SnapBuffer (see page 2188)	Specifies distance for screen snap.
	WindowState (see page 2195)	Represents how the form appears on the screen.

Published Properties

	Name	Description
	Left (see page 2171)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Top (see page 2189)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

7.1.5.7.1 TCustomForm.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
```

```
    Action.OnExecute := ChangeColor;

    { initialize the random number generator }
    Randomize;
  end;
```

7.1.5.7.2 TCustomForm.Active Property

Specifies whether the form has focus.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine whether the form has focus. The Active Form receives all keyboard input. If it has a title bar, it is drawn using the active colors specified in the Control panel.

If Active is true , the form has focus; if Active is false, the form does not have focus.

See Also

SetFocus (see page 2135)

OnActivate (see page 2196)

OnDeactivate (see page 2209)

ActiveControl (see page 2145)

7.1.5.7.3 TCustomForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
```

```
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 2146)

OnExit

OnEnter

Active (see page 2145)

7.1.5.7.4 TCustomForm.ActiveMDIChild Property

Specifies the MDI child that has focus.

Pascal

```
property ActiveMDIChild: TForm;
```

C++

```
__property TForm * ActiveMDIChild;
```

Description

Use ActiveMDIChild to get the MDI child that has focus.

If the form is not an MDI parent (that is, if the FormStyle (see page 2162) property of the form is not fsMDIForm), ActiveMDIChild returns nil (Delphi) or NULL (C++).

See Also

FormStyle (see page 2162)

MDIChildCount (see page 2172)

7.1.5.7.5 TCustomForm.ActiveOleControl Property

Specifies the OLE control on the form that reacts to changes in focus.

Pascal

```
property ActiveOleControl: TWinControl;
```

C++

```
__property TWinControl * ActiveOleControl;
```

Description

Use ActiveOLEControl to get or set an OLE control on the form that reacts to changes in focus.

If ActiveOLEControl is not nil (Delphi) or NULL (C++), the user interface for the specified OLE control is deactivated when it is not focused. Also, when the form receives or loses focus, the document window for the specified OLE control is activated or deactivated respectively.

See Also

ActiveControl (see page 2145)

TWinControl

7.1.5.7.6 TCustomForm.AlphaBlend Property

Specifies whether the form is translucent.

Pascal

```
property AlphaBlend: Boolean;
```

C++

```
__property Boolean AlphaBlend;
```

Description

Set AlphaBlend to specify that the form represents a layered window that allows a translucent color. The AlphaBlendValue (see page 2147) property specifies the degree of translucency.

Note: AlphaBlend does not work on all systems. The application must be running under Windows 2000 or better, and the machine must be a P90 or better.

See Also

AlphaBlendValue (see page 2147)

TransparentColor (see page 2190)

7.1.5.7.7 TCustomForm.AlphaBlendValue Property

Specifies the degree of translucency on a translucent form.

Pascal

```
property AlphaBlendValue: Byte;
```

C++

```
__property Byte AlphaBlendValue;
```

Description

Set AlphaBlendValue to a value between 0 and 255 to indicate the degree of translucency when the AlphaBlend (see page 2147) property is true. A value of 0 indicates a completely transparent window. A value of 255 indicates complete opacity.

Note: AlphaBlendValue only has an effect when the AlphaBlend (see page 2147) property is true.

See Also

AlphaBlend (see page 2147)

TransparentColor (see page 2190)

TransparentColorValue (see page 2191)

7.1.5.7.8 TCustomForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.5.7.9 TCustomForm.BorderIcons Property

Specifies which icons appear on the title bar of the form.

Pascal

```
property BorderIcons: TBorderIcons;
```

C++

```
__property TBorderIcons BorderIcons;
```

Description

Use BorderIcons to get or set the icons that appear on the title bar of the form. BorderIcons can include any of the following TBorderIcons (see page 2596) values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).
biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	If BorderStyle (see page 2149) is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise,no question mark appears.

Note: Certain combinations of the BorderIcons and BorderStyle (see page 2149) properties are mutually exclusive. For example, BorderIcons biMax, biMin with BorderStyle (see page 2149) of bsDialog are mutually exclusive.

C++ Examples:

```
/*
The following code removes a form's Maximize button when the
user clicks a button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TBorderIcons tempBI = BorderIcons;
  tempBI >> biMaximize;
  BorderIcons = tempBI;
}
```

Delphi Examples:

```
{
The following code removes a form's Maximize button when the
user clicks a button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
BorderIcons := BorderIcons - [biMaximize];
end;
```

See Also

BorderStyle (see page 2149)

7.1.5.7.10 TCustomForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm (see page 2241) BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 2148)

7.1.5.7.11 TCustomForm.Canvas Property

Provides access to the drawing area of the form.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use TCanvas (see page 2673) to draw on the client area of the form. Canvas is often used in the OnPaint (see page 2233) event handler.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> BrushBmp(new Graphics::TBitmap);
  BrushBmp->LoadFromFile("../bm1.BMP");
  Form1->Canvas->Brush->Bitmap = BrushBmp.get();
  Form1->Canvas->FillRect(Rect(0,0,100,100));
  Form1->Canvas->Brush->Bitmap = NULL;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TForm1 *myForm;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TForm1")
  {
    myForm = dynamic_cast<TForm1 *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myForm->Canvas->MoveTo(myForm->ClientRect.Left, myForm->ClientRect.Top);
    myForm->Canvas->LineTo(myForm->ClientRect.Right, myForm->ClientRect.Bottom);
    myForm = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TForm1) then
```

```
    begin
      myForm:= (ListBox1.Items.Objects[Index] as TForm1);
      with myForm.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myForm.Canvas.MoveTo(Left, Top);
        myForm.Canvas.LineTo(Right, Bottom);
      end;
    myForm:= nil;
    end;
  {
  The following code loads a bitmap from a file and assigns it
  to the Brush of the Canvas of Form1:
  }
  var
    Bitmap: TBitmap;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Bitmap := TBitmap.Create;
    try
      Bitmap.LoadFromFile('bm1.BMP');
      Form1.Canvas.Brush.Bitmap := Bitmap;
      Form1.Canvas.FillRect(Rect(0,0,100,100));
    finally
      Form1.Canvas.Brush.Bitmap := nil;
      Bitmap.Free;
    end;
  end;
```

See Also

TCanvas (see page 2673)

OnPaint (see page 2233)

7.1.5.7.12 TCustomForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
```

```
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
```

```
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

7.1.5.7.13 TCustomForm.ClientHandle Property

Provides access to the handle of the form's internal MDI client window.

Pascal

```
property ClientHandle: HWND;
```

C++

```
__property HWND ClientHandle;
```

Description

Use ClientHandle to get the handle for the form's internal MDI client window.

ClientHandle is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 2162) property is set to fsMDIForm).

Note: Use the Handle property instead to get the handle to the form window itself.

See Also

FormStyle (see page 2162)

Handle

7.1.5.7.14 TCustomForm.ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Pascal

```
property ClientHeight;
```

C++

```
__property ClientHeight;
```

Description

Use ClientHeight to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set ClientHeight to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the Height (see page 2164) property instead.

C++ Examples:

```
/*
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form1->ClientHeight = Form1->ClientHeight/2;
}
```

Delphi Examples:

```
{
```

```
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientHeight := ClientHeight div 2;
end;
```

See Also

ClientRect

ClientWidth (see page 2155)

Height

7.1.5.7.15 TCustomForm.ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Pascal

```
property ClientWidth;
```

C++

```
__property ClientWidth;
```

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width (see page 2193) property instead.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
```

```
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

ClientHeight (see page 2154)

ClientRect

Width

7.1.5.7.16 TCustomForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
```

```
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

7.1.5.7.17 TCustomForm.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

7.1.5.7.18 TCustomForm.DefaultMonitor Property

Specifies the monitor on which the form appears.

Pascal

```
property DefaultMonitor: TDefaultMonitor;
```

C++

```
__property TDefaultMonitor DefaultMonitor;
```

Description

Use DefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

Note: DefaultMonitor has no effect if the application does not have a main form.

See Also

Monitors (see page 2474)

TMonitor (see page 2439)

Screen (see page 2615)

MainForm (see page 1983)

Active (see page 2145)

Monitor (see page 2177)

MakeFullyVisible (see page 2129)

7.1.5.7.19 TCustomForm.Designer Property

Specifies the designer interface for the form.

Pascal

```
property Designer: IDesignerHook;
```

C++

```
__property IDesignerHook Designer;
```

Description

The Designer property is used internally at design time. Never assign a value to this property. This is done automatically by the form designer. Only use this property when designing classes for use in the form designer, such as property editors and component editors.

See Also

TPropertyEditor

TComponentEditor

7.1.5.7.20 TCustomForm.DropTarget Property

Specifies whether the form is the target of a drag-and-drop operation.

Pascal

```
property DropTarget: Boolean;
```

C++

```
__property Boolean DropTarget;
```

Description

Use DropTarget to determine whether the form is currently the target of a drag-and-drop operation.

See Also

BeginDrag

DragMode

EndDrag

OnDragOver

7.1.5.7.21 TCustomForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

7.1.5.7.22 TCustomForm.FormState Property

Indicates transitional state information about the form.

Pascal

```
property FormState: TFormState;
```

C++

```
__property TFormState FormState;
```

Description

Read FormState to determine the state of the form. FormState indicates when the form is in various transitional states or when certain Windows operations have occurred. The following table lists the values that can be included in a form's state:

Value	Meaning
fsCreating	The form's constructor is currently executing.
fsVisible	The form's window is visible. This state is used to update the Visible (see page 2192) property.
fsShowing	The form's WindowState (see page 2195) property is changing. This state is used to prevent WindowState (see page 2195) changes from interfering with a transition that is in progress.
fsModal	The form was created as a modal window.
fsCreatedMDIChild	The form is an MDI parent window. This state is only set once the MDI client window has been created.
fsActivated	The form has received a CM_ACTIVATE message (because it received focus or the application became active) but has not yet called the Activate (see page 2111) method to generate an OnActivate (see page 2196) event.

See Also

Visible (see page 2192)

Create (see page 2108)

WindowState (see page 2195)

ShowModal (see page 2139)

FormStyle (see page 2162)

Activate (see page 2111)

7.1.5.7.23 TCustomForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example ensures that the main form of the application
is an MDI parent:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  if(FormStyle != fsMDIForm)
          FormStyle = fsMDIForm;
  if(FormStyle == fsMDIForm)
          Edit1->Text = "MDI form";
  else
```

```
    //This line never executes
    Edit1->Text = "Not an MDI form";
}
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example ensures that the main form of the application
is an MDI parent:
}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

constructor TForm1.Create(AOwner : TComponent);

begin
  inherited Create(AOwner);
  if FormStyle <> fsMDIForm then
          FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
          Edit1.Text := 'MDI form'
  else
```

```
    //This line never executes
    Edit1.Text := 'Not an MDI form';
end;
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

ArrangeIcons (see page 2247)

Tile (see page 2249)

Cascade (see page 2248)

Windows Overview

7.1.5.7.24 TCustomForm.GlassFrame Property

Pascal

```
property GlassFrame: TGlassFrame;
```

C++

```
__property TGlassFrame GlassFrame;
```

Description

This is GlassFrame, a member of class TCustomForm.

7.1.5.7.25 TCustomForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
```

```
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

7.1.5.7.26 TCustomForm.HelpFile Property

Specifies the name of the file the form uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property UnicodeString HelpFile;
```

Description

Use HelpFile for forms that have a help file to use with the Help system. Help topics are selected from the specified file.

See Also

HelpContext

OnHelp (see page 2217)

HelpFile (see page 1974)

HelpKeyword

7.1.5.7.27 TCustomForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
```

```
      Range := 600;
      Position := Range - ClientWidth; { start form out fully scrolled }
      Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
      Visible := True;  { Show the scrollbar }
    end;
  end;
```

See Also

AutoScroll (see page 2561)

VertScrollBar (see page 2563)

TControlScrollBar (see page 2009)

7.1.5.7.28 TCustomForm.Icon Property

Specifies the icon that appears when the form is minimized.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Set the Icon property to provide an icon for the form. If Icon is not set, the system provides a default icon when the form is minimized.

C++ Examples:

```
/*
This code assigns an icon to a form when the form is created:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Icon->LoadFromFile("..\\MDIChild1.ICO");
}
```

Delphi Examples:

```
{
This code assigns an icon to a form when the form is created:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MDIChild1.ICO');
end;
```

See Also

WindowState (see page 2195)

7.1.5.7.29 TCustomForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl (see page 2145) property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

ActiveControl (see page 2145)

OnKeyDown

OnKeyUp

OnKeyPress

7.1.5.7.30 TCustomForm.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
```

```
    if ActiveControl <> nil then
      ActiveControl.Left := ActiveControl.Left + 1;
  end;
```

See Also

Top

7.1.5.7.31 TCustomForm.MDIChildCount Property

Specifies the number of open MDI child forms.

Pascal

```
property MDIChildCount: Integer;
```

C++

```
__property int MDIChildCount;
```

Description

Use MDIChildCount to get the number of open MDI child forms.

MDIChildCount is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 2162) property is set to fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
*/
void __fastcall TForm1::CloseChildrenClick(TObject *Sender)
{
  for(int i = MDIChildCount - 1; i >= 0; i--)
    MDIChildren[i]->Close();
}

void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}

void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
  }
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
```

```
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildren (see page 2173)

7.1.5.7.32 TCustomForm.MDIChildren Property

Provides indexed access to all MDI child forms.

Pascal

```
property MDIChildren [I: Integer]: TForm;
```

C++

```
__property TForm * MDIChildren[int I];
```

Description

Use MDIChildren to access a child form.

I is the index of the child form to access. The order of MDI children changes as different MDI forms are activated. Initially, forms are listed in creation order, with new forms being added to the end of the array. Whenever an MDI child is activated, however, it moves to the front of the list. Thus, use I to index over all forms in the array, but not to refer to a specific form.

MDIChildren is meaningful only if the form is an MDI frame (that is, if the form's FormStyle (see page 2162) property is set to fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
*/
void __fastcall TForm1::CloseChildrenClick(TObject *Sender)
{
  for(int i = MDIChildCount - 1; i >= 0; i--)
    MDIChildren[i]->Close();
}

void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
```

```
  Form1->ArrangeIcons();
}

void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
  }
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildCount (see page 2172)

FormStyle (see page 2162)

7.1.5.7.33 TCustomForm.Menu Property

Specifies the form's main menu.

Pascal

```
property Menu: TMainMenu;
```

C++

```
__property TMainMenu Menu;
```

Description

Use Menu to get or set the form's main menu. At design time, Menu is set to the first TMainMenu (see page 4828) component added to the form.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
```

```
definition of the TForm1 class.  Click on the menu item
or type "Ctrl+N" to execute MyOnExecute.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

TMainMenu (see page 4828)

7.1.5.7.34 TCustomForm.ModalResult Property

Represents the return value of a form that is used as a modal dialog.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Use ModalResult to close the form when it is displayed modally.

By default, ModalResult is mrNone. Set ModalResult to any nonzero value to close the form. The value assigned to ModalResult becomes the return value of the ShowModal (see page 2139) function call used to display the form.

Note: This property is modified automatically by some components such as the TButton object.

C++ Examples:

```
/*
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
*/
#include "Unit2.h"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyDialogBox1->ShowModal() == mrOk)
    MessageBeep(0);
}
/*
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
*/
void __fastcall TMyDialogBox1::OKButtonClick(TObject *Sender)
{
  ModalResult = mrOk;
}

void __fastcall TMyDialogBox1::CancelButtonClick(TObject *Sender)
{
  ModalResult = mrCancel;
}
```

Delphi Examples:

```
{
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    if MyDialogBox1.ShowModal = mrOK then
      Beep;
  end;
  {
  The following methods are used for buttons in a form that is
  used as a modal dialog box. The methods cause the dialog box
  to terminate when the user clicks either the OK or Cancel
  button, returning mrOk or mrCancel from ShowModal,
  respectively. You could also set the ModalResult value to
  mrOk for the OK button and mrCancel for the Cancel button to
  accomplish the same thing. When the user clicks either
  button, the dialog box closes.
  }
  procedure TMyDialogBox1.CancelButtonClick(Sender: TObject);
  begin
    ModalResult := mrCancel;
  end;

  procedure TMyDialogBox1.OKButtonClick(Sender: TObject);
  begin
    ModalResult := mrOK;
  end;
```

See Also

ShowModal (see page 2139)

ModalResult

7.1.5.7.35 TCustomForm.Monitor Property

Provides access to the monitor on which the form appears.

Pascal

```
property Monitor: TMonitor;
```

C++

```
__property TMonitor Monitor;
```

Description

Use Monitor to access information about the monitor on which the form appears if the application runs on a multi-monitor system. The monitor is determined by the DefaultMonitor (see page 2158) property.

See Also

Monitors (see page 2474)

BoundsRect

DefaultMonitor (see page 2158)

MakeFullyVisible (see page 2129)

7.1.5.7.36 TCustomForm.ObjectMenuItem Property

Represents an OLE object menu item that reacts to selections of OLE objects.

Pascal

```
property ObjectMenuItem: TMenuItem;
```

C++

```
__property TMenuItem ObjectMenuItem;
```

Description

Use ObjectMenuItem to get or set a menu item that becomes enabled or disabled when an OLE object on the form is selected or unselected respectively.

The OLE object menu item can be used for standard OLE commands, such as Activate (see page 2111) or Convert. (The implementations of these commands are provided by the OLE server.)

See Also

OleFormObject (see page 2178)

7.1.5.7.37 TCustomForm.OldCreateOrder Property

Specifies when OnCreate (see page 2207) and OnDestroy (see page 2209) events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate (see page 2207) event occurs after all constructors are finished and the OnDestroy (see page 2209) event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate (see page 2207) event occurred when the TCustomForm (see page 2103) constructor executed and the OnDestroy (see page 2209) event occurred when the TCustomForm (see page 2103) destructor executed. Applications that require the OnCreate (see page 2207) event and OnDestroy (see page 2209) events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 2207)

OnDestroy (see page 2209)

BeforeDestruction (see page 2113)

AfterConstruction (see page 2112)

7.1.5.7.38 TCustomForm.OleFormObject Property

Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.

Pascal

```
property OleFormObject: IOleForm;
```

C++

```
__property IOleForm OleFormObject;
```

Description

OleFormObject is used to communicate resize and destroy notifications when a form is used as a frame for an in-place OLE object.

See Also

ObjectMenuItem (see page 2177)

7.1.5.7.39 TCustomForm.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

7.1.5.7.40 TCustomForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled (see page 2187) property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 2187)

7.1.5.7.41 TCustomForm.PopupChildren Property

Pascal

```
property PopupChildren: TList;
```

C++

```
__property TList PopupChildren;
```

Description

This is PopupChildren, a member of class TCustomForm.

7.1.5.7.42 TCustomForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
```

```
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
```

```
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
```

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
```

```
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

7.1.5.7.43 TCustomForm.PopupMode Property

Pascal

```
property PopupMode: TPopupMode;
```

C++

```
__property TPopupMode PopupMode;
```

Description

This is PopupMode, a member of class TCustomForm.

7.1.5.7.44 TCustomForm.PopupParent Property

Sets an order for stacked forms that users cannot change.

Pascal

```
property PopupParent: TCustomForm;
```

C++

```
__property TCustomForm PopupParent;
```

Description

If the PopupMode (see page 2185) property is set to pmExplicit and PopupParent is nil, then the Application.MainForm is implicitly used as the PopupParent. If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent.

If the PopupMode (see page 2185) property is set to pmAuto, Screen.ActiveForm is used as the PopupParent property.

7.1.5.7.45 TCustomForm.Position Property

Represents the size and placement of the form.

Pascal

```
property Position: TPosition;
```

C++

```
__property TPosition Position;
```

Description

Use Position to get or set the size and placement of the form. Position can have one of the following TPosition (see page 2590) values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.

poDefault	The form appears in a position on the screen and with a height and width determined by the operating system. Each time you run the application, the form moves slightly down and to the right. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle (see page 2149) of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen. Each time you run the application, the form moves slightly down and to the right. When the form can no longer move down and to the right and keep the same size while remaining entirely visible on the screen, the form displays at the top-left corner of the screen. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle (see page 2149) of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. In multi-monitor applications, the form may be moved from this center position so that it falls entirely on one monitor, as specified by the DefaultMonitor (see page 2158) property.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. No adjustments are made for multi-monitor applications.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. No adjustments are made for multi-monitor applications. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

OnResize

SetInitialBounds

7.1.5.7.46 TCustomForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale (see page 2591) values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.

poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 2132)

7.1.5.7.47 TCustomForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch (see page 2179) property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch (see page 2179) property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 2179)

Considerations When Dynamically Resizing Forms and Controls

7.1.5.7.48 TCustomForm.ScreenSnap Property

Specifies whether form snaps to edge of screen.

Pascal

```
property ScreenSnap: Boolean;
```

C++

```
__property Boolean ScreenSnap;
```

Description

ScreenSnap determines whether the edges of the form snap to the edge of the screen when the user moves the form. To control the snap distance, set the SnapBuffer (see page 2188) property.

See Also

SnapBuffer (see page 2188)

7.1.5.7.49 TCustomForm.SnapBuffer Property

Specifies distance for screen snap.

Pascal

```
property SnapBuffer: Integer;
```

C++

```
__property int SnapBuffer;
```

Description

If ScreenSnap (see page 2187) is true, SnapBuffer determines the maximum number of pixels that be between the form edge and the screen edge before the form snaps to the screen edge.

See Also

ScreenSnap (see page 2187)

7.1.5.7.50 TCustomForm.TileMode Property

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
property TileMode: TTileMode;
```

C++

```
__property TTileMode TileMode;
```

Description

TileMode can have one of the following TTileMode (see page 2593) values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form
tbVertical	Each form stretches along the height of the parent form

TileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle (see page 2162) property is set to fsMDIChild).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

FormStyle (see page 2162)

Tile (see page 2249)

7.1.5.7.51 TCustomForm.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    dynamic_cast<TWinControl *>(GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Left

7.1.5.7.52 TCustomForm.TransparentColor Property

Specifies whether a color on the form appears transparent.

Pascal

```
property TransparentColor: Boolean;
```

C++

```
__property Boolean TransparentColor;
```

Description

Use TransparentColor to indicate that one of the colors on the form should be treated as transparent, allowing windows behind the form to completely show through. The TransparentColorValue (see page 2191) property indicates the color that appears completely transparent.

Note: To make the entire form transparent, or to make it translucent rather than transparent, use the AlphaBlend (see page 2147) and AlphaBlendValue (see page 2147) properties.

Note: TransparentColor does not work on all systems. The application must be running under Windows 2000 or better, with adequate hardware.

See Also

TransparentColorValue (see page 2191)

AlphaBlend (see page 2147)

AlphaBlendValue (see page 2147)

7.1.5.7.53 TCustomForm.TransparentColorValue Property

Indicates the color on the form that appears transparent when TransparentColor (see page 2190) is true.

Pascal

```
property TransparentColorValue: TColor;
```

C++

```
__property TColor TransparentColorValue;
```

Description

Use TransparentColorValue to indicate the color that appears transparent when the TransparentColor (see page 2190) property is true.

Note: If the application uses a 16-bit color palette, TransparentColorValue must be a basic color.

See Also

TransparentColor (see page 2190)

7.1.5.7.54 TCustomForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
```

```
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

TControlScrollBar (see page 2009)

7.1.5.7.55 TCustomForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show (see page 2137) and ShowModal (see page 2139) methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront

Show (see page 2137)

ShowModal (see page 2139)

7.1.5.7.56 TCustomForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
```

```
    pdlb->Width = Form1->ClientWidth/3;

    // Now use the splitter to divide the directory pane from the file pane.
    ps->Parent = Form1;
    ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
    ps->Align = pdlb->Align; // Give same alignment as directory list.

    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

7.1.5.7.57 TCustomForm.WindowMenu Property

Specifies the Window menu for an MDI parent form.

Pascal

```
property WindowMenu: TMenuItem;
```

C++

```
__property TMenuItem WindowMenu;
```

Description

Use WindowMenu to get or set the Window menu for an MDI parent form. The Window menu is a standard menu in MDI applications. It contains commands that let the user manage the windows in the application. Menu (see page 2174) items usually include Cascade, Arrange Icons, Tile, and so on.

The Window menu also lists (at the bottom) the child windows that are currently open in the application. When the user selects one of these windows from the menu, the window becomes the active window in the application.

Although this menu is commonly called the Window menu, it can have any name that corresponds to an existing menu item on the menu bar. The names of open child forms are merged onto the menu automatically at run-time.

WindowMenu is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 2162) property is set to fsMDIForm).

See Also

FormStyle (see page 2162)

7.1.5.7.58 TCustomForm.WindowState Property

Represents how the form appears on the screen.

Pascal

```
property WindowState: TWindowState;
```

C++

```
__property TWindowState WindowState;
```

Description

Set WindowState to minimize, maximize, or restore the form window. Read WindowState to determine whether the form is minimized, maximized, or in a normal state.

See Also

Icon (see page 2167)

BorderStyle (see page 2149)

7.1.5.8 TCustomForm Events

The events of the TCustomForm class are listed here.

Protected Events

	Name	Description
	OnActivate (see page 2196)	Occurs when the form becomes active.
	OnCanResize (see page 2202)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2203)	Occurs when the user clicks the control.
	OnClose (see page 2205)	Occurs when the form closes.
	OnCloseQuery (see page 2206)	Occurs when close is attempted.
	OnCreate (see page 2207)	Occurs when the form is created.

	OnDblClick (see page 2209)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2209)	Occurs when the form loses focus.
	OnDestroy (see page 2209)	Occurs when the form is destroyed.
	OnDragDrop (see page 2211)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2214)	Occurs when the user drags an object over a control.
	OnHelp (see page 2217)	Occurs when the form receives a request for help.
	OnHide (see page 2217)	Occurs when the form is hidden (that is, when its Visible (see page 2192) property is set to false).
	OnKeyDown (see page 2219)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2223)	Occurs when a key is pressed.
	OnKeyUp (see page 2224)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2226)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2227)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 2229)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2231)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 2233)	Occurs when the form is redrawn.
	OnResize (see page 2235)	Occurs immediately after the control is resized.
	OnShortCut (see page 2235)	Occurs when the user presses a key (before the OnKeyDown (see page 2219) event).
	OnShow (see page 2236)	Occurs when the form is shown (that is, when its Visible (see page 2192) property is set to true).

7.1.5.8.1 TCustomForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication (see page 1924)), not the form, occurs when Windows switches control from another application.

C++ Examples:

```
/*
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
*/

TForm1 *Form1;
TTable *Customers, *Blobs;
Boolean Form1Done = False;

void CreateCustSource()
```

```
{
  Customers = new TTable(Form1); // The owner will clean this up.
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  Form1->CustomerDS->DataSet = Customers;
  Customers->Active = True;
}

void CreateBlobSource()
{
  int i;
  Blobs = new TTable(Form1);
  Blobs->DatabaseName = "DBDEMOS";
  Blobs->TableType = ttParadox;
  Blobs->TableName = "MyBlobInfo";
  Blobs->Active = False;
    // Don't overwrite an existing table
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
  if (Blobs->Exists)
  {
    Blobs->Close();
    Blobs->DeleteTable();
  };
  // The Table component must not be active }
  // First, describe the type of table and give }
  // it a name }
  // Next, describe the fields in the table }
  Blobs->FieldDefs->Clear();
  TFieldDef *newDef = Blobs->FieldDefs->AddFieldDef();
  newDef->Name = "Field1";
  newDef->DataType = ftInteger;
  newDef->Required = True;
  newDef = Blobs->FieldDefs->AddFieldDef();
```

```
    newDef->Name = "Field2";
    newDef->DataType = ftBlob;
    newDef->Size = 30;
    // Next, describe any indexes }
    Blobs->IndexDefs->Clear();
    // The 1st index has no name because it is
    // a Paradox primary key }
    TIndexDef *newIDef = Blobs->IndexDefs->AddIndexDef();
    newIDef->Name = "";
    newIDef->Fields = "Field1";
    newIDef->Options << ixPrimary;
  /*
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
  */
    // Call the CreateTable method to create the table }
    Blobs->CreateTable();
    Blobs->Active = True;
    for (int i = 1; i <= 20; i++)
      Blobs->AppendRecord(ARRAYOFCONST((i*111, IntToStr(i*222))));
    Form1->BlobDS->DataSet = Blobs;
    Blobs->Active = True;
  };


  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    CreateCustSource();
    CreateBlobSource();
    for (int i = 0; i < ComponentCount; i++)
    {
      if (Components[i]->ClassNameIs("TDataSource"))
        ListBox1->Items->AddObject(Components[i]->Name,
           dynamic_cast<TObject *>(Components[i]));
    }
  }

  void __fastcall TForm1::ListBox1MouseDown(TObject *Sender, TMouseButton Button, TShiftState
  Shift,
            int X, int Y)
  {
    Form1Done = True;
  }
  /*
  The following code sets the DataSource for a DBGrid
  component in Form2 to the currently selected data source in
  a list provided in Form1. This allows users to specify the
  desired information in Form1, and then move to Form2 to view
  the information. Select an item in the listbox and then
  activate Form2 by clicking in it. Also, set the visible property
  of Form2 to true.
  */
  void __fastcall TForm2::FormActivate(TObject *Sender)
  {
    if (Form1Done)
    {
      DBGrid2->DataSource =
        dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
      DBNavigator2->DataSource =
        dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
    };
  }
```

Delphi Examples:

```
{
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;

procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
//    Looking for c:/ProgramFiles/Common Files/Codegear Shared/Data/dbdemos.gdb
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
```

```
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field3';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field4';
          DataType := ftString;
          Size := 30;
        end;
      end;

//    Take out this part to remove indexing
//    Next, describe any indexes
      with IndexDefs do
      begin
        Clear;
//        The 1st index has no name because it is
//        a Paradox primary key
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;

//      Call the CreateTable method to create the table
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;
```

```
procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
    begin
      if (Blobs.Exists) then
      begin
        Blobs.Close;
        Blobs.DeleteTable;
      end;
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftBlob;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
{
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
}
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Blobs.Active:= True;
```

```
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
  end;

  procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  begin
    Form1Done := True;
  end;

  initialization

  Form1Done := False;
  {
  The following code sets the DataSource for a DBGrid
  component in Form2 to the currently selected data source in
  a list provided in Form1. This allows users to specify the
  desired information in Form1, and then move to Form2 to view
  the information. Select an item in the listbox and then
  activate Form2 by clicking in it. Also, set the visible property
  of Form2 to true.
  }
  procedure TForm2.FormActivate(Sender: TObject);
  begin
    if (Form1Done) then
    begin
      DBGrid2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
      DBNavigator2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    end;
  end;
```

See Also

FormStyle (see page 2162)

OnDeactivate (see page 2209)

OnActivate (see page 1992)

7.1.5.8.2 TCustomForm.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

7.1.5.8.3 TCustomForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

7.1.5.8.4 TCustomForm.OnClose Event

Occurs when the form closes.

Pascal

```
property OnClose: TCloseEvent;
```

C++

```
__property TCloseEvent OnClose;
```

Description

Use OnClose to perform special processing when the form closes. The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close (see page 2114) method or when the user chooses Close (see page 2114) from the form's system menu.

The TCloseEvent (see page 2597) type points to a method that handles the closing of a form. The value of the Action (see page 2144) parameter determines if the form actually closes. These are the possible values of Action (see page 2144):

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

If a form is an MDI child form, and its BorderIcons (see page 2148) property is biMinimize, then the default Action (see page 2144) is caMinimize. If a MDI child form does not have these settings, the default Action (see page 2144) is caNone, meaning that nothing happens when the user attempts to close the form.

If a form is an SDI child form, Action (see page 2144) defaults to caHide.

To close the form and free it in an OnClose event, set Action (see page 2144) to caFree.

Note: When the application shuts down, the main form receives an OnClose event, but any child forms do not receive the OnClose event.

C++ Examples:

```
/*
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
*/
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
  if (MessageDlg("Close application ?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
    Action = caFree;
```

```
  else
    Action = caMinimize;
}
```

Delphi Examples:

```
{
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
}
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if MessageDlg('Close application ?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
    Action := caFree
  else
    Action := caMinimize;
end;
```

See Also

OnCreate (see page 2207)

OnCloseQuery (see page 2206)

OnDestroy (see page 2209)

7.1.5.8.5 TCustomForm.OnCloseQuery Event

Occurs when close is attempted.

Pascal

```
property OnCloseQuery: TCloseQueryEvent;
```

C++

```
__property TCloseQueryEvent OnCloseQuery;
```

Description

Use OnCloseQuery to specify the conditions under which the form can close. An OnCloseQuery event handler returns a Boolean CanClose value that determines whether a form is allowed to close. Its default value is true.

You can use an OnCloseQuery event handler to ask users if they are sure they really want the form closed immediately. For example, you can use the handler to display a message box that prompts the user to save a file before closing the form.

The TCloseQueryEvent (see page 2598) type points to the method that determines whether a form can be closed. The value of the CanClose parameter determines if the form can close or not.

C++ Examples:

```
/*
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
*/
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
```

```
{
  if (MessageDlg("Close the form?", mtConfirmation, TMsgDlgButtons() << mbOK << mbCancel,0) ==
mrCancel)
    CanClose = false;
}
```

Delphi Examples:

```
{
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
}
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Close the form?', mtConfirmation,
    [mbOk, mbCancel], 0) = mrCancel then
     CanClose := False;
end;
```

See Also

OnClose (see page 2205)

7.1.5.8.6 TCustomForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm (see page 2103)'s constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy (see page 2209) event.

When a form is being created and its Visible (see page 2192) property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint (see page 2233)

Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder (see page 2178)). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);  // These tabsheets are cleaned up by their
owner (PageControl1).
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

OnDestroy (see page 2209)

OnActivate (see page 2196)

OnPaint (see page 2233)

OnShow (see page 2236)

OldCreateOrder (see page 2178)

7.1.5.8.7 TCustomForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

7.1.5.8.8 TCustomForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication (see page 1924) object's OnDeactivate event.

See Also

OnActivate (see page 2196)

OnDeactivate (see page 1992)

7.1.5.8.9 TCustomForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate (see page 2207) event.

C++ Examples:

```
/*
```

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
*/
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 2207)

System::TObject::Free

Destroy (see page 2109)

OldCreateOrder (see page 2178)

7.1.5.8.10 TCustomForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.5.8.11 TCustomForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.5.8.12 TCustomForm.OnHelp Event

Occurs when the form receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set the CallHelp parameter to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. The form calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The event handler returns true if it succeeds, false if it fails.

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpFile (see page 2165)

HelpJump (see page 1942)

HelpFile (see page 1974)

OnHelp (see page 1995)

7.1.5.8.13 TCustomForm.OnHide Event

Occurs when the form is hidden (that is, when its Visible (see page 2192) property is set to false).

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Use OnHide to perform special processing when the form is hidden (that is, when the form's Visible (see page 2192) property is set to false).

C++ Examples:

```
/*
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
*/
#include "TFormOnHide.h"
#include "Unit2.h"  // We must include unit2's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
  Hide();
}

void __fastcall TForm1::FormHide(TObject *Sender)
{
  Form2->Label1->Caption = "Form1 is hiding";
}
/*
This is the implementation of Unit2:
*/
#include "Unit2.h"
#include "TFormOnHide.h"  // We must include unit1's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm2::Button1Click(TObject *Sender)
{
  Form1->Show();
  Hide();
}

void __fastcall TForm2::FormHide(TObject *Sender)
{
  Form1->Label1->Caption = "Form2 is hiding";
}
```

Delphi Examples:

```
{
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
}

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
  Hide;
end;

procedure TForm1.FormHide(Sender: TObject);
begin
  Form2.Label1.Caption := 'Form1 is hiding';
end;
{
This is the implementation section of Unit2:
}
implementation

{$R *.dfm}

uses TFormOnHide;

procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Hide;
end;

procedure TForm2.FormHide(Sender: TObject);
begin
  Form1.Label1.Caption := 'Form2 is hiding';
end;
```

See Also

OnShow (see page 2236)

7.1.5.8.14 TCustomForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse

buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
```

```
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;
```

```
procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
```

```
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

7.1.5.8.15 TCustomForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/
```

```
void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

7.1.5.8.16 TCustomForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
```

```
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

7.1.5.8.17 TCustomForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.5.8.18 TCustomForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.5.8.19 TCustomForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

7.1.5.8.20 TCustomForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
```

```
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
```

```
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

7.1.5.8.21 TCustomForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
  property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas (see page 2150) property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _TheGraphicCleaner(TheGraphic = new
Graphics::TBitmap());  // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 2150)

ClipRect (see page 2718)

7.1.5.8.22 TCustomForm.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

7.1.5.8.23 TCustomForm.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown (see page 2219) event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form handles them. When the user presses a key, the form can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown (see page 2219), OnKeyPress (see page 2223), and OnKeyUp (see page 2224)). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut allows the form to implement additional shortcuts.

If the OnShortCut implements a response to the keystroke, set the Handled parameter of the event handler to true. This prevents the keystroke from being passed on to menus or actions associated with the form. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

See Also

OnKeyDown

IsShortCut (see page 2128)

IsShortCut

IsShortCut (see page 4845)

7.1.5.8.24 TCustomForm.OnShow Event

Occurs when the form is shown (that is, when its Visible (see page 2192) property is set to true).

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Use OnShow to perform special processing when the form is shown (that is, when the form's Visible (see page 2192) property is set to true).

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
```

```
  populate the Page Control by right clicking and selecting
  New Page from the context menu. As the tabs are selected by
  the user, the Page Control OnChange event fires, which
  causes the form caption to display the captions of each of
  the TTabSheets attached to the Page Control.
  Note the use of the Form's OnShow event handler to set the
  caption when the form first appears.  This occurs before
  any changes trigger the OnChange event of the Page Control.
  }
  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
  end;

  procedure TForm1.FormShow(Sender: TObject);
  begin
    PageControl1Change(Sender);
  end;
```

See Also

OnHide (see page 2217)

7.1.6 TCustomFrame Class

TCustomFrame is the base class from which TFrame (see page 2366) descends.

Class Hierarchy

Pascal

```
TCustomFrame = class(TScrollingWinControl);
```

C++

```
class TCustomFrame : public TScrollingWinControl;
```

File

Forms

Description

TCustomFrame is the base class for frames. A frame, like a form, is a container for controls and components. It uses the same ownership mechanism as forms for automatic instantiation and destruction of the components on it, and the same parent-child relationships for synchronization of component properties. Unlike forms, however, frames can be nested within forms or other frames, and can be saved on the Component palette for easy reuse. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. Moreover, an embedded frame continues to inherit changes made to the frame from which it is derived.

Do not create instances of TCustomFrame. Instead, to create and work with frames, define a TCustomFrame descendant by choosing File|New|Frame, and adding the controls and components that you want your frame to encapsulate.

See Also

TForm (see page 2241)

TScrollingWinControl (see page 2554)

TFrame (see page 2366)

7.1.6.1 TCustomFrame Members

The following tables list the members exposed by TCustomFrame.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2239)	Initializes frame creation parameters for efficient redrawing.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	PaintWindow (see page 2240)	Renders the image of a windowed control.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	SetParent (see page 2240)	Sets the Parent property.

Protected Properties

	Name	Description
	AutoScroll (see page 2561)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Public Methods

	Name	Description
	Create (see page 2238)	Creates and initializes a TCustomFrame (see page 2237) object.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	GetChildren (see page 2239)	Calls a specified method for each child of the frame.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.

7.1.6.2 TCustomFrame.Create Constructor

Creates and initializes a TCustomFrame (see page 2237) object.

Pascal

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

C++

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

Description

Use Create to create and initialize a frame.

AOwner is the component that owns the frame.

See Also

TFrame (see page 2366)

Owner

ParentWindow

7.1.6.3 TCustomFrame Methods

The methods of the TCustomFrame class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2239)	Initializes frame creation parameters for efficient redrawing.
	PaintWindow (see page 2240)	Renders the image of a windowed control.
	SetParent (see page 2240)	Sets the Parent property.

Public Methods

	Name	Description
	GetChildren (see page 2239)	Calls a specified method for each child of the frame.

7.1.6.3.1 TCustomFrame.CreateParams Method

Initializes frame creation parameters for efficient redrawing.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2559) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle.

See Also

CreateParams (see page 2559)

CreateWnd (see page 2559)

CreateWindowHandle

7.1.6.3.2 TCustomFrame.GetChildren Method

Calls a specified method for each child of the frame.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the component streaming system that loads and saves forms and data modules. Applications seldom need to call this routine.

Proc is a callback to execute for every component in the frame.

Root indicates the top-level component that is the entry point for the entire streaming process. This can be a form, data module, or frame.

GetChildren executes the method specified by Proc for every component contained by the frame. It provides extra support for components that do not have parents if the Root parameter identifies this frame.

See Also

GetChildren

7.1.6.3.3 TCustomFrame.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler

PaintTo

DoubleBuffered

7.1.6.3.4 TCustomFrame.SetParent Method

Sets the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. TCustomFrame (see page 2237) overrides the method inherited from TControl to

Update the parent form so that it is aware of any action lists in the frame.

Regenerate the frame's window so that it fits properly into its new parent.

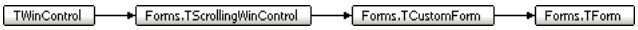
See Also

Parent

7.1.7 TForm Class

TForm represents a standard application window (form).

Class Hierarchy



Pascal

```
TForm = class(TCustomForm);
```

C++

```
class TForm : public TCustomForm;
```

File

Forms

Description

When you create forms in the Form designer at design time, they are implemented as descendants of TForm. Forms (see page 1921) can represent the application's main window, or dialog boxes, or MDI children. A form can contain other objects, such as TButton, TCheckBox, and TComboBox objects.

Examples of forms include TLoginDialog and TPasswordDialog objects.

See Also

TLoginDialog

TPasswordDialog

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Adding Forms

Controlling When Forms Reside in Memory

Creating Forms Dynamically

Creating Modeless Forms Such as Windows

Displaying an Auto-created Form

Using Forms

Passing Additional Arguments to Forms

Retrieving Data from Forms

Retrieving Data from Modal Forms

Retrieving Data from Modeless Forms

Controlling Application Behavior

Creating a Form Instance Using a Local Variable

Using the Main Form

VCL Overview

Windows Overview

7.1.7.1 TForm Members

The following tables list the members exposed by TForm.

Protected Fields

	Name	Description
	FFormState (see page 2110)	This is FFormState, a member of class TCustomForm.

Protected Methods

	Name	Description
	Activate (see page 2111)	Activation event dispatcher.
	ActiveChanged (see page 2112)	Active (see page 2145) state event dispatcher.
V	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
V	AlignControls (see page 2113)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
V	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
V	BeginAutoDrag (see page 2113)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
V	ChangeScale (see page 2114)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
V	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CloseModal (see page 2115)	This is CloseModal, a member of class TCustomForm.
V	CreateParams (see page 2116)	Initializes the window-creation parameter record when the form window is created.
V	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
V	CreateWindowHandle (see page 2116)	Creates a form window given a set of window-creation parameters.
V	CreateWnd (see page 2117)	Creates the form window.
V	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	Deactivate (see page 2117)	Generates an OnDeactivate (see page 2209) event.
V	DefineProperties (see page 2118)	Designates methods for storing a form's unpublished data on a stream such as a form file.
V	DestroyHandle (see page 2119)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 2119)	Destroys the window created in the CreateWindowHandle (see page 2116) method.
	DoClose (see page 2120)	Form closing event dispatcher.
V	DoCreate (see page 2120)	Form creation event dispatcher.
V	DoDestroy (see page 2121)	Form destruction event dispatcher.
V	DoDock (see page 2121)	Docking event dispatcher.
V	DoFlipChildren (see page 2560)	Flips the control's children.
	DoHide (see page 2122)	Form hide event dispatcher.
	DoShow (see page 2122)	Form show event dispatcher.
	get_ActiveMDIChild (see page 2123)	This is get_ActiveMDIChild, a member of class TCustomForm.
	get_MDIChildCount (see page 2123)	This is get_MDIChildCount, a member of class TCustomForm.

	get_MDIChildren (see page 2123)	This is get_MDIChildren, a member of class TCustomForm.
	GetBorderIconStyles (see page 2123)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 2124)	This is GetBorderStyles, a member of class TCustomForm.
V	GetClientRect (see page 2124)	Returns the value of the ClientRect property.
V	GetFloating (see page 2125)	Returns the value of the Floating property.
	GetOwnerWindow (see page 2127)	This is GetOwnerWindow, a member of class TCustomForm.
	HandleCreateException (see page 2127)	Handles exceptions that are raised by the OnCreate (see page 2207) event handler.
	InitializeNewForm (see page 2128)	This is InitializeNewForm, a member of class TCustomForm.
V	Loaded (see page 2129)	Performs fix-ups when the form is first loaded into memory.
V	Notification (see page 2130)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 2131)	Generates an OnPaint (see page 2233) event.
V	PaintWindow (see page 2131)	Renders the image of the form.
V	PaletteChanged (see page 2132)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
V	QueryInterface (see page 2132)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2133)	Reads the form's published properties, stored data, and owned components from a reader object.
V	RequestAlign (see page 2134)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
V	Resizing (see page 2134)	Makes any required adjustments when the form changes size.
V	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
V	SetChildOrder (see page 2135)	Changes the order in which Child appears when GetChildren (see page 2124) is called.
V	SetParent (see page 2136)	Sets the Parent property.
V	SetParentBiDiMode (see page 2136)	Sets the ParentBiDiMode property.
V	UpdateActions (see page 2140)	Updates all actions associated with the form.
	UpdateWindowState (see page 2141)	Updates the WindowState (see page 2195) property to reflect the current state of the form.
V	ValidateRename (see page 2141)	Determines if a form can rename one of its owned components.
V	VisibleChanging (see page 2141)	Responds just before the form changes the value of its Visible (see page 2192) property.
V	WndProc (see page 2142)	Provides specific message responses for the form.

Protected Properties

	Name	Description
R	PopupChildren (see page 2179)	This is PopupChildren, a member of class TCustomForm.

Public Methods

	Name	Description
V	AfterConstruction (see page 2112)	Construction event dispatcher.
	ArrangeIcons (see page 2247)	Arranges the icons of minimized MDI child forms.
V	BeforeDestruction (see page 2113)	Destruction event dispatcher.
	Cascade (see page 2248)	Arranges MDI child forms so they overlap.
	Close (see page 2114)	Closes the form.
V	CloseQuery (see page 2115)	Close (see page 2114) attempt event dispatcher.
V	Create (see page 2108)	Creates and initializes a new TForm (see page 2241) object.
V	Create (see page 2555)	Creates and initializes a new TScrollingWinControl (see page 2554) object.
V	CreateNew (see page 2108)	Creates and initializes a new form.
V	DefaultHandler (see page 2118)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 2118)	Removes focus from a control on the form.
V	Destroy (see page 2109)	Removes the form from memory.
V	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
V	Dock (see page 2120)	Handles automatic docking of the form into other windows.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.

	FocusControl (see page 2122)	Sets focus to a control on the form.
V	GetChildren (see page 2124)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetFormImage (see page 2125)	Returns a bitmap of the form.
	Hide (see page 2127)	Hides the form.
	IsShortCut (see page 2128)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 2129)	Ensures that the form is fully visible on a specified monitor.
V	MouseWheelHandler (see page 2130)	Directs mouse wheel messages to the appropriate control.
	Next (see page 2249)	Activates the next child form in the form sequence.
	Previous (see page 2249)	Activates the previous MDI child form in the form sequence.
	Print (see page 2132)	Prints the form.
	RecreateAsPopup (see page 2133)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 2133)	Destroys the form and frees its associated memory.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 2134)	Cancels modes on the form.
	set_PopupParent (see page 2135)	This is set_PopupParent, a member of class TCustomForm.
V	SetFocus (see page 2135)	Sets focus to the form.
V	SetFocusedControl (see page 2136)	Sets focus to a control on the form.
	Show (see page 2137)	Shows the form.
V	ShowModal (see page 2139)	Shows a form as a modal dialog.
	Tile (see page 2249)	Arranges MDI child forms so that they are all the same size.
V	WantChildKey (see page 2142)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
R	Active (see page 2145)	Specifies whether the form has focus.
	ActiveMDIChild (see page 2254)	Specifies the MDI child that has focus.
	ActiveOleControl (see page 2146)	Specifies the OLE control on the form that reacts to changes in focus.
R	Canvas (see page 2150)	Provides access to the drawing area of the form.
	ClientHandle (see page 2264)	Provides access to the handle of the form's internal MDI client window.
	Designer (see page 2159)	Specifies the designer interface for the form.
	DockManager (see page 2270)	Specifies the control's docking manager interface.
	DropTarget (see page 2159)	Specifies whether the form is the target of a drag-and-drop operation.
R	FormState (see page 2161)	Indicates transitional state information about the form.
	MDIChildCount (see page 2288)	Specifies the number of open MDI child forms.
	MDIChildren (see page 2289)	Provides indexed access to all MDI child forms.
	ModalResult (see page 2176)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 2177)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 2178)	Specifies the IOleForm (see page 2572) interface for an in-place OLE object contained in the form.
	TileMode (see page 2304)	Represents how MDI child forms are arranged when the Tile method is called.

Published Events

	Name	Description
	OnActivate (see page 2312)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 2318)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2318)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 2319)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2319)	Occurs when the user clicks the control.
	OnClose (see page 2321)	Occurs when the form closes.
	OnCloseQuery (see page 2323)	Occurs when close is attempted.
	OnConstrainedResize (see page 2324)	Adjust resize constraints.
	OnContextPopup (see page 2324)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 2325)	Occurs when the form is created.

	OnDblClick (see page 2327)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2327)	Occurs when the form loses focus.
	OnDestroy (see page 2327)	Occurs when the form is destroyed.
	OnDockDrop (see page 2329)	Occurs when another control is docked to the control.
	OnDockOver (see page 2331)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2333)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2336)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2339)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 2339)	Returns the control's docking information.
	OnHelp (see page 2341)	Occurs when the form receives a request for help.
	OnHide (see page 2342)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 2344)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2347)	Occurs when a key is pressed.
	OnKeyUp (see page 2348)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2350)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2351)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2353)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2354)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2354)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2356)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2359)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2359)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2359)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 2360)	Occurs when the form is redrawn.
	OnResize (see page 2361)	Occurs immediately after the control is resized.
	OnShortCut (see page 2362)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 2362)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 2363)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 2364)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 2252)	Designates the action associated with the control.
	ActiveControl (see page 2253)	Specifies the control that has focus on the form.
	Align (see page 2254)	Determines how the control aligns within its container (parent control).
	AlphaBlend (see page 2257)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2258)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 2258)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2259)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2259)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 2260)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 2260)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 2261)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 2262)	Specifies the width of the control's border.
	Caption (see page 2262)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 2265)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 2266)	Specifies the width (in pixels) of the form's client area.
	Color (see page 2268)	Specifies the background color of the control.

	Constraints (see page 2269)	Specifies the size constraints for the control.
	Ctl3D (see page 2269)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2270)	Specifies the monitor on which the form appears.
	DockSite (see page 2272)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2273)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragKind (see page 2275)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2275)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2277)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2277)	Controls the attributes of text written on or in the control.
	FormStyle (see page 2279)	Determines the form's style.
	GlassFrame (see page 2281)	This is GlassFrame, a member of class TForm.
	Height (see page 2281)	Specifies the vertical size of the control in pixels.
	HelpFile (see page 2282)	Specifies the name of the file the form uses to display Help.
	HorzScrollBar (see page 2283)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 2284)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 2284)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 2171)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Menu (see page 2290)	Specifies the form's main menu.
	ObjectMenuItem (see page 2292)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 2292)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 2292)	Specifies the padding of a control.
	ParentBiDiMode (see page 2293)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2293)	Determines where a control looks for its font information.
	PixelsPerInch (see page 2294)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 2294)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 2299)	Controls how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 2300)	Sets an order for stacked forms that users cannot change.
	Position (see page 2300)	Represents the size and placement of the form.
	PrintScale (see page 2301)	Represents the proportions of a printed form.
	Scaled (see page 2302)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 2302)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 2302)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 2304)	Specifies distance for screen snap.
	Top (see page 2189)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 2305)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2306)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 2306)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 2306)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 2307)	Indicates whether the form is visible.
	Width (see page 2308)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 2310)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 2310)	Represents how the form appears on the screen.

7.1.7.2 TForm Methods

The methods of the TForm class are listed here.

Public Methods

	Name	Description
	ArrangeIcons (see page 2247)	Arranges the icons of minimized MDI child forms.
	Cascade (see page 2248)	Arranges MDI child forms so they overlap.
	Next (see page 2249)	Activates the next child form in the form sequence.
	Previous (see page 2249)	Activates the previous MDI child form in the form sequence.
	Tile (see page 2249)	Arranges MDI child forms so that they are all the same size.

7.1.7.2.1 TForm.ArrangeIcons Method

Arranges the icons of minimized MDI child forms.

Pascal

```
procedure ArrangeIcons;
```

C++

```
__fastcall ArrangeIcons();
```

Description

Use ArrangeIcons to arrange the icons of minimized MDI child forms so they are evenly spaced and don't overlap.

ArrangeIcons works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 2279) property is fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
```

See Also

Cascade (see page 2248)

Tile (see page 2249)

FormStyle (see page 2162)

7.1.7.2.2 TForm.Cascade Method

Arranges MDI child forms so they overlap.

Pascal

```
procedure Cascade;
```

C++

```
__fastcall Cascade();
```

Description

Use Cascade to arrange MDI child forms so they overlap.

Cascade works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 2279) property is fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to fsMDIForm. The others have their
FormStyle properties set to fsMDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyCascade. This is code for the
MyCascadeClick handler.  When the user chooses the
Cascade command, restored forms are arranged
so that they overlap.  DO NOT name the menu items "Cascade"
as that will override the TForm method.
*/
void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to fsMDIForm. The others have their
FormStyle properties set to fsMDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyCascade. This is code for the
MyCascadeClick handler.  When the user chooses the
Cascade command, restored forms are arranged
so that they overlap.  DO NOT name the menu items "Cascade"
as that will override the TForm method.
}
procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

Tile (see page 2249)

ArrangeIcons (see page 2247)

FormStyle (see page 2162)

7.1.7.2.3 TForm.Next Method

Activates the next child form in the form sequence.

Pascal

```
procedure Next;
```

C++

```
__fastcall Next();
```

Description

Use Next to change the active child form of an MDI parent. For example, if you have three child forms within a parent form and Form2 is the active form, the Next method makes Form3 the active form. Calling Next again makes Form4 active. The next time your application calls Next, the sequence starts over again and Form2 becomes the active form once again.

The Next method applies only to forms that are MDI parent forms (have a FormStyle (see page 2279) property value of fsMDIForm).

See Also

Previous (see page 2249)

FormStyle (see page 2162)

7.1.7.2.4 TForm.Previous Method

Activates the previous MDI child form in the form sequence.

Pascal

```
procedure Previous;
```

C++

```
__fastcall Previous();
```

Description

Use Previous to change the active child form of an MDI parent. For example, if you have three child forms within a parent form and Form4 is the active form, the Previous method makes Form3 the active form. Calling Previous again makes Form2 active. The next time your application calls Previous, the sequence starts over again and Form4 becomes the active form once again.

The Previous method applies only to forms that are MDI parent forms (have a FormStyle (see page 2279) property value of fsMDIForm).

See Also

Next (see page 2249)

7.1.7.2.5 TForm.Tile Method

Arranges MDI child forms so that they are all the same size.

Pascal

```
procedure Tile;
```

C++

```
__fastcall Tile();
```

Description

Use Tile to arrange MDI child forms so that they are all the same size. Tiled forms completely fill up the client area of the parent form. How the forms arrange themselves depends upon the values of their TileMode (see page 2304) properties.

Tile works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 2279) property is fsMDIForm).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

TileMode (see page 2188)

7.1.7.3 TForm Properties

The properties of the TForm class are listed here.

Public Properties

	Name	Description
	ActiveMDIChild (see page 2254)	Specifies the MDI child that has focus.
	ClientHandle (see page 2264)	Provides access to the handle of the form's internal MDI client window.
	DockManager (see page 2270)	Specifies the control's docking manager interface.
	MDIChildCount (see page 2288)	Specifies the number of open MDI child forms.
	MDIChildren (see page 2289)	Provides indexed access to all MDI child forms.
	TileMode (see page 2304)	Represents how MDI child forms are arranged when the Tile method is called.

Published Properties

	Name	Description
	Action (see page 2252)	Designates the action associated with the control.
	ActiveControl (see page 2253)	Specifies the control that has focus on the form.
	Align (see page 2254)	Determines how the control aligns within its container (parent control).
	AlphaBlend (see page 2257)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 2258)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 2258)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2259)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2259)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 2260)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 2260)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 2261)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 2262)	Specifies the width of the control's border.
	Caption (see page 2262)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 2265)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 2266)	Specifies the width (in pixels) of the form's client area.
	Color (see page 2268)	Specifies the background color of the control.
	Constraints (see page 2269)	Specifies the size constraints for the control.
	Ctl3D (see page 2269)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 2270)	Specifies the monitor on which the form appears.
	DockSite (see page 2272)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2273)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragKind (see page 2275)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2275)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2277)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2277)	Controls the attributes of text written on or in the control.
	FormStyle (see page 2279)	Determines the form's style.
	GlassFrame (see page 2281)	This is GlassFrame, a member of class TForm.
	Height (see page 2281)	Specifies the vertical size of the control in pixels.
	HelpFile (see page 2282)	Specifies the name of the file the form uses to display Help.
	HorzScrollBar (see page 2283)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 2284)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 2284)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 2290)	Specifies the form's main menu.
	ObjectMenuItem (see page 2292)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 2292)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 2292)	Specifies the padding of a control.
	ParentBiDiMode (see page 2293)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2293)	Determines where a control looks for its font information.
	PixelsPerInch (see page 2294)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 2294)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 2299)	Controls how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 2300)	Sets an order for stacked forms that users cannot change.
	Position (see page 2300)	Represents the size and placement of the form.
	PrintScale (see page 2301)	Represents the proportions of a printed form.
	Scaled (see page 2302)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 2302)	Specifies whether form snaps to edge of screen.

	ShowHint (see page 2302)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 2304)	Specifies distance for screen snap.
	TransparentColor (see page 2305)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 2306)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 2306)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 2306)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 2307)	Indicates whether the form is visible.
	Width (see page 2308)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 2310)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 2310)	Represents how the form appears on the screen.

7.1.7.3.1 TForm.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

7.1.7.3.2 TForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 2146)

OnExit

OnEnter

Active (see page 2145)

7.1.7.3.3 TForm.ActiveMDIChild Property

Specifies the MDI child that has focus.

Pascal

```
property ActiveMDIChild: TForm;
```

C++

```
__property TForm * ActiveMDIChild;
```

Description

Use ActiveMDIChild to get the MDI child that has focus.

If the form is not an MDI parent (that is, if the FormStyle property of the form is not fsMDIForm), ActiveMDIChild returns nil (Delphi) or NULL (C++).

See Also

FormStyle (see page 2162)

MDIChildCount (see page 2172)

7.1.7.3.4 TForm.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
```

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
```

```
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

7.1.7.3.5 TForm.AlphaBlend Property

Specifies whether the form is translucent.

Pascal

```
property AlphaBlend: Boolean;
```

C++

```
__property Boolean AlphaBlend;
```

Description

Set AlphaBlend to specify that the form represents a layered window that allows a translucent color. The AlphaBlendValue property specifies the degree of translucency.

Note: AlphaBlend does not work on all systems. The application must be running under Windows 2000 or better, and the machine must be a P90 or better.

See Also

AlphaBlendValue (see page 2147)

TransparentColor (see page 2190)

7.1.7.3.6 TForm.AlphaBlendValue Property

Specifies the degree of translucency on a translucent form.

Pascal

```
property AlphaBlendValue: Byte;
```

C++

```
__property Byte AlphaBlendValue;
```

Description

Set AlphaBlendValue to a value between 0 and 255 to indicate the degree of translucency when the AlphaBlend property is true. A value of 0 indicates a completely transparent window. A value of 255 indicates complete opacity.

Note: AlphaBlendValue only has an effect when the AlphaBlend property is true.

See Also

AlphaBlend (see page 2147)

TransparentColor (see page 2190)

TransparentColorValue (see page 2191)

7.1.7.3.7 TForm.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at

design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.7.3.8 TForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.7.3.9 TForm.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

7.1.7.3.10 TForm.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

7.1.7.3.11 TForm.BorderIcons Property

Specifies which icons appear on the title bar of the form.

Pascal

```
property BorderIcons: TBorderIcons;
```

C++

```
__property TBorderIcons BorderIcons;
```

Description

Use BorderIcons to get or set the icons that appear on the title bar of the form. BorderIcons can include any of the following TBorderIcons values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).
biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise,no question mark appears.

Note: Certain combinations of the BorderIcons and BorderStyle properties are mutually exclusive. For example, BorderIcons biMax, biMin with BorderStyle of bsDialog are mutually exclusive.

C++ Examples:

```
/*
The following code removes a form's Maximize button when the
user clicks a button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TBorderIcons tempBI = BorderIcons;
  tempBI >> biMaximize;
  BorderIcons = tempBI;
}
```

Delphi Examples:

```
{
The following code removes a form's Maximize button when the
user clicks a button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
BorderIcons := BorderIcons - [biMaximize];
end;
```

See Also

BorderStyle (see page 2149)

7.1.7.3.12 TForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 2148)

7.1.7.3.13 TForm.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth;
```

C++

```
__property BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter

7.1.7.3.14 TForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
```

```
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
```

```
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

7.1.7.3.15 TForm.ClientHandle Property

Provides access to the handle of the form's internal MDI client window.

Pascal

```
property ClientHandle: HWND;
```

C++

```
__property HWND ClientHandle;
```

Description

Use ClientHandle to get the handle for the form's internal MDI client window.

ClientHandle is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

Note: Use the Handle property instead to get the handle to the form window itself.

See Also

FormStyle (see page 2162)

Handle

7.1.7.3.16 TForm.ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Pascal

```
property ClientHeight;
```

C++

```
__property ClientHeight;
```

Description

Use ClientHeight to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set ClientHeight to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the Height property instead.

C++ Examples:

```
/*
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form1->ClientHeight = Form1->ClientHeight/2;
}
```

Delphi Examples:

```
{
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientHeight := ClientHeight div 2;
end;
```

See Also

ClientRect

ClientWidth (see page 2155)

Height

7.1.7.3.17 TForm.ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Pascal

```
property ClientWidth;
```

C++

```
__property ClientWidth;
```

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width property instead.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
```

```
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
```

```
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientHeight (see page 2154)

ClientRect

Width

7.1.7.3.18 TForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

7.1.7.3.19 TForm.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

7.1.7.3.20 TForm.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

7.1.7.3.21 TForm.DefaultMonitor Property

Specifies the monitor on which the form appears.

Pascal

```
property DefaultMonitor: TDefaultMonitor;
```

C++

```
__property TDefaultMonitor DefaultMonitor;
```

Description

Use DefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

Note: DefaultMonitor has no effect if the application does not have a main form.

See Also

Monitors (see page 2474)

TMonitor (see page 2439)

Screen (see page 2615)

MainForm (see page 1983)

Active (see page 2145)

Monitor (see page 2177)

MakeFullyVisible (see page 2129)

7.1.7.3.22 TForm.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager;
```

C++

```
__property DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a

default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
```

```
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

UseDockManager

DefaultDockTreeClass

7.1.7.3.23 TForm.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

7.1.7.3.24 TForm.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
```

```
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

7.1.7.3.25 TForm.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

7.1.7.3.26 TForm.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

7.1.7.3.27 TForm.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

7.1.7.3.28 TForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

7.1.7.3.29 TForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example ensures that the main form of the application
is an MDI parent:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  if(FormStyle != fsMDIForm)
          FormStyle = fsMDIForm;
  if(FormStyle == fsMDIForm)
          Edit1->Text = "MDI form";
  else
```

```
    //This line never executes
    Edit1->Text = "Not an MDI form";
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example ensures that the main form of the application
is an MDI parent:
}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

constructor TForm1.Create(AOwner : TComponent);

begin
  inherited Create(AOwner);
  if FormStyle <> fsMDIForm then
          FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
          Edit1.Text := 'MDI form'
  else
```

```
    //This line never executes
    Edit1.Text := 'Not an MDI form';
end;
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

ArrangeIcons (see page 2247)

Tile (see page 2249)

Cascade (see page 2248)

Windows Overview

7.1.7.3.30 TForm.GlassFrame Property

Pascal

```
property GlassFrame: TGlassFrame;
```

C++

```
__property TGlassFrame GlassFrame;
```

Description

This is GlassFrame, a member of class TForm.

7.1.7.3.31 TForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
```

```
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

7.1.7.3.32 TForm.HelpFile Property

Specifies the name of the file the form uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property UnicodeString HelpFile;
```

Description

Use HelpFile for forms that have a help file to use with the Help system. Help topics are selected from the specified file.

See Also

HelpContext

OnHelp (see page 2217)

HelpFile (see page 1974)

HelpKeyword

7.1.7.3.33 TForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
```

```
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 2561)

VertScrollBar (see page 2563)

TControlScrollBar (see page 2009)

7.1.7.3.34 TForm.Icon Property

Specifies the icon that appears when the form is minimized.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Set the Icon property to provide an icon for the form. If Icon is not set, the system provides a default icon when the form is minimized.

C++ Examples:

```
/*
This code assigns an icon to a form when the form is created:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Icon->LoadFromFile("..\\MDIChild1.ICO");
}
```

Delphi Examples:

```
{
This code assigns an icon to a form when the form is created:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MDIChild1.ICO');
end;
```

See Also

WindowState (see page 2195)

7.1.7.3.35 TForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

ActiveControl (see page 2145)

OnKeyDown

OnKeyUp

OnKeyPress

7.1.7.3.36 TForm.MDIChildCount Property

Specifies the number of open MDI child forms.

Pascal

```
property MDIChildCount: Integer;
```

C++

```
__property int MDIChildCount;
```

Description

Use MDIChildCount to get the number of open MDI child forms.

MDIChildCount is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
*/
void __fastcall TForm1::CloseChildrenClick(TObject *Sender)
{
  for(int i = MDIChildCount - 1; i >= 0; i--)
    MDIChildren[i]->Close();
}

void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}

void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
  }
```

Delphi Examples:

```
{
```

```
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildren (see page 2173)

7.1.7.3.37 TForm.MDIChildren Property

Provides indexed access to all MDI child forms.

Pascal

```
property MDIChildren [I: Integer]: TForm;
```

C++

```
__property TForm * MDIChildren[int I];
```

Description

Use MDIChildren to access a child form.

I is the index of the child form to access. The order of MDI children changes as different MDI forms are activated. Initially, forms are listed in creation order, with new forms being added to the end of the array. Whenever an MDI child is activated, however, it moves to the front of the list. Thus, use I to index over all forms in the array, but not to refer to a specific form.

MDIChildren is meaningful only if the form is an MDI frame (that is, if the form's FormStyle property is set to fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
*/
void __fastcall TForm1::CloseChildrenClick(TObject *Sender)
```

```
{
  for(int i = MDIChildCount - 1; i >= 0; i--)
    MDIChildren[i]->Close();
}

void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}

void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
  }
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildCount (see page 2172)

FormStyle (see page 2162)

7.1.7.3.38 TForm.Menu Property

Specifies the form's main menu.

Pascal

```
property Menu: TMainMenu;
```

C++

```
__property TMainMenu Menu;
```

Description

Use Menu to get or set the form's main menu. At design time, Menu is set to the first TMainMenu component added to the form.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.  Click on the menu item
or type "Ctrl+N" to execute MyOnExecute.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

TMainMenu (see page 4828)

7.1.7.3.39 TForm.ObjectMenuItem Property

Represents an OLE object menu item that reacts to selections of OLE objects.

Pascal

```
property ObjectMenuItem: TMenuItem;
```

C++

```
__property TMenuItem ObjectMenuItem;
```

Description

Use ObjectMenuItem to get or set a menu item that becomes enabled or disabled when an OLE object on the form is selected or unselected respectively.

The OLE object menu item can be used for standard OLE commands, such as Activate or Convert. (The implementations of these commands are provided by the OLE server.)

See Also

OleFormObject (see page 2178)

7.1.7.3.40 TForm.OldCreateOrder Property

Specifies when OnCreate and OnDestroy events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate event occurs after all constructors are finished and the OnDestroy event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate event occurred when the TCustomForm constructor executed and the OnDestroy event occurred when the TCustomForm destructor executed. Applications that require the OnCreate event and OnDestroy events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 2207)

OnDestroy (see page 2209)

BeforeDestruction (see page 2113)

AfterConstruction (see page 2112)

7.1.7.3.41 TForm.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

7.1.7.3.42 TForm.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

7.1.7.3.43 TForm.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

7.1.7.3.44 TForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 2187)

7.1.7.3.45 TForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
```

```
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

7.1.7.3.46 TForm.PopupMode Property

Controls how the top-level form behaves with respect to Window's WS_POPUP style.

Pascal

```
property PopupMode: TPopupMode;
```

C++

```
__property TPopupMode PopupMode;
```

Description

The PopupMode property controls how the top-level form behaves with respect to Window's WS_POPUP style. A window that has the WS_POPUP style is always above its "owner" in Z-order. You can use the PopupMode property in conjunction with the PopupParent (see page 2300) property to avoid the appearance of a hung application caused by a modal dialog showing up behind another form on the screen.

Note: Using the PopupMode property is similar to specifying fsStayOnTop for the FormStyle (see page 2279) property, but allows better control over the layering.

The PopupMode property is automatically set to pmAuto when the ShowModal (see page 2139) method is called. However, this causes the window handle to be recreated, which is not usually desirable. To avoid the re-creation of window handles, you can explicitly set the PopupMode property to pmAuto prior to calling the ShowModal (see page 2139) method (such as at design time).

Set the PopupMode property to pmExplicit for non-modal design windows such as tool palettes and other floating tool windows. This causes the windows to always remain on top of the main form. In the case of a snapped together design (in which the window is docked to the main form), the design window remains on top of the designer.

Note: For undocked windows, the design window remains above the main form, allowing other top-level forms to obscure the design window.

Set thePopupMode property to pmNone if you want the popup window to have the pre-Delphi 8 behavior (except for the ShowModal (see page 2139) item mentioned above).

You can force the Z-Ordering of your forms and create a "stacked" appearance that the user cannot change by setting the PopupParent (see page 2300) property to an explicit TCustomForm (see page 2103). If the PopupMode property is pmExplicit and PopupParent (see page 2300) is nil, then the Application.MainForm is implicitly used as the PopupParent (see page 2300). If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent (see page 2300). If the PopupMode property is set to pmAuto, Screen.ActiveForm is used as the PopupParent (see page 2300).

7.1.7.3.47 TForm.PopupParent Property

Sets an order for stacked forms that users cannot change.

Pascal

```
property PopupParent: TCustomForm;
```

C++

```
__property TCustomForm PopupParent;
```

Description

If the PopupMode property is set to pmExplicit and PopupParent is nil, then the Application.MainForm is implicitly used as the PopupParent. If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent.

If the PopupMode property is set to pmAuto, Screen.ActiveForm is used as the PopupParent property.

7.1.7.3.48 TForm.Position Property

Represents the size and placement of the form.

Pascal

```
property Position: TPosition;
```

C++

```
__property TPosition Position;
```

Description

Use Position to get or set the size and placement of the form. Position can have one of the following TPosition values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.
poDefault	The form appears in a position on the screen and with a height and width determined by the operating system. Each time you run the application, the form moves slightly down and to the right. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.

poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen. Each time you run the application, the form moves slightly down and to the right. When the form can no longer move down and to the right and keep the same size while remaining entirely visible on the screen, the form displays at the top-left corner of the screen. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. In multi-monitor applications, the form may be moved from this center position so that it falls entirely on one monitor, as specified by the DefaultMonitor property.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. No adjustments are made for multi-monitor applications.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. No adjustments are made for multi-monitor applications. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

OnResize

SetInitialBounds

7.1.7.3.49 TForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 2132)

7.1.7.3.50 TForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 2179)

Considerations When Dynamically Resizing Forms and Controls

7.1.7.3.51 TForm.ScreenSnap Property

Specifies whether form snaps to edge of screen.

Pascal

```
property ScreenSnap: Boolean;
```

C++

```
__property Boolean ScreenSnap;
```

Description

ScreenSnap determines whether the edges of the form snap to the edge of the screen when the user moves the form. To control the snap distance, set the SnapBuffer property.

See Also

SnapBuffer (see page 2188)

7.1.7.3.52 TForm.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

7.1.7.3.53 TForm.SnapBuffer Property

Specifies distance for screen snap.

Pascal

```
property SnapBuffer: Integer;
```

C++

```
__property int SnapBuffer;
```

Description

If ScreenSnap is true, SnapBuffer determines the maximum number of pixels that be between the form edge and the screen edge before the form snaps to the screen edge.

See Also

ScreenSnap (see page 2187)

7.1.7.3.54 TForm.TileMode Property

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
property TileMode: TTileMode;
```

C++

```
__property TTileMode TileMode;
```

Description

TileMode can have one of the following TTileMode values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form
tbVertical	Each form stretches along the height of the parent form

TileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle property is set to fsMDIChild).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

FormStyle (see page 2162)

Tile (see page 2249)

7.1.7.3.55 TForm.TransparentColor Property

Specifies whether a color on the form appears transparent.

Pascal

```
property TransparentColor: Boolean;
```

C++

```
__property Boolean TransparentColor;
```

Description

Use TransparentColor to indicate that one of the colors on the form should be treated as transparent, allowing windows behind the form to completely show through. The TransparentColorValue property indicates the color that appears completely transparent.

Note: To make the entire form transparent, or to make it translucent rather than transparent, use the AlphaBlend and AlphaBlendValue properties.

Note: TransparentColor does not work on all systems. The application must be running under Windows 2000 or better, with adequate hardware.

See Also

TransparentColorValue (see page 2191)

AlphaBlend (see page 2147)

AlphaBlendValue (see page 2147)

7.1.7.3.56 TForm.TransparentColorValue Property

Indicates the color on the form that appears transparent when TransparentColor is true.

Pascal

```
property TransparentColorValue: TColor;
```

C++

```
__property TColor TransparentColorValue;
```

Description

Use TransparentColorValue to indicate the color that appears transparent when the TransparentColor property is true.

Note: If the application uses a 16-bit color palette, TransparentColorValue must be a basic color.

See Also

TransparentColor (see page 2190)

7.1.7.3.57 TForm.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager;
```

C++

```
__property UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager

7.1.7.3.58 TForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

TControlScrollBar (see page 2009)

7.1.7.3.59 TForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show and ShowModal methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront

Show (see page 2137)

ShowModal (see page 2139)

7.1.7.3.60 TForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
```

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

ClientWidth

Height

Left

7.1.7.3.61 TForm.WindowMenu Property

Specifies the Window menu for an MDI parent form.

Pascal

```
property WindowMenu: TMenuItem;
```

C++

```
__property TMenuItem WindowMenu;
```

Description

Use WindowMenu to get or set the Window menu for an MDI parent form. The Window menu is a standard menu in MDI applications. It contains commands that let the user manage the windows in the application. Menu items usually include Cascade, Arrange Icons, Tile, and so on.

The Window menu also lists (at the bottom) the child windows that are currently open in the application. When the user selects one of these windows from the menu, the window becomes the active window in the application.

Although this menu is commonly called the Window menu, it can have any name that corresponds to an existing menu item on the menu bar. The names of open child forms are merged onto the menu automatically at run-time.

WindowMenu is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

See Also

FormStyle (see page 2162)

7.1.7.3.62 TForm.WindowState Property

Represents how the form appears on the screen.

Pascal

```
property WindowState: TWindowState;
```

C++

```
__property TWindowState WindowState;
```

Description

Set WindowState to minimize, maximize, or restore the form window. Read WindowState to determine whether the form is minimized, maximized, or in a normal state.

See Also

Icon (see page 2167)

BorderStyle (see page 2149)

7.1.7.4 TForm Events

The events of the TForm class are listed here.

Published Events

	Name	Description
	OnActivate (see page 2312)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 2318)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2318)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 2319)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2319)	Occurs when the user clicks the control.
	OnClose (see page 2321)	Occurs when the form closes.
	OnCloseQuery (see page 2323)	Occurs when close is attempted.
	OnConstrainedResize (see page 2324)	Adjust resize constraints.
	OnContextPopup (see page 2324)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 2325)	Occurs when the form is created.
	OnDblClick (see page 2327)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 2327)	Occurs when the form loses focus.
	OnDestroy (see page 2327)	Occurs when the form is destroyed.
	OnDockDrop (see page 2329)	Occurs when another control is docked to the control.
	OnDockOver (see page 2331)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2333)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2336)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2339)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 2339)	Returns the control's docking information.
	OnHelp (see page 2341)	Occurs when the form receives a request for help.
	OnHide (see page 2342)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 2344)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2347)	Occurs when a key is pressed.
	OnKeyUp (see page 2348)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2350)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2351)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2353)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2354)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2354)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2356)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2359)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2359)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2359)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 2360)	Occurs when the form is redrawn.
	OnResize (see page 2361)	Occurs immediately after the control is resized.
	OnShortCut (see page 2362)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 2362)	Occurs when the form is shown (that is, when its Visible property is set to true).

	OnStartDock (see page 2363)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 2364)	Occurs when the application tries to undock a control that is docked to the windowed control.

7.1.7.4.1 TForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication), not the form, occurs when Windows switches control from another application.

C++ Examples:

```
/*
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
*/

TForm1 *Form1;
TTable *Customers, *Blobs;
Boolean Form1Done = False;

void CreateCustSource()
{
  Customers = new TTable(Form1); // The owner will clean this up.
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
```

```
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  Form1->CustomerDS->DataSet = Customers;
  Customers->Active = True;
}

void CreateBlobSource()
{
  int i;
  Blobs = new TTable(Form1);
  Blobs->DatabaseName = "DBDEMOS";
  Blobs->TableType = ttParadox;
  Blobs->TableName = "MyBlobInfo";
  Blobs->Active = False;
    // Don't overwrite an existing table
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
  if (Blobs->Exists)
  {
    Blobs->Close();
    Blobs->DeleteTable();
  };
  // The Table component must not be active }
  // First, describe the type of table and give }
  // it a name }
  // Next, describe the fields in the table }
  Blobs->FieldDefs->Clear();
  TFieldDef *newDef = Blobs->FieldDefs->AddFieldDef();
  newDef->Name = "Field1";
  newDef->DataType = ftInteger;
  newDef->Required = True;
  newDef = Blobs->FieldDefs->AddFieldDef();
  newDef->Name = "Field2";
  newDef->DataType = ftBlob;
  newDef->Size = 30;
  // Next, describe any indexes }
  Blobs->IndexDefs->Clear();
  // The 1st index has no name because it is
  // a Paradox primary key }
  TIndexDef *newIDef = Blobs->IndexDefs->AddIndexDef();
  newIDef->Name = "";
  newIDef->Fields = "Field1";
  newIDef->Options << ixPrimary;
/*
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
*/
  // Call the CreateTable method to create the table }
  Blobs->CreateTable();
  Blobs->Active = True;
  for (int i = 1; i <= 20; i++)
    Blobs->AppendRecord(ARRAYOFCONST((i*111, IntToStr(i*222))));
  Form1->BlobDS->DataSet = Blobs;
```

```
  Blobs->Active = True;
};


void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CreateCustSource();
  CreateBlobSource();
  for (int i = 0; i < ComponentCount; i++)
  {
    if (Components[i]->ClassNameIs("TDataSource"))
      ListBox1->Items->AddObject(Components[i]->Name,
         dynamic_cast<TObject *>(Components[i]));
  }
}

void __fastcall TForm1::ListBox1MouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift,
          int X, int Y)
{
  Form1Done = True;
}
/*
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it. Also, set the visible property
of Form2 to true.
*/
void __fastcall TForm2::FormActivate(TObject *Sender)
{
  if (Form1Done)
  {
    DBGrid2->DataSource =
      dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
    DBNavigator2->DataSource =
      dynamic_cast<TDataSource *>(Form1->ListBox1->Items->Objects[Form1->ListBox1->ItemIndex]);
  };
}
```

Delphi Examples:

```
{
This example fills a list box with the names of TDataSources
that are present and then fills a DBGrid a second form with
the data from the selected data source.  The DBGrid is not
filled in until Form2 is activated.  Here is the code for Form1
that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
```

```
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;

procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
//    Looking for c:/ProgramFiles/Common Files/Codegear Shared/Data/dbdemos.gdb
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

  //    Take out this part to remove indexing
  //    Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
  //        The 1st index has no name because it is
  //        a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

  //      Call the CreateTable method to create the table
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 20 do
          Customers.AppendRecord([i*111, i*222, i*333, i*444]);
      end;
  end;

  procedure CreateBlobSource;
  var
    i: Integer;
  begin
    Blobs:= TTable.Create(Form1);
    with Blobs do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'MyBlobInfo';
      Blobs.Active := False;
      { Don't overwrite an existing table }
  //    if Blobs.Exists then
  //      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
  //    else
      begin
        if (Blobs.Exists) then
        begin
          Blobs.Close;
          Blobs.DeleteTable;
        end;
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
```

```
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftBlob;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
{
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
}
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Blobs.Active:= True;
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
end;

procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    Form1Done := True;
end;

initialization

Form1Done := False;
{
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it. Also, set the visible property
of Form2 to true.
```

```
}
procedure TForm2.FormActivate(Sender: TObject);
begin
  if (Form1Done) then
  begin
    DBGrid2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    DBNavigator2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
  end;
end;
```

See Also

FormStyle (see page 2162)

OnDeactivate (see page 2209)

OnActivate (see page 1992)

7.1.7.4.2 TForm.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

7.1.7.4.3 TForm.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

7.1.7.4.4 TForm.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

7.1.7.4.5 TForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

7.1.7.4.6 TForm.OnClose Event

Occurs when the form closes.

Pascal

```
property OnClose: TCloseEvent;
```

C++

```
__property TCloseEvent OnClose;
```

Description

Use OnClose to perform special processing when the form closes. The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close method or when the user chooses Close from the form's system menu.

The TCloseEvent type points to a method that handles the closing of a form. The value of the Action parameter determines if the form actually closes. These are the possible values of Action:

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

If a form is an MDI child form, and its BorderIcons property is biMinimize, then the default Action is caMinimize. If a MDI child form does not have these settings, the default Action is caNone, meaning that nothing happens when the user attempts to close the form.

If a form is an SDI child form, Action defaults to caHide.

To close the form and free it in an OnClose event, set Action to caFree.

Note: When the application shuts down, the main form receives an OnClose event, but any child forms do not receive the OnClose event.

C++ Examples:

```
/*
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
*/
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
  if (MessageDlg("Close application ?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
    Action = caFree;
  else
    Action = caMinimize;
}
```

Delphi Examples:

```
{
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
```

```
file to open Form2 and set the Form2 Visible property to
True.
}
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if MessageDlg('Close application ?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
    Action := caFree
  else
    Action := caMinimize;
end;
```

See Also

OnCreate (see page 2207)

OnCloseQuery (see page 2206)

OnDestroy (see page 2209)

7.1.7.4.7 TForm.OnCloseQuery Event

Occurs when close is attempted.

Pascal

```
property OnCloseQuery: TCloseQueryEvent;
```

C++

```
__property TCloseQueryEvent OnCloseQuery;
```

Description

Use OnCloseQuery to specify the conditions under which the form can close. An OnCloseQuery event handler returns a Boolean CanClose value that determines whether a form is allowed to close. Its default value is true.

You can use an OnCloseQuery event handler to ask users if they are sure they really want the form closed immediately. For example, you can use the handler to display a message box that prompts the user to save a file before closing the form.

The TCloseQueryEvent type points to the method that determines whether a form can be closed. The value of the CanClose parameter determines if the form can close or not.

C++ Examples:

```
/*
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
*/
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
  if (MessageDlg("Close the form?", mtConfirmation, TMsgDlgButtons() << mbOK << mbCancel,0) ==
mrCancel)
    CanClose = false;
}
```

Delphi Examples:

```
{
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
```

```
}
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Close the form?', mtConfirmation,
    [mbOk, mbCancel], 0) = mrCancel then
     CanClose := False;
end;
```

See Also

OnClose (see page 2205)

7.1.7.4.8 TForm.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

7.1.7.4.9 TForm.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

7.1.7.4.10 TForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm's constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy event.

When a form is being created and its Visible property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint

Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
```

```
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);  // These tabsheets are cleaned up by their
owner (PageControl1).
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

OnDestroy (see page 2209)

OnActivate (see page 2196)

OnPaint (see page 2233)

OnShow (see page 2236)

OldCreateOrder (see page 2178)

7.1.7.4.11 TForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

7.1.7.4.12 TForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication object's OnDeactivate event.

See Also

OnActivate (see page 2196)

OnDeactivate (see page 1992)

7.1.7.4.13 TForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate event.

C++ Examples:

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
*/
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 2207)

System::TObject::Free

Destroy (see page 2109)

OldCreateOrder (see page 2178)

7.1.7.4.14 TForm.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}
```

```
void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

7.1.7.4.15 TForm.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

7.1.7.4.16 TForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.7.4.17 TForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.7.4.18 TForm.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

7.1.7.4.19 TForm.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
```

```
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

7.1.7.4.20 TForm.OnHelp Event

Occurs when the form receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set the CallHelp parameter to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. The form calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The event handler returns true if it succeeds, false if it fails.

See Also

HelpCommand (see page 1941)

HelpContext (see page 1941)

HelpFile (see page 2165)

HelpJump (see page 1942)

HelpFile (see page 1974)

OnHelp (see page 1995)

7.1.7.4.21 TForm.OnHide Event

Occurs when the form is hidden (that is, when its Visible property is set to false).

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Use OnHide to perform special processing when the form is hidden (that is, when the form's Visible property is set to false).

C++ Examples:

```
/*
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
*/
#include "TFormOnHide.h"
#include "Unit2.h"  // We must include unit2's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
  Hide();
```

```
}

void __fastcall TForm1::FormHide(TObject *Sender)
{
  Form2->Label1->Caption = "Form1 is hiding";
}
/*
This is the implementation of Unit2:
*/
#include "Unit2.h"
#include "TFormOnHide.h"  // We must include unit1's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm2::Button1Click(TObject *Sender)
{
  Form1->Show();
  Hide();
}

void __fastcall TForm2::FormHide(TObject *Sender)
{
  Form1->Label1->Caption = "Form2 is hiding";
}
```

Delphi Examples:

```
{
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
}

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
  Hide;
end;

procedure TForm1.FormHide(Sender: TObject);
begin
  Form2.Label1.Caption := 'Form1 is hiding';
end;
{
This is the implementation section of Unit2:
}
implementation

{$R *.dfm}

uses TFormOnHide;
```

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Hide;
end;

procedure TForm2.FormHide(Sender: TObject);
begin
  Form1.Label1.Caption := 'Form2 is hiding';
end;
```

See Also

OnShow (see page 2236)

7.1.7.4.22 TForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
```

```
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
```

```
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}
```

```
uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

7.1.7.4.23 TForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

7.1.7.4.24 TForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  Behavior is dependent on keyboard driver(USB, serial, etc).
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown

OnKeyPress

7.1.7.4.25 TForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
```

```
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.7.4.26 TForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
```

```
  The following example requires a form with a four-paneled
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  */
  int StartX, StartY; // Declare at the top of the form's unit
  // Use this code as the OnMouseDown event handler of the form:
  void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    StartX = X;
    StartY = Y;
  }

  // Use this code as the OnMouseUp event handler of the form:
  void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    Form1->Canvas->Rectangle(StartX, StartY, X, Y);
    StatusBar1->Panels->Items[0]->Text = "";
    StatusBar1->Panels->Items[1]->Text = "";
    StatusBar1->Panels->Items[2]->Text = "";
    StatusBar1->Panels->Items[3]->Text = "";
  }

  // Use this code as the OnMouseMove event handler of the form:
  void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
  {
    if (Shift.Contains(ssLeft)) // make sure button is down
    {
      if (Y > StartY)
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
  {
  The following example requires a form with a four-paneled
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
```

```
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.7.4.27 TForm.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

7.1.7.4.28 TForm.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

7.1.7.4.29 TForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

7.1.7.4.30 TForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

7.1.7.4.31 TForm.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel;
```

C++

```
__property OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown

OnMouseWheelUp

7.1.7.4.32 TForm.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

7.1.7.4.33 TForm.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

7.1.7.4.34 TForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _TheGraphicCleaner(TheGraphic = new
Graphics::TBitmap());  // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 2150)

ClipRect (see page 2718)

7.1.7.4.35 TForm.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

7.1.7.4.36 TForm.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form handles them. When the user presses a key, the form can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut allows the form to implement additional shortcuts.

If the OnShortCut implements a response to the keystroke, set the Handled parameter of the event handler to true. This prevents the keystroke from being passed on to menus or actions associated with the form. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

See Also

OnKeyDown

IsShortCut (see page 2128)

IsShortCut

IsShortCut (see page 4845)

7.1.7.4.37 TForm.OnShow Event

Occurs when the form is shown (that is, when its Visible property is set to true).

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Use OnShow to perform special processing when the form is shown (that is, when the form's Visible property is set to true).

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
```

```
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  PageControl1Change(Sender);
end;
```

See Also

OnHide (see page 2217)

7.1.7.4.38 TForm.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

7.1.7.4.39 TForm.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
```

```
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
```

```
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

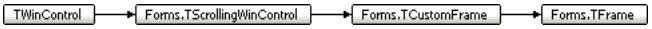
OnDockDrop

OnDockOver

7.1.8 TFrame Class

TFrame is a container for components; it can be nested within forms or other frames.

Class Hierarchy



Pascal

```
TFrame = class(TCustomFrame);
```

C++

```
class TFrame : public TCustomFrame;
```

File

Forms

Description

When you create frames, they are implemented as descendants of TFrame.

A frame, like a form, is a container for other components. It uses the same ownership mechanism as forms for automatic instantiation and destruction of the components on it, and the same parent-child relationships for synchronization of component properties. But frames can be nested within forms or other frames, and they can be saved on the Component palette for easy reuse. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. Moreover, an embedded frame continues to inherit changes made to the frame from which it is derived.

There is a nonstandard A-footnote (Frames_Object) in the main topic for TFrame. Please do not remove it, since it is necessary for making the Component palette Help work.

See Also

TForm (see page 2241)

7.1.8.1 TFrame Members

The following tables list the members exposed by TFrame.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2239)	Initializes frame creation parameters for efficient redrawing.
	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	PaintWindow (see page 2240)	Renders the image of a windowed control.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.
	SetParent (see page 2240)	Sets the Parent property.

Public Methods

	Name	Description
	Create (see page 2238)	Creates and initializes a TCustomFrame (see page 2237) object.
	Create (see page 2555)	Creates and initializes a new TScrollingWinControl (see page 2554) object.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	GetChildren (see page 2239)	Calls a specified method for each child of the frame.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2397)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2397)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 2398)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2398)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 2400)	Adjust resize constraints.
	OnContextPopup (see page 2401)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2401)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2402)	Occurs when another control is docked to the control.
	OnDockOver (see page 2404)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2406)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2409)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2412)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2412)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2413)	Occurs when a control receives the input focus.
	OnExit (see page 2416)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2419)	Returns the control's docking information.
	OnMouseActivate (see page 2421)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2422)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2424)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2424)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2424)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2427)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2429)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2429)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2430)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 2430)	Occurs immediately after the control is resized.
	OnStartDock (see page 2431)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2431)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2432)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 2370)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2373)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2374)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2374)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 2375)	Specifies the bi-directional mode for the control.
	Color (see page 2375)	Specifies the background color of the control.
	Constraints (see page 2376)	Specifies the size constraints for the control.
	Ctl3D (see page 2376)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2377)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2378)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	DragCursor (see page 2380)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2380)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2380)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2382)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2382)	Controls the attributes of text written on or in the control.
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	Padding (see page 2384)	Specifies the padding of a control.
	ParentBackground (see page 2384)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2384)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2385)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2385)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2385)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2386)	Determines where a control looks for its font information.
	ParentShowHint (see page 2386)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2387)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2392)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2393)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2394)	Determines if the user can tab to a control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 2394)	Determines whether the component appears onscreen.

7.1.8.2 TFrame Properties

The properties of the TFrame class are listed here.

Published Properties

	Name	Description
	Align (see page 2370)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2373)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2374)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2374)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 2375)	Specifies the bi-directional mode for the control.
	Color (see page 2375)	Specifies the background color of the control.
	Constraints (see page 2376)	Specifies the size constraints for the control.
	Ctl3D (see page 2376)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2377)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2378)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2380)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2380)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2380)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2382)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2382)	Controls the attributes of text written on or in the control.
	Padding (see page 2384)	Specifies the padding of a control.
	ParentBackground (see page 2384)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2384)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2385)	Determines where a control looks for its color information.

	ParentCtl3D (see page 2385)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2385)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2386)	Determines where a control looks for its font information.
	ParentShowHint (see page 2386)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2387)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2392)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2393)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2394)	Determines if the user can tab to a control.
	Visible (see page 2394)	Determines whether the component appears onscreen.

7.1.8.2.1 TFrame.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
```

```
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
```

```
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

7.1.8.2.2 TFrame.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at

design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.8.2.3 TFrame.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.8.2.4 TFrame.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

7.1.8.2.5 TFrame.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

7.1.8.2.6 TFrame.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
```

```
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

7.1.8.2.7 TFrame.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

7.1.8.2.8 TFrame.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

7.1.8.2.9 TFrame.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

7.1.8.2.10 TFrame.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

7.1.8.2.11 TFrame.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

7.1.8.2.12 TFrame.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

7.1.8.2.13 TFrame.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

7.1.8.2.14 TFrame.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

7.1.8.2.15 TFrame.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

7.1.8.2.16 TFrame.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

7.1.8.2.17 TFrame.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

7.1.8.2.18 TFrame.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

7.1.8.2.19 TFrame.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

7.1.8.2.20 TFrame.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

7.1.8.2.21 TFrame.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

7.1.8.2.22 TFrame.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

7.1.8.2.23 TFrame.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

7.1.8.2.24 TFrame.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
```

```
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
```

```
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

7.1.8.2.25 TFrame.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
```

```
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

7.1.8.2.26 TFrame.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is

sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

7.1.8.2.27 TFrame.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

7.1.8.2.28 TFrame.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
```

```
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

7.1.8.3 TFrame Events

The events of the TFrame class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2397)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2397)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 2398)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2398)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 2400)	Adjust resize constraints.
	OnContextPopup (see page 2401)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2401)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2402)	Occurs when another control is docked to the control.
	OnDockOver (see page 2404)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2406)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2409)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2412)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2412)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2413)	Occurs when a control receives the input focus.
	OnExit (see page 2416)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2419)	Returns the control's docking information.
	OnMouseActivate (see page 2421)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2422)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2424)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2424)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2424)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2427)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2429)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2429)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2430)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 2430)	Occurs immediately after the control is resized.
	OnStartDock (see page 2431)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2431)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2432)	Occurs when the application tries to undock a control that is docked to the windowed control.

7.1.8.3.1 TFrame.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

7.1.8.3.2 TFrame.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

7.1.8.3.3 TFrame.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

7.1.8.3.4 TFrame.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

7.1.8.3.5 TFrame.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

7.1.8.3.6 TFrame.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

7.1.8.3.7 TFrame.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

7.1.8.3.8 TFrame.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
```

```
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

7.1.8.3.9 TFrame.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

7.1.8.3.10 TFrame.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.8.3.11 TFrame.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.8.3.12 TFrame.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

7.1.8.3.13 TFrame.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

7.1.8.3.14 TFrame.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

7.1.8.3.15 TFrame.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));
```

```
    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

7.1.8.3.16 TFrame.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
```

```
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

7.1.8.3.17 TFrame.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.8.3.18 TFrame.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.8.3.19 TFrame.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

7.1.8.3.20 TFrame.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

7.1.8.3.21 TFrame.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

7.1.8.3.22 TFrame.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

7.1.8.3.23 TFrame.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel;
```

C++

```
__property OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown

OnMouseWheelUp

7.1.8.3.24 TFrame.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

7.1.8.3.25 TFrame.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

7.1.8.3.26 TFrame.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

7.1.8.3.27 TFrame.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

7.1.8.3.28 TFrame.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

7.1.8.3.29 TFrame.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
```

```
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

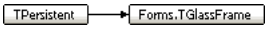
OnDockDrop

OnDockOver

7.1.9 TGlassFrame Class

TGlassFrame controls Windows Vista Aero glass effects.

Class Hierarchy



Pascal

```
TGlassFrame = class(TPersistent);
```

C++

```
class TGlassFrame : public TPersistent;
```

File

Forms

Description

TGlassFrame uses the DwmExtendFrameIntoClientArea API to extend glass effects into the client area. Use the Top (see page 2439), Left (see page 2438), Bottom (see page 2438), and Right (see page 2439) properties to specify the amount to extend the glass effect into the client area, or use SheetOfGlass (see page 2439) to render the entire client area as a glass surface. This is only available on Windows Vista with Aero.

See Also

TForm (see page 2241)

TCustomForm (see page 2103)

7.1.9.1 TGlassFrame Members

The following tables list the members exposed by TGlassFrame.

Protected Methods

	Name	Description
	Change (see page 2437)	This is Change, a member of class TGlassFrame.

Public Events

	Name	Description
	OnChange (see page 2439)	Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

Public Methods

	Name	Description
	Assign (see page 2436)	Copies the contents of another, similar object.
	Create (see page 2436)	This is Create, a member of class TGlassFrame.
	FrameExtended (see page 2437)	Returns True if glass effects are extended into the client area.
	IntersectsControl (see page 2438)	Returns True if a control intersects the glass area.

Published Properties

	Name	Description
	Bottom (see page 2438)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Enabled (see page 2438)	This property controls whether glass effects are extended into the client area or not.
	Left (see page 2438)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Right (see page 2439)	Specifies the amount (in pixels) glass effects are extended into the client area.
	SheetOfGlass (see page 2439)	Extends glass effects into the entire client area.
	Top (see page 2439)	Specifies the amount (in pixels) glass effects are extended into the client area.

7.1.9.2 TGlassFrame.Create Constructor

Pascal

```
constructor Create(Client: TCustomForm);
```

C++

```
__fastcall TGlassFrame(TCustomForm Client);
```

Description

This is Create, a member of class TGlassFrame.

7.1.9.3 TGlassFrame Methods

The methods of the TGlassFrame class are listed here.

Protected Methods

	Name	Description
	Change (see page 2437)	This is Change, a member of class TGlassFrame.

Public Methods

	Name	Description
	Assign (see page 2436)	Copies the contents of another, similar object.
	FrameExtended (see page 2437)	Returns True if glass effects are extended into the client area.
	IntersectsControl (see page 2438)	Returns True if a control intersects the glass area.

7.1.9.3.1 TGlassFrame.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

7.1.9.3.2 TGlassFrame.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TGlassFrame.

7.1.9.3.3 TGlassFrame.FrameExtended Method

Returns True if glass effects are extended into the client area.

Pascal

```
function FrameExtended: Boolean;
```

C++

```
__fastcall Boolean FrameExtended();
```

Description

Use FrameExtended to determine if glass effects are supported and enabled. The return value is True if Desktop Window Manager (DWM) composition is enabled, Enabled (see page 2438) is True and either SheetOfGlass (see page 2439) is True or any of the properties Top (see page 2439), Left (see page 2438), Bottom (see page 2438) or Right (see page 2439) have a value other than zero.

7.1.9.3.4 TGlassFrame.IntersectsControl Method

Returns True if a control intersects the glass area.

Pascal

```
function IntersectsControl(Control: TControl): Boolean;
```

C++

```
__fastcall Boolean IntersectsControl(TControl * Control);
```

7.1.9.4 TGlassFrame Properties

The properties of the TGlassFrame class are listed here.

Published Properties

	Name	Description
	Bottom (see page 2438)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Enabled (see page 2438)	This property controls whether glass effects are extended into the client area or not.
	Left (see page 2438)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Right (see page 2439)	Specifies the amount (in pixels) glass effects are extended into the client area.
	SheetOfGlass (see page 2439)	Extends glass effects into the entire client area.
	Top (see page 2439)	Specifies the amount (in pixels) glass effects are extended into the client area.

7.1.9.4.1 TGlassFrame.Bottom Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Bottom: Integer;
```

C++

```
__property int Bottom;
```

7.1.9.4.2 TGlassFrame.Enabled Property

This property controls whether glass effects are extended into the client area or not.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

7.1.9.4.3 TGlassFrame.Left Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

7.1.9.4.4 TGlassFrame.Right Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Right: Integer;
```

C++

```
__property int Right;
```

7.1.9.4.5 TGlassFrame.SheetOfGlass Property

Extends glass effects into the entire client area.

Pascal

```
property SheetOfGlass: Boolean;
```

C++

```
__property Boolean SheetOfGlass;
```

7.1.9.4.6 TGlassFrame.Top Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

7.1.9.5 TGlassFrame Events

The events of the TGlassFrame class are listed here.

Public Events

	Name	Description
	OnChange (see page 2439)	Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

7.1.9.5.1 TGlassFrame.OnChange Event

Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

Pascal

```
property OnChange: TNotifyEvent;
```

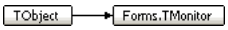
C++

```
__property TNotifyEvent OnChange;
```

7.1.10 TMonitor Class

TMonitor represents a monitor on which an application runs.

Class Hierarchy



Pascal

```
TMonitor = class(TObject);
```

C++

```
class TMonitor : public TObject;
```

File

Forms

Description

TMonitor represents a monitor that displays the user interface of an application. It introduces properties to describe the dimensions of a single monitor. In addition, the Primary (see page 2443) property distinguishes the primary property from other monitors on the system.

When writing multi-monitor applications, use TMonitor to help lay out forms and dialogs so that they are not split over several monitors.

Applications do not create instances of TMonitor. Instead, monitor components for all available monitors are listed by the Monitors property of the global Screen (see page 2615) variable. The global Screen (see page 2615) variable also provides global information about the monitors on which the application runs, such as screen resolution, available fonts, and so on.

See Also

TApplication (see page 1924)

TScreen (see page 2445)

TCustomForm (see page 2103)

Screen (see page 2615)

TCustomForm (see page 2103)

7.1.10.1 TMonitor Members

The following tables list the members exposed by TMonitor.

Public Properties

	Name	Description
R	BoundsRect (see page 2441)	Indicates the dimensions of the monitor in pixels.
R	Handle (see page 2441)	Indicates the Windows handle for the monitor.
R	Height (see page 2442)	Indicates the vertical size of the monitor in pixels.
R	Left (see page 2442)	Indicates the logical position of the left edge of the monitor.
R	MonitorNum (see page 2443)	Specifies the index of the monitor in the global screen object's Monitors list.
R	Primary (see page 2443)	Indicates whether this is the primary monitor.
R	Top (see page 2443)	Indicates the logical position of the top edge of the monitor.
R	Width (see page 2444)	Indicates the horizontal size of the monitor in pixels.
R	WorkareaRect (see page 2444)	Gives the application useable area of the monitor.

7.1.10.2 TMonitor Properties

The properties of the TMonitor class are listed here.

Public Properties

	Name	Description
	BoundsRect (see page 2441)	Indicates the dimensions of the monitor in pixels.
	Handle (see page 2441)	Indicates the Windows handle for the monitor.
	Height (see page 2442)	Indicates the vertical size of the monitor in pixels.
	Left (see page 2442)	Indicates the logical position of the left edge of the monitor.
	MonitorNum (see page 2443)	Specifies the index of the monitor in the global screen object's Monitors list.
	Primary (see page 2443)	Indicates whether this is the primary monitor.
	Top (see page 2443)	Indicates the logical position of the top edge of the monitor.
	Width (see page 2444)	Indicates the horizontal size of the monitor in pixels.
	WorkareaRect (see page 2444)	Gives the application useable area of the monitor.

7.1.10.2.1 TMonitor.BoundsRect Property

Indicates the dimensions of the monitor in pixels.

Pascal

```
property BoundsRect: TRect;
```

C++

```
__property TRect BoundsRect;
```

Description

Read BoundsRect to learn the dimensions of the monitor. BoundsRect gives the dimensions of the monitor in pixels, where (0,0) represents the top-left corner of the primary monitor.

The top of BoundsRect is given by the Top (see page 2443) property, the left edge by the Left (see page 2442) property, and the height and width by the Height (see page 2442) and Width (see page 2444) properties respectively.

Note: The BoundsRect property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Width (see page 2444)

Top (see page 2443)

Left (see page 2442)

Height (see page 2442)

WorkAreaRect (see page 2444)

DesktopRect (see page 2464)

Primary (see page 2443)

7.1.10.2.2 TMonitor.Handle Property

Indicates the Windows handle for the monitor.

Pascal

```
property Handle: HMONITOR;
```

C++

```
__property HMONITOR Handle;
```

Description

Read Monitor to obtain a Windows handle for the monitor represented by this TMonitor (see page 2439) component. Use this handle for Windows API calls such as GetMonitorInfo.

See Also

MonitorNum (see page 2443)

7.1.10.2.3 TMonitor.Height Property

Indicates the vertical size of the monitor in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Read Height to learn the size of the monitor in pixels. Use this value along with the Top (see page 2443) property to calculate how to position objects vertically in multi-monitor applications.

Note: The Height property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Height (see page 2470)

DesktopHeight (see page 2463)

Width (see page 2444)

Top (see page 2443)

BoundsRect (see page 2441)

WorkAreaRect (see page 2444)

7.1.10.2.4 TMonitor.Left Property

Indicates the logical position of the left edge of the monitor.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

In multi-monitor applications, the available screen space can span multiple monitors from left to right. Read Left to determine where, in the entire available screen space, this monitor appears. The value of Left is in pixels relative to the left edge of the primary monitor.

See Also

Width (see page 2444)

Top (see page 2443)

BoundsRect (see page 2441)

WorkAreaRect (see page 2444)

Primary (see page 2443)

DesktopLeft (see page 2463)

7.1.10.2.5 TMonitor.MonitorNum Property

Specifies the index of the monitor in the global screen object's Monitors list.

Pascal

```
property MonitorNum: Integer;
```

C++

```
__property int MonitorNum;
```

Description

The global screen variable maintains a list of all monitors in a multi-monitor application. MonitorNum is the index of this monitor in the list object that is the screen's Monitors property.

Use MonitorNum to identify a specific monitor when working with multiple monitors.

See Also

Monitors (see page 2474)

Screen (see page 2615)

7.1.10.2.6 TMonitor.Primary Property

Indicates whether this is the primary monitor.

Pascal

```
property Primary: Boolean;
```

C++

```
__property Boolean Primary;
```

Description

Read Primary to determine whether a monitor is the primary monitor in a multi-monitor application. When Primary is true, the monitor is the primary monitor. The coordinates of other monitors are given in terms of the primary monitor.

See Also

Monitors (see page 2474)

BoundsRect (see page 2441)

DesktopRect (see page 2464)

7.1.10.2.7 TMonitor.Top Property

Indicates the logical position of the top edge of the monitor.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

In multi-monitor applications, the available screen space can span multiple monitors from top to bottom. Read Top to determine where, in the entire available screen space, this monitor appears. The value of Top is in pixels relative to the top edge of the primary monitor.

See Also

Height (see page 2442)

Left (see page 2442)

BoundsRect (see page 2441)

WorkAreaRect (see page 2444)

Primary (see page 2443)

DesktopTop (see page 2465)

7.1.10.2.8 TMonitor.Width Property

Indicates the horizontal size of the monitor in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Read Width to learn the size of the monitor in pixels. Use this value along with the Left (see page 2442) property to calculate vertical positions for objects in multi-monitor applications.

Note: The Width property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Height (see page 2442)

Left (see page 2442)

Width (see page 2475)

DesktopWidth (see page 2465)

BoundsRect (see page 2441)

WorkAreaRect (see page 2444)

7.1.10.2.9 TMonitor.WorkareaRect Property

Gives the application useable area of the monitor.

Pascal

```
property WorkareaRect: TRect;
```

C++

```
__property TRect WorkareaRect;
```

Description

WorkareaRect returns a TRect value furnished with the coordinates and dimensions of the work area of the Monitor. On Windows, for example, the application tabs at the screen mean that the Workarea is smaller than the monitor size.

Note: The TRect Right and Bottom values are one pixel beyond Workarea boundary. They are given these values to allow for easy calculation of Workarea width and height as (Right-Left (see page 2442)) and (Bottom-Top (see page 2443)) respectively.

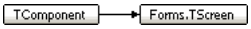
See Also

TRect

7.1.11 TScreen Class

TScreen represents the state of the screen in which an application runs.

Class Hierarchy



Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TScreen = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TScreen : public TComponent;
```

File

Forms

Description

TScreen introduces properties that keep track of

What forms and data modules have been instantiated by the application.

The active form, and the active control within that form.

The size and resolution of the screen.

The cursors and fonts available for the application to use.

The monitors on which the application's forms can appear, including their dimensions.

There is a global variable, Screen (see page 2615), of type TScreen, which is already instantiated for use by GUI applications. Use Screen (see page 2615) to obtain information about the current state of the screen in an application.

See Also

TApplication (see page 1924)

TDataModule

TForm (see page 2241)

TFont (see page 2730)

Screen (see page 2615)

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Handling the Screen

Controlling Application Behavior

7.1.11.1 TScreen Members

The following tables list the members exposed by TScreen.

Public Events

	Name	Description
	OnActiveControlChange (see page 2479)	Occurs immediately after input focus changes to a new windowed control.
	OnActiveFormChange (see page 2481)	Occurs immediately after a new form becomes active in a multi-form application.

Public Methods

	Name	Description
V	Create (see page 2447)	Creates an instance of TScreen (see page 2445).
V	Destroy (see page 2448)	Destroys an instance of TScreen (see page 2445).
	DisableAlign (see page 2448)	Prevents forms from being aligned in the screen.
	EnableAlign (see page 2449)	Allows forms to be aligned in the screen.
	MonitorFromPoint (see page 2449)	Returns the monitor where a specified point is located.
	MonitorFromRect (see page 2449)	Returns the monitor where a specified rectangle is located.
	MonitorFromWindow (see page 2450)	Returns the monitor where a specified window is located.
	Realign (see page 2450)	Realigns the screen's forms according to their Align properties.
	ResetFonts (see page 2451)	Reinitializes the fonts listed in the Fonts (see page 2466) property.

Public Properties

	Name	Description
R	ActiveControl (see page 2452)	Indicates which control currently has input focus on the screen.
R	ActiveCustomForm (see page 2452)	Indicates the descendant of TCustomForm (see page 2103) that currently has focus.
R	ActiveForm (see page 2453)	Indicates which form currently has focus.
	Cursor (see page 2455)	Controls the mouse cursor image on a global level.
R	CursorCount (see page 2458)	This is CursorCount, a member of class TScreen.
	Cursors (see page 2458)	Lists the cursors available to the application.
R	CustomFormCount (see page 2460)	Indicates the number of forms or property pages displayed on the screen.
R	CustomForms (see page 2460)	Lists all the forms and property pages that are currently displayed in the application.
R	DataModuleCount (see page 2461)	Indicates the number of data modules that are currently instantiated in the application.
R	DataModules (see page 2462)	Lists all the data modules that are currently instantiated in the application.
R	DefaultIme (see page 2462)	Indicates which input method editor (IME) is active when the TScreen (see page 2445) object is created.
R	DefaultKbLayout (see page 2463)	Indicates the Windows handle to the keyboard layout that was active when the application started running.
R	DesktopHeight (see page 2463)	Specifies the height of the entire virtual desktop.

R	DesktopLeft (see page 2463)	Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.
R	DesktopRect (see page 2464)	Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.
R	DesktopTop (see page 2465)	Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.
R	DesktopWidth (see page 2465)	Specifies the width of the entire virtual desktop.
	FocusedForm (see page 2466)	Provides a reference to the form that has focus.
R	Fonts (see page 2466)	Lists the face names for all fonts supported by the screen.
R	FormCount (see page 2467)	Indicates the number of forms displayed on the screen.
R	Forms (see page 2468)	Lists all the forms currently displayed in the application.
R	Height (see page 2470)	Indicates the vertical size of the screen in pixels.
	HintFont (see page 2471)	Specifies the font used to display help hints.
	IconFont (see page 2472)	Specifies the font used to display the labels on the icons in file selection dialogs.
R	Imes (see page 2472)	Lists all input method editors (IMEs) installed on the system.
	MenuFont (see page 2473)	Specifies the font used for menu commands.
	MessageFont (see page 2473)	This is MessageFont, a member of class TScreen.
R	MonitorCount (see page 2473)	Specifies the number of monitors used to comprise the desktop.
R	Monitors (see page 2474)	Provides access to an individual monitor used to comprise the desktop.
R	PixelsPerInch (see page 2474)	Indicates the number of screen pixels that make up a logical inch in the vertical direction.
R	PrimaryMonitor (see page 2474)	Provides a reference to the primary monitor.
R	SaveFocusedList (see page 2475)	This is SaveFocusedList, a member of class TScreen.
R	Width (see page 2475)	Indicates the horizontal size of the screen in pixels.
R	WorkAreaHeight (see page 2476)	Specifies the height of the work area on the primary monitor.
R	WorkAreaLeft (see page 2477)	Specifies the left edge of the work area on the primary monitor.
R	WorkAreaRect (see page 2477)	Specifies the boundaries of the usable work area on the primary monitor.
R	WorkAreaTop (see page 2478)	Specifies the top edge of the work area on the primary monitor.
R	WorkAreaWidth (see page 2478)	Specifies the width of the work area on the primary monitor.

7.1.11.2 TScreen.Create Constructor

Creates an instance of TScreen (see page 2445).

Pascal

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

C++

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

Description

Do not create separate instances of TScreen (see page 2445). There is a global variable, Screen (see page 2615), of type TScreen (see page 2445), which is already instantiated for use by GUI applications. The properties of Screen (see page 2615) are global to the entire application.

AOwner specifies the Owner of the screen, which is responsible for freeing it. For the global TScreen (see page 2445) object, this is the application.

See Also

Screen (see page 2615)

Owner

TApplication (see page 1924)

7.1.11.3 TScreen.Destroy Destructor

Destroys an instance of TScreen (see page 2445).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreen();
```

Description

Do not destroy the global object Screen (see page 2615) that is automatically instantiated in GUI applications.

See Also

Screen (see page 2615)

System::TObject::Free

7.1.11.4 TScreen Methods

The methods of the TScreen class are listed here.

Public Methods

	Name	Description
	DisableAlign (see page 2448)	Prevents forms from being aligned in the screen.
	EnableAlign (see page 2449)	Allows forms to be aligned in the screen.
	MonitorFromPoint (see page 2449)	Returns the monitor where a specified point is located.
	MonitorFromRect (see page 2449)	Returns the monitor where a specified rectangle is located.
	MonitorFromWindow (see page 2450)	Returns the monitor where a specified window is located.
	Realign (see page 2450)	Realigns the screen's forms according to their Align properties.
	ResetFonts (see page 2451)	Reinitializes the fonts listed in the Fonts (see page 2466) property.

7.1.11.4.1 TScreen.DisableAlign Method

Prevents forms from being aligned in the screen.

Pascal

```
procedure DisableAlign;
```

C++

```
__fastcall DisableAlign();
```

Description

Call DisableAlign to prevent forms from being aligned in the screen. After calling DisableAlign, the Align properties of the screen's forms are ignored until the call is matched by a call to EnableAlign (see page 2449).

See Also

EnableAlign (see page 2449)

Realign (see page 2450)

Constraints

7.1.11.4.2 TScreen.EnableAlign Method

Allows forms to be aligned in the screen.

Pascal

```
procedure EnableAlign;
```

C++

```
__fastcall EnableAlign();
```

Description

Call EnableAlign to allow forms to be aligned in the screen. After calling DisableAlign (see page 2448), the Align properties of the screen's forms are ignored until that call is matched by a call to EnableAlign.

See Also

DisableAlign (see page 2448)

Realign (see page 2450)

Constraints

7.1.11.4.3 TScreen.MonitorFromPoint Method

Returns the monitor where a specified point is located.

Pascal

```
function MonitorFromPoint(const Point: TPoint; MonitorDefault: TMonitorDefaultTo = mdNearest):
TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromPoint(const TPoint Point, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromPoint determine the monitor where a specified point is located.

Point is the coordinates of the point to locate. It is in pixels relative to the upper-left corner of the primary monitor.

MonitorDefault indicates how to select a monitor when the point does not fall clearly on a single monitor.

See Also

Monitors (see page 2474)

DesktopRect (see page 2464)

BoundsRect (see page 2441)

7.1.11.4.4 TScreen.MonitorFromRect Method

Returns the monitor where a specified rectangle is located.

Pascal

```
function MonitorFromRect(const Rect: TRect; MonitorDefault: TMonitorDefaultTo = mdNearest):
TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromRect(const TRect Rect, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromRect determine the monitor where a specified boundary rectangle falls.

Rect is the coordinates of the rectangle to locate. It is in pixels relative to the upper-left corner of the primary monitor.

MonitorDefault indicates how to select a monitor when the rectangle does not fall entirely on a single monitor.

See Also

Monitors (see page 2474)

DesktopRect (see page 2464)

BoundsRect (see page 2441)

7.1.11.4.5 TScreen.MonitorFromWindow Method

Returns the monitor where a specified window is located.

Pascal

```
function MonitorFromWindow(const Handle: THandle; MonitorDefault: TMonitorDefaultTo =
mdNearest): TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromWindow(const THandle Handle, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromWindow determine the monitor that displays a specified window.

Handle is the window handle for the window to be located.

MonitorDefault indicates how to select a monitor when the window does not fall entirely on a single monitor.

See Also

Monitors (see page 2474)

DesktopRect (see page 2464)

BoundsRect (see page 2441)

7.1.11.4.6 TScreen.Realign Method

Realigns the screen's forms according to their Align properties.

Pascal

```
procedure Realign;
```

C++

```
__fastcall Realign();
```

Description

Call Realign to realign the screen's forms according to their Align properties.

See Also

EnableAlign (see page 2449)

DisableAlign (see page 2448)

Constraints

7.1.11.4.7 TScreen.ResetFonts Method

Reinitializes the fonts listed in the Fonts (see page 2466) property.

Pascal

```
procedure ResetFonts;
```

C++

```
__fastcall ResetFonts();
```

Description

Call ResetFonts to ensure that the fonts listed in the Fonts (see page 2466) property are up to date. ResetFonts reinitializes the Fonts (see page 2466) property so that its value is reread from the operating system the next time it is needed.

TScreen (see page 2445) calls ResetFonts internally in response to messages from Windows that the available fonts have changed.

See Also

Fonts (see page 2466)

7.1.11.5 TScreen Properties

The properties of the TScreen class are listed here.

Public Properties

	Name	Description
R	ActiveControl (see page 2452)	Indicates which control currently has input focus on the screen.
R	ActiveCustomForm (see page 2452)	Indicates the descendant of TCustomForm (see page 2103) that currently has focus.
R	ActiveForm (see page 2453)	Indicates which form currently has focus.
	Cursor (see page 2455)	Controls the mouse cursor image on a global level.
R	CursorCount (see page 2458)	This is CursorCount, a member of class TScreen.
	Cursors (see page 2458)	Lists the cursors available to the application.
R	CustomFormCount (see page 2460)	Indicates the number of forms or property pages displayed on the screen.
R	CustomForms (see page 2460)	Lists all the forms and property pages that are currently displayed in the application.
R	DataModuleCount (see page 2461)	Indicates the number of data modules that are currently instantiated in the application.
R	DataModules (see page 2462)	Lists all the data modules that are currently instantiated in the application.
R	DefaultIme (see page 2462)	Indicates which input method editor (IME) is active when the TScreen (see page 2445) object is created.
R	DefaultKbLayout (see page 2463)	Indicates the Windows handle to the keyboard layout that was active when the application started running.
R	DesktopHeight (see page 2463)	Specifies the height of the entire virtual desktop.
R	DesktopLeft (see page 2463)	Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.
R	DesktopRect (see page 2464)	Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.

R	DesktopTop (see page 2465)	Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.
R	DesktopWidth (see page 2465)	Specifies the width of the entire virtual desktop.
	FocusedForm (see page 2466)	Provides a reference to the form that has focus.
R	Fonts (see page 2466)	Lists the face names for all fonts supported by the screen.
R	FormCount (see page 2467)	Indicates the number of forms displayed on the screen.
R	Forms (see page 2468)	Lists all the forms currently displayed in the application.
R	Height (see page 2470)	Indicates the vertical size of the screen in pixels.
	HintFont (see page 2471)	Specifies the font used to display help hints.
	IconFont (see page 2472)	Specifies the font used to display the labels on the icons in file selection dialogs.
R	Imes (see page 2472)	Lists all input method editors (IMEs) installed on the system.
	MenuFont (see page 2473)	Specifies the font used for menu commands.
	MessageFont (see page 2473)	This is MessageFont, a member of class TScreen.
R	MonitorCount (see page 2473)	Specifies the number of monitors used to comprise the desktop.
R	Monitors (see page 2474)	Provides access to an individual monitor used to comprise the desktop.
R	PixelsPerInch (see page 2474)	Indicates the number of screen pixels that make up a logical inch in the vertical direction.
R	PrimaryMonitor (see page 2474)	Provides a reference to the primary monitor.
R	SaveFocusedList (see page 2475)	This is SaveFocusedList, a member of class TScreen.
R	Width (see page 2475)	Indicates the horizontal size of the screen in pixels.
R	WorkAreaHeight (see page 2476)	Specifies the height of the work area on the primary monitor.
R	WorkAreaLeft (see page 2477)	Specifies the left edge of the work area on the primary monitor.
R	WorkAreaRect (see page 2477)	Specifies the boundaries of the usable work area on the primary monitor.
R	WorkAreaTop (see page 2478)	Specifies the top edge of the work area on the primary monitor.
R	WorkAreaWidth (see page 2478)	Specifies the width of the work area on the primary monitor.

7.1.11.5.1 TScreen.ActiveControl Property

Indicates which control currently has input focus on the screen.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Read ActiveControl to learn which windowed control object in the active form currently receives the input from the keyboard.

ActiveControl is a read-only property. To change the ActiveControl, use the SetFocusedControl method of the form that contains the control that should receive focus.

After focus shifts from one control to another, the screen receives an OnActiveControlChange (see page 2479) event.

See Also

OnActiveControlChange (see page 2479)

ActiveControl (see page 2145)

SetFocusedControl (see page 2136)

7.1.11.5.2 TScreen.ActiveCustomForm Property

Indicates the descendant of TCustomForm (see page 2103) that currently has focus.

Pascal

```
property ActiveCustomForm: TCustomForm;
```

C++

```
__property TCustomForm ActiveCustomForm;
```

Description

Read ActiveCustomForm to learn which form or property page has input focus. If the active custom form is a TForm (see page 2241) object, ActiveCustomForm is the same as ActiveForm (see page 2453).

See Also

ActiveForm (see page 2453)

OnActiveFormChange (see page 2481)

TForm (see page 2241)

TPropertyPage

7.1.11.5.3 TScreen.ActiveForm Property

Indicates which form currently has focus.

Pascal

```
property ActiveForm: TForm;
```

C++

```
__property TForm * ActiveForm;
```

Description

Read ActiveForm to learn which form in the application has input focus. If the application is not currently active, ActiveForm is the form that will have focus when the application becomes active again.

ActiveForm is a read-only property. To change the ActiveForm, use the SetFocus method of the form that should receive focus. ActiveForm also changes if the SetFocusedControl method of an inactive form is called to set focus to a control on that inactive form.

After focus shifts from one form to another, the screen receives an OnActiveFormChange (see page 2481) event.

Note: If the application window with focus is not a TForm (see page 2241) descendant (for example, if it is a property page), it will not appear as ActiveForm. To handle such cases, use the ActiveCustomForm (see page 2452) property instead.

C++ Examples:

```
/*
This example uses two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the definition of the TForm1 class.
*/
#include "Unit2.h"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
```

```
}

void __fastcall TForm1::ColorForm(TObject *Sender)
{
  Color = clWhite;
  Form2->Color = clWhite;
  Screen->ActiveForm->Color = clAqua;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveFormChange = ColorForm;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // This stops ColorForm from being called during termination.
  Screen->OnActiveFormChange = NULL;
}
```

Delphi Examples:

```
{
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the header for form 1's unit.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ColorForm(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.ColorForm(Sender: TObject);
begin
  Color := clWhite;
  Form2.Color := clWhite;
  Screen.ActiveForm.Color := clAqua;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Screen.OnActiveFormChange := ColorForm;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  // This stops ColorForm from being called during termination.
  Screen.OnActiveFormChange := nil;
end;
```

See Also

ActiveCustomForm (see page 2452)

OnActiveFormChange (see page 2481)

ActiveMDIChild (see page 2146)

SetFocus (see page 2135)

SetFocusedControl (see page 2136)

How Actions Find Their Targets

7.1.11.5.4 TScreen.Cursor Property

Controls the mouse cursor image on a global level.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Set Cursor to override the cursor image associated with individual control objects. When Cursor is crDefault, the individual objects determine the cursor image. Assigning any other value sets the mouse cursor image for all windows belonging to the application. The global mouse cursor image remains in effect until the screen's Cursor property is changed back to crDefault.

Cursor can be set to any of the cursor values available in the Cursors (see page 2458) property. This can be one of the built-in cursor values, or a custom cursor that has been added to the Cursors (see page 2458) property.

Note: Changes to the value of the Cursor property may not show up immediately. Call the ProcessMessages method of the global Application (see page 2612) object to allow the application to respond to a notification that the value has changed.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/

#include <memory>       //for STL auto_ptr class

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bmpMask(new Graphics::TBitmap);
  std::auto_ptr<Graphics::TBitmap> bmpColor(new Graphics::TBitmap);
  std::auto_ptr<TIconInfo> iconInfo(new TIconInfo);
```

```
  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo.get());

  Screen->Cursor = crMyCursor;
}
/*
Assignments to the Screen object's cursor property are
typically guarded by a try...finally statement to ensure
that normal cursor behavior is restored. Note that the
original cursor is saved and restored.  This allows cursor
changes to be nested without accidentally restoring the
cursor to a hard-coded value prematurely.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TCursor Save_Cursor = Screen->Cursor;
Screen->Cursor = crHourGlass;    // Show hourglass cursor
try
{
  // Do some lengthy operation
  int I = 0;
  int J = 0;
  int X, Y;
  const magicnumber = 150000000;
  while (I < magicnumber)
  {
    Randomize;
    while (J < magicnumber)
    {
      Y = Random(J);
      J++;
    }
    X = Random(I);
    I++;
  }
}
__finally
{
  Screen->Cursor = Save_Cursor; // always restore the cursor
}
}
```

Delphi Examples:

```
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
    bmpMask := TBitmap.Create;
    bmpColor := TBitmap.Create;

    bmpMask.LoadFromFile('SquareMask.bmp');
    bmpColor.LoadFromFile('Square.bmp');

    with iconInfo do
    begin
      fIcon := false;
      xHotspot := 15;
      yHotspot := 15;
      hbmMask := bmpMask.Handle;
      hbmColor := bmpColor.Handle;
    end;

    Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

    Screen.Cursor := crMyCursor;

    bmpMask.Free;
    bmpColor.Free;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    DestroyIcon(Screen.Cursors[crMyCursor]);
  end;
  {
  Assignments to the Screen object's cursor property are
  typically guarded by a try...finally statement to ensure
  that normal cursor behavior is restored. Note that the
  original cursor is saved and restored.  This allows cursor
  changes to be nested without accidentally restoring the
  cursor to a hard-coded value prematurely.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Save_Cursor : TCursor;
    I, J: Integer;
    X, Y: Word;
  const
    magicnumber = 3000000;
  begin
    Save_Cursor := Screen.Cursor;
    Screen.Cursor := crHourGlass;    { Show hourglass cursor }
    try
      { Do some lengthy operation }
    I := 0;
    J := 0;
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    finally
      Screen.Cursor := Save_Cursor;  { Always restore to normal }
    end;
  end;
```

See Also

Cursors (see page 2458)

Cursor

ProcessMessages (see page 1954)

7.1.11.5.5 TScreen.CursorCount Property

Pascal

```
property CursorCount: Integer;
```

C++

```
__property int CursorCount;
```

Description

This is CursorCount, a member of class TScreen.

7.1.11.5.6 TScreen.Cursors Property

Lists the cursors available to the application.

Pascal

```
property Cursors [Index: Integer]: HCURSOR;
```

C++

```
__property HCURSOR Cursors[int Index];
```

Description

Use Cursor (see page 2455) to access a particular cursor for use by the application or by a control within the application. TScreen (see page 2445) includes several built-in cursors that are indexed by symbolic cursor constants. The image associated with the built-in cursors constants can be changed by setting the Cursors property.

Custom cursors can be added to the Cursors property for use by the application or any of its controls. To add a custom cursor to an application:

1Create (see page 2447) the cursor resource using a resource editor.

2Declare a cursor constant with a value that does not conflict with an existing cursor constant.

3Use the Windows API function LoadCursor to obtain a handle to the new cursor.

4Set the Cursors property, indexed by the newly declared cursor constant, to the handle obtained from LoadCursor.

Note: Don't call the Windows API function DestroyCursor when finished with a custom cursor; it is handled automatically.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/

#include <memory>       //for STL auto_ptr class
```

```
const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bmpMask(new Graphics::TBitmap);
  std::auto_ptr<Graphics::TBitmap> bmpColor(new Graphics::TBitmap);
  std::auto_ptr<TIconInfo> iconInfo(new TIconInfo);

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo.get());

  Screen->Cursor = crMyCursor;
}
```

Delphi Examples:

```
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

  Screen.Cursor := crMyCursor;

  bmpMask.Free;
  bmpColor.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  DestroyIcon(Screen.Cursors[crMyCursor]);
end;
```

See Also

Cursor (see page 2455)

Cursor

7.1.11.5.7 TScreen.CustomFormCount Property

Indicates the number of forms or property pages displayed on the screen.

Pascal

```
property CustomFormCount: Integer;
```

C++

```
__property int CustomFormCount;
```

Description

Read CustomFormCount to learn the number of forms or property pages currently displayed on the screen. These forms can be accessed by the CustomForms (see page 2460) property. CustomFormCount can be used with CustomForms (see page 2460) to iterate over all the forms and property pages in an application.

C++ Examples:

```
/*
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->CustomFormCount; i++)
    ListBox1->Items->Add(Screen->CustomForms[i]->Name);
}
```

Delphi Examples:

```
{
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Screen.CustomFormCount-1 do
    ListBox1.Items.Add(Screen.CustomForms[I].Name);
end;
```

See Also

CustomForms (see page 2460)

FormCount (see page 2467)

7.1.11.5.8 TScreen.CustomForms Property

Lists all the forms and property pages that are currently displayed in the application.

Pascal

```
property CustomForms [Index: Integer]: TCustomForm;
```

C++

```
__property TCustomForm CustomForms[int Index];
```

Description

Use CustomForms to access a form or property page by index. The value of Index is a number between zero (the first custom form) and CustomFormCount (see page 2460) - 1. CustomForms can be used with CustomFormCount (see page 2460) when an application needs to iterate over all its forms and property pages.

Unlike the Forms (see page 2468) property, CustomForms lists TCustomForm (see page 2103) descendants that do not derive from TForm (see page 2241) (such as property pages).

Warning: The order in which CustomForms lists its forms is affected by the Z order of the forms. Do not change the Z order of forms when using CustomForms to iterate over the forms in an application.

C++ Examples:

```
/*
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->CustomFormCount; i++)
    ListBox1->Items->Add(Screen->CustomForms[i]->Name);
}
```

Delphi Examples:

```
{
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Screen.CustomFormCount-1 do
    ListBox1.Items.Add(Screen.CustomForms[I].Name);
end;
```

See Also

CustomFormCount (see page 2460)

Forms (see page 2468)

7.1.11.5.9 TScreen.DataModuleCount Property

Indicates the number of data modules that are currently instantiated in the application.

Pascal

```
property DataModuleCount: Integer;
```

C++

```
__property int DataModuleCount;
```

Description

Read DataModuleCount to learn the number of data modules in the DataModules (see page 2462) property. DataModuleCount can be used with DataModules (see page 2462) when an application needs to iterate over all its data

modules.

See Also

DataModules (see page 2462)

TDataModule

7.1.11.5.10 TScreen.DataModules Property

Lists all the data modules that are currently instantiated in the application.

Pascal

```
property DataModules [Index: Integer]: TDataModule;
```

C++

```
__property TDataModule DataModules[int Index];
```

Description

Use DataModules to access an instantiated data module by index. The value of Index is a number between zero (the first data module) and DataModuleCount (see page 2461) - 1. DataModules can be used with DataModuleCount (see page 2461) when an application needs to iterate over all its data modules.

See Also

DataModuleCount (see page 2461)

7.1.11.5.11 TScreen.DefaultIme Property

Indicates which input method editor (IME) is active when the TScreen (see page 2445) object is created.

Pascal

```
property DefaultIme: string;
```

C++

```
__property UnicodeString DefaultIme;
```

Description

In applications that support Asian language characters, read DefaultIme to determine the IME that is active when the application starts.

An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application. There can be multiple IMEs installed on a single system. They vary in user interface, conversion dictionaries, and input method.

Individual controls can override the default IME. When a control specifies a different IME should be used to enter information into the control, the IME that processes keyboard input changes when the control has focus. When the control loses focus, DefaultIme is restored as the system IME.

See Also

DefaultKbLayout (see page 2463)

Imes (see page 2472)

ImeName

ResetIme

7.1.11.5.12 TScreen.DefaultKbLayout Property

Indicates the Windows handle to the keyboard layout that was active when the application started running.

Pascal

```
property DefaultKbLayout: HKL;
```

C++

```
__property HKL DefaultKbLayout;
```

Description

Use DefaultKbLayout for making Windows API calls that require a handle to a keyboard layout. For example, if an application changes the keyboard layout to match the settings for another country, the value of DefaultKbLayout can be used as a parameter to the Windows ActivateKeyboardLayout function in order to restore the original layout.

For asian keyboards, the keyboard layout identifies the input method editor (IME) as well as the default language. Use DefaultKbLayout in Windows API calls that work with the IME. DefaultKbLayout is the keyboard layout handle associated with the IME named by the DefaultIme (see page 2462) property.

See Also

DefaultIme (see page 2462)

Imes (see page 2472)

7.1.11.5.13 TScreen.DesktopHeight Property

Specifies the height of the entire virtual desktop.

Pascal

```
property DesktopHeight: Integer;
```

C++

```
__property int DesktopHeight;
```

Description

Use DesktopHeight to determine the height of the entire desktop. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopHeight corresponds to Height (see page 2470).

See Also

DesktopLeft (see page 2463)

DesktopRect (see page 2464)

DesktopTop (see page 2465)

DesktopWidth (see page 2465)

WorkAreaHeight (see page 2476)

Height (see page 2470)

7.1.11.5.14 TScreen.DesktopLeft Property

Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopLeft: Integer;
```

C++

```
__property int DesktopLeft;
```

Description

Use DesktopLeft to determine the x-coordinate of the entire desktop's left edge relative to the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopLeft is 0.

See Also

DesktopHeight (see page 2463)

DesktopRect (see page 2464)

DesktopTop (see page 2465)

DesktopWidth (see page 2465)

WorkAreaLeft (see page 2477)

Primary (see page 2443)

Monitors (see page 2474)

7.1.11.5.15 TScreen.DesktopRect Property

Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopRect: TRect;
```

C++

```
__property TRect DesktopRect;
```

Description

Use DesktopRect to determine the coordinates of the entire virtual desktop, which includes all monitors in the system. DesktopRect is expressed in coordinates where (0,0) is the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true.

See Also

DesktopHeight (see page 2463)

DesktopLeft (see page 2463)

DesktopTop (see page 2465)

DesktopWidth (see page 2465)

WorkAreaRect (see page 2477)

Primary (see page 2443)

Monitors (see page 2474)

7.1.11.5.16 TScreen.DesktopTop Property

Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopTop: Integer;
```

C++

```
__property int DesktopTop;
```

Description

Use DesktopTop to determine the y-coordinate of the desktop's top edge relative to the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopTop is 0.

See Also

DesktopHeight (see page 2463)

DesktopRect (see page 2464)

DesktopLeft (see page 2463)

DesktopWidth (see page 2465)

WorkAreaTop (see page 2478)

Primary (see page 2443)

Monitors (see page 2474)

7.1.11.5.17 TScreen.DesktopWidth Property

Specifies the width of the entire virtual desktop.

Pascal

```
property DesktopWidth: Integer;
```

C++

```
__property int DesktopWidth;
```

Description

Use DesktopWidth to determine the width of the desktop. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopWidth corresponds to Width (see page 2475).

See Also

DesktopHeight (see page 2463)

DesktopRect (see page 2464)

DesktopLeft (see page 2463)

DesktopTop (see page 2465)

WorkAreaWidth (see page 2478)

Width (see page 2475)

7.1.11.5.18 TScreen.FocusedForm Property

Provides a reference to the form that has focus.

Pascal

```
property FocusedForm: TCustomForm;
```

C++

```
__property TCustomForm FocusedForm;
```

Description

FocusedForm returns a reference to the form that currently has focus.

See Also

TCustomForm (see page 2103)

7.1.11.5.19 TScreen.Fonts Property

Lists the face names for all fonts supported by the screen.

Pascal

```
property Fonts: TStrings;
```

C++

```
__property TStrings Fonts;
```

Description

Read Fonts to learn what screen fonts are currently installed. Applications can use Fonts to ensure that they do not request a font that is not installed on the user's system. When an application uses a TFont (see page 2730) object to request a font that is not installed, the system substitutes another font, which may not be a font appropriate to the application's needs.

Note: Fonts is a list of screen fonts. It does not give any indication of the printer fonts available to the application.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

TFont (see page 2730)

ResetFonts (see page 2451)

7.1.11.5.20 TScreen.FormCount Property

Indicates the number of forms displayed on the screen.

Pascal

```
property FormCount: Integer;
```

C++

```
__property int FormCount;
```

Description

Read FormCount to learn the number of forms currently displayed on the screen. These forms can be accessed by the Forms (see page 2468) property. FormCount can be used with Forms (see page 2468) to iterate over all the forms in an application.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
```

```
      NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
      NewItem->Caption = Screen->Forms[I]->Name + "Item";
      NewItem->Name = Screen->Forms[I]->Name;
      MainMenu1->Items->Items[1]->Add(NewItem);
    }
  }
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
```

See Also

Forms (see page 2468)

TForm (see page 2241)

7.1.11.5.21 TScreen.Forms Property

Lists all the forms currently displayed in the application.

Pascal

```
property Forms [Index: Integer]: TForm;
```

C++

```
__property TForm * Forms[int Index];
```

Description

Use Forms to access a form by index. The value of Index is a number between zero (the first form) and FormCount (see page 2467) - 1. Forms can be used with FormCount (see page 2467) when an application needs to iterate over all its forms, including all dialogs.

Forms only lists the TForm (see page 2241) descendants in the application. This does not include, for example, property pages. To get a list that includes TCustomForm (see page 2103) descendants that do not descend from TForm (see page 2241), use CustomForms (see page 2460) instead.

Warning: The order in which Forms lists its forms is affected by the Z order of the forms. Do not change the Z order of forms when using Forms to iterate over the forms in an application.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items->Items[1]->Add(NewItem);
  }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
```

```
    for  I := 0 to Screen.FormCount-1 do
    begin
      NewItem := TMenuItem.Create(MainMenu1.Items[1]);
      NewItem.Caption := Screen.Forms[I].Name + 'Item';
      NewItem.Name := Screen.Forms[I].Name;
      MainMenu1.Items[1].Add(NewItem);
     end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I : Integer;
  begin
    // Count down, not up!
    for  I := MainMenu1.Items[1].Count - 1 downto 0 do
    begin
      MainMenu1.Items[1].Items[I].Free;
    end;
  end;
```

See Also

FormCount (see page 2467)

CustomForms (see page 2460)

7.1.11.5.22 TScreen.Height Property

Indicates the vertical size of the screen in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Read Height to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded.

C++ Examples:

```
/*
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->FormCount; i++)
  {
    if (Screen->Forms[i]->Top < 0)
    {
      Screen->Forms[i]->Height += Screen->Forms[i]->Top;
      Screen->Forms[i]->Top = 0;
    }
    if (Screen->Forms[i]->Left < 0)
    {
      Screen->Forms[i]->Width += Screen->Forms[i]->Left;
      Screen->Forms[i]->Left = 0;
    }
    if (Screen->Forms[i]->Left + Screen->Forms[i]->Width > Screen->Width)
      Screen->Forms[i]->Width = Screen->Width - Screen->Forms[i]->Left;
    if (Screen->Forms[i]->Top + Screen->Forms[i]->Height > Screen->Height)
      Screen->Forms[i]->Height = Screen->Height - Screen->Forms[i]->Top;
```

```
  }
}
```

Delphi Examples:

```
{
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
}
procedure TForm1.Button1Click(Sender: TObject);
var I: Integer;
begin
with Screen do
  for I := 0 to FormCount-1 do
    begin
    if Forms[I].Top < 0 then
    begin
      Forms[I].Height := Forms[I].Height + Forms[I].Top;
      Forms[I].Top := 0;
    end;
    if Forms[I].Left < 0 then
    begin
      Forms[I].Width := Forms[I].Width + Forms[I].Left;
      Forms[I].Left := 0;
    end;
    if Forms[I].Left + Forms[I].Width > Width then
      Forms[I].Width := Width - Forms[I].Left;
    if Forms[I].Top + Forms[I].Height > Height then
      Forms[I].Height := Height - Forms[I].Top;
    end;
end;
```

See Also

PixelsPerInch (see page 2474)

Height

Width (see page 2475)

WorkAreaHeight (see page 2476)

DesktopHeight (see page 2463)

Considerations When Dynamically Resizing Forms and Controls

7.1.11.5.23 TScreen.HintFont Property

Specifies the font used to display help hints.

Pascal

```
property HintFont: TFont;
```

C++

```
__property TFont HintFont;
```

Description

HintFont stores the Windows status font setting. This font is used by default for the help hints that appear when the user pauses the mouse over a control or menu item.

See Also

Fonts (see page 2466)

IconFont (see page 2472)

MenuFont (see page 2473)

Font

Hint

Hint (see page 1976)

7.1.11.5.24 TScreen.IconFont Property

Specifies the font used to display the labels on the icons in file selection dialogs.

Pascal

```
property IconFont: TFont;
```

C++

```
__property TFont IconFont;
```

Description

IconFont stores the Windows icon font setting. This font is used to label the icons on the desktop, and icons in the file selection dialogs.

See Also

Fonts (see page 2466)

HintFont (see page 2471)

MenuFont (see page 2473)

7.1.11.5.25 TScreen.Imes Property

Lists all input method editors (IMEs) installed on the system.

Pascal

```
property Imes: TStrings;
```

C++

```
__property TStrings Imes;
```

Description

Read Imes to obtain a list of all IMEs that are listed in the Windows registry. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application. IMEs vary in user interface, conversion dictionaries, and input method.

Individual controls can require a specific IME to process the keyboard input when they have focus. To specify an IME, the control sets its ImeName property to one of the strings in Imes.

Each IME name in Imes has an associated keyboard layout handle. A keyboard layout handle is a Windows handle that identifies the primary language and IME used to process keyboard input. Use the keyboard layout handle in Windows API calls for programming the IME. To get the keyboard layout handle for a particular IME, a control can use code such as the following:

```
HKeyboard := Screen.Imes.Objects(Screen.Imes.IndexOf(ImeName));
HKeyboard = Screen->Imes->Objects(Screen->Imes->IndexOf(ImeName));
```

See Also

DefaultIme (see page 2462)

DefaultKbLayout (see page 2463)

TStrings

ImeName

7.1.11.5.26 TScreen.MenuFont Property

Specifies the font used for menu commands.

Pascal

```
property MenuFont: TFont;
```

C++

```
__property TFont MenuFont;
```

Description

MenuFont stores the Windows menu font setting. This font is used for menu items in main menus or popup menus that do not draw their own menu items in an OnDrawItem event handler.

See Also

Fonts (see page 2466)

IconFont (see page 2472)

HintFont (see page 2471)

OnDrawItem (see page 4928)

7.1.11.5.27 TScreen.MessageFont Property

Pascal

```
property MessageFont: TFont;
```

C++

```
__property TFont MessageFont;
```

Description

This is MessageFont, a member of class TScreen.

7.1.11.5.28 TScreen.MonitorCount Property

Specifies the number of monitors used to comprise the desktop.

Pascal

```
property MonitorCount: Integer;
```

C++

```
__property int MonitorCount;
```

Description

Use MonitorCount to determine the number of monitors used to comprise the desktop.

See Also

Monitors (see page 2474)

DesktopRect (see page 2464)

7.1.11.5.29 TScreen.Monitors Property

Provides access to an individual monitor used to comprise the desktop.

Pascal

```
property Monitors [Index: Integer]: TMonitor;
```

C++

```
__property TMonitor Monitors[int Index];
```

Description

Use Monitors to access an individual monitor used to comprise the desktop.

The Monitor with its Primary property set to true is the primary monitor.

See Also

MonitorCount (see page 2473)

DesktopRect (see page 2464)

Primary (see page 2443)

7.1.11.5.30 TScreen.PixelsPerInch Property

Indicates the number of screen pixels that make up a logical inch in the vertical direction.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Read PixelsPerInch to convert between measurements in pixels and measurements in logical inches. PixelsPerInch is only accurate for vertical measurements. Most screens have a different scaling factor for horizontal measurements. The value in PixelsPerInch is set from system when the application loads.

See Also

Height (see page 2470)

PixelsPerInch (see page 2179)

7.1.11.5.31 TScreen.PrimaryMonitor Property

Provides a reference to the primary monitor.

Pascal

```
property PrimaryMonitor: TMonitor;
```

C++

```
__property TMonitor PrimaryMonitor;
```

Description

The coordinates of Monitors (see page 2474) in a multi-monitor configuration are related to the primary monitor. PrimaryMonitor returns a reference to the primary monitor. Use the Primary property to determine whether a specific monitor is

the primary one.

See Also

TMonitor (see page 2439)

7.1.11.5.32 TScreen.SaveFocusedList Property

Pascal

```
property SaveFocusedList: TList;
```

C++

```
__property TList SaveFocusedList;
```

Description

This is SaveFocusedList, a member of class TScreen.

7.1.11.5.33 TScreen.Width Property

Indicates the horizontal size of the screen in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Read Width to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded.

C++ Examples:

```
/*
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->FormCount; i++)
  {
    if (Screen->Forms[i]->Top < 0)
    {
      Screen->Forms[i]->Height += Screen->Forms[i]->Top;
      Screen->Forms[i]->Top = 0;
    }
    if (Screen->Forms[i]->Left < 0)
    {
      Screen->Forms[i]->Width += Screen->Forms[i]->Left;
      Screen->Forms[i]->Left = 0;
    }
    if (Screen->Forms[i]->Left + Screen->Forms[i]->Width > Screen->Width)
      Screen->Forms[i]->Width = Screen->Width - Screen->Forms[i]->Left;
    if (Screen->Forms[i]->Top + Screen->Forms[i]->Height > Screen->Height)
      Screen->Forms[i]->Height = Screen->Height - Screen->Forms[i]->Top;
  }
}
```

Delphi Examples:

```
{
```

```
  The following code determines the height of all forms on the
  screen and resizes any that are taller than the screen height.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var I: Integer;
  begin
  with Screen do
    for I := 0 to FormCount-1 do
      begin
      if Forms[I].Top < 0 then
      begin
        Forms[I].Height := Forms[I].Height + Forms[I].Top;
        Forms[I].Top := 0;
      end;
      if Forms[I].Left < 0 then
      begin
        Forms[I].Width := Forms[I].Width + Forms[I].Left;
        Forms[I].Left := 0;
      end;
      if Forms[I].Left + Forms[I].Width > Width then
        Forms[I].Width := Width - Forms[I].Left;
      if Forms[I].Top + Forms[I].Height > Height then
        Forms[I].Height := Height - Forms[I].Top;
      end;
  end;
```

See Also

Height (see page 2470)

Width

DesktopWidth (see page 2465)

WorkAreaWidth (see page 2478)

Considerations When Dynamically Resizing Forms and Controls

7.1.11.5.34 TScreen.WorkAreaHeight Property

Specifies the height of the work area on the primary monitor.

Pascal

```
property WorkAreaHeight: Integer;
```

C++

```
__property int WorkAreaHeight;
```

Description

Use WorkAreaHeight to determine the height of the usable space on the primary monitor. WorkAreaHeight differs from the Height (see page 2470) property in that it adjusts the work area to subtract any space occupied by the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaLeft (see page 2477)

WorkAreaRect (see page 2477)

WorkAreaTop (see page 2478)

WorkAreaWidth (see page 2478)

DesktopHeight (see page 2463)

Height (see page 2470)

Monitors (see page 2474)

Primary (see page 2443)

7.1.11.5.35 TScreen.WorkAreaLeft Property

Specifies the left edge of the work area on the primary monitor.

Pascal

```
property WorkAreaLeft: Integer;
```

C++

```
__property int WorkAreaLeft;
```

Description

Use WorkAreaLeft to determine the left edge of the usable space on the primary monitor. This value indicates the number of pixels between the left edge of the primary monitor, and the space left after accounting for the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 2476)

WorkAreaRect (see page 2477)

WorkAreaTop (see page 2478)

WorkAreaWidth (see page 2478)

DesktopLeft (see page 2463)

Primary (see page 2443)

Monitors (see page 2474)

7.1.11.5.36 TScreen.WorkAreaRect Property

Specifies the boundaries of the usable work area on the primary monitor.

Pascal

```
property WorkAreaRect: TRect;
```

C++

```
__property TRect WorkAreaRect;
```

Description

Use WorkAreaRect to determine the coordinates of the usable space on the primary monitor. The usable work space is the surface of the primary monitor with the space occupied by the task bar and any tool bars subtracted off.

The primary monitor is the monitor with its Primary property set to true.

See Also

WorkAreaHeight (see page 2476)

WorkAreaLeft (see page 2477)

WorkAreaTop (see page 2478)

WorkAreaWidth (see page 2478)

DesktopRect (see page 2464)

Primary (see page 2443)

Monitors (see page 2474)

WorkAreaRect (see page 2444)

7.1.11.5.37 TScreen.WorkAreaTop Property

Specifies the top edge of the work area on the primary monitor.

Pascal

```
property WorkAreaTop: Integer;
```

C++

```
__property int WorkAreaTop;
```

Description

Use WorkAreaTop to determine the top edge of the usable space on the primary monitor. This value indicates the number of pixels between the top edge of the primary monitor, and the space left after accounting for the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 2476)

WorkAreaRect (see page 2477)

WorkAreaLeft (see page 2477)

WorkAreaWidth (see page 2478)

DesktopTop (see page 2465)

Primary (see page 2443)

Monitors (see page 2474)

7.1.11.5.38 TScreen.WorkAreaWidth Property

Specifies the width of the work area on the primary monitor.

Pascal

```
property WorkAreaWidth: Integer;
```

C++

```
__property int WorkAreaWidth;
```

Description

Use WorkAreaWidth to determine the width of the usable space on the primary monitor. WorkAreaWidth differs from the Width (see page 2475) property in that it adjusts the work area to subtract any space occupied by the task bar and any tool bars that

sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 2476)

WorkAreaRect (see page 2477)

WorkAreaLeft (see page 2477)

WorkAreaTop (see page 2478)

WorkAreaWidth

Width (see page 2475)

7.1.11.6 TScreen Events

The events of the TScreen class are listed here.

Public Events

	Name	Description
	OnActiveControlChange (see page 2479)	Occurs immediately after input focus changes to a new windowed control.
	OnActiveFormChange (see page 2481)	Occurs immediately after a new form becomes active in a multi-form application.

7.1.11.6.1 TScreen.OnActiveControlChange Event

Occurs immediately after input focus changes to a new windowed control.

Pascal

```
property OnActiveControlChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActiveControlChange;
```

Description

Write an OnActiveControlChange event handler to take specific action when input focus changes to a new control. The change in focus may be within the active form, or across forms to a new form that then becomes the active form.

When focus moves from one control to another, the following events occur.

1If the new focused control is in a different form, focus moves to the new form.

2Focus moves to the new active control.

3If the active form changed, an OnActiveFormChange (see page 2481) event occurs.

4An OnActiveControlChange event occurs.

C++ Examples:

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
*/
```

```
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
}

type
  TForm1 = class(TForm)
    StatusBar1, StatusBar2: TStatusBar;
    Button1: TButton;
    RadioButton1: TRadioButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure MouseOverChanged(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ActiveControlChanged(Sender: TObject);
  end;

var
  Active : TWinControl;
  I: Integer;
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.ActiveControlChanged(Sender: TObject);
var
  Active : TWinControl;
  I: Integer;
begin
  Active := nil;
  for I:= 0 to ControlCount -1 do
  begin
    if (Controls[I] is TWinControl) then
      if (Controls[I] as TWinControl).Focused then
        Active := TWinControl(Controls[I]);
  end;
  if ((Active <> nil) and (Active.Hint <> '')) then
  begin
    StatusBar2.SimpleText := GetLongHint(Active.Hint) + ' focus';
  end
end;

procedure TForm1.MouseOverChanged(Sender: TObject);
var
  Active : TWinControl;
  I: Integer;
begin
  if (Sender is TWinControl) then
  begin
    Active := TWinControl(Sender);
    StatusBar1.SimpleText := GetLongHint(Active.Hint) + ' mouse over';
  end;
end;

{
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Screen.OnActiveControlChange := ActiveControlChanged;
end;

{
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
}
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Screen.OnActiveControlChange := nil;
end;
```

See Also

ActiveControl (see page 2452)

OnActiveFormChange (see page 2481)

SetFocusedControl (see page 2136)

7.1.11.6.2 TScreen.OnActiveFormChange Event

Occurs immediately after a new form becomes active in a multi-form application.

Pascal

```
property OnActiveFormChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActiveFormChange;
```

Description

Write an OnActiveFormChange event handler to take specific action when a new form becomes active. OnActiveFormChange occurs when the active form for the application changes, not when a form becomes active because the application becomes active.

C++ Examples:

```
/*
This example uses two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the definition of the TForm1 class.
*/
#include "Unit2.h"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::ColorForm(TObject *Sender)
{
  Color = clWhite;
  Form2->Color = clWhite;
  Screen->ActiveForm->Color = clAqua;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveFormChange = ColorForm;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // This stops ColorForm from being called during termination.
  Screen->OnActiveFormChange = NULL;
}
```

Delphi Examples:

```
{
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the header for form 1's unit.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure ColorForm(Sender: TObject);
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  uses Unit2;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.ColorForm(Sender: TObject);
  begin
    Color := clWhite;
    Form2.Color := clWhite;
    Screen.ActiveForm.Color := clAqua;
  end;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Screen.OnActiveFormChange := ColorForm;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    // This stops ColorForm from being called during termination.
    Screen.OnActiveFormChange := nil;
  end;
```

See Also

ActiveCustomForm (see page 2452)

OnActiveControlChange (see page 2479)

SetFocus (see page 2135)

7.1.12 TScrollBox Class

TScrollBox represents a scrolling area (a scroll box) in a window.

Class Hierarchy

Pascal

```
  TScrollBox = class(TScrollingWinControl);
```

C++

```
  class TScrollBox : public TScrollingWinControl;
```

File

Forms

Description

Use TScrollBox to create a scroll box in a window.

One use of a scroll box is to prevent areas of a window, such as a toolbar or status bar built with TPanel components, from scrolling. To prevent a toolbar and status bar from scrolling, hide the window's scroll bars, and then position a scroll box in the client area of the window between the toolbar and status bar. The scroll bars associated with the scroll box will appear to belong to the window, but will scroll only the area inside the scroll box.

Another use of scroll boxes is to create multiple scrolling areas (views) in a window. Views are common in commercial word-processor, spreadsheet, and project management applications.

A scroll box can contain objects, such as TButton and TCheckBox objects.

See Also

TForm (see page 2241)

TFrame (see page 2366)

Grouping Controls

Scroll Boxes

7.1.12.1 TScrollBox Members

The following tables list the members exposed by TScrollBox.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2487)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	PaintWindow (see page 2487)	Renders the image of a windowed control.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.

Public Methods

	Name	Description
	Create (see page 2486)	Creates and initializes a new TScrollBox (see page 2483) object.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 2517)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2518)	Occurs when the user clicks the control.

	OnConstrainedResize (see page 2520)	Adjust resize constraints.
	OnContextPopup (see page 2520)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2521)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2521)	Occurs when another control is docked to the control.
	OnDockOver (see page 2523)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2525)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2528)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2531)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2532)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2533)	Occurs when a control receives the input focus.
	OnExit (see page 2535)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2538)	Returns the control's docking information.
	OnMouseActivate (see page 2540)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2541)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2543)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2544)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2544)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2546)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2548)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2549)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2549)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 2549)	Occurs immediately after the control is resized.
	OnStartDock (see page 2550)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2551)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2551)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 2488)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2492)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2492)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 2493)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 2493)	Specifies which edges of the control are beveled.
	BevelInner (see page 2493)	Specifies the cut of the inner bevel.
	BevelKind (see page 2494)	Specifies the control's bevel style.
	BevelOuter (see page 2494)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2495)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2495)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2496)	Specifies whether the scroll box has a border.
	Color (see page 2496)	Specifies the background color of the control.
	Constraints (see page 2497)	Specifies the size constraints for the control.
	Ctl3D (see page 2497)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2498)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2499)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	DragCursor (see page 2501)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2501)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2501)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2503)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2503)	Controls the attributes of text written on or in the control.
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	Padding (see page 2504)	Specifies the padding of a control.
	ParentBackground (see page 2505)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2505)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2505)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2506)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2506)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2506)	Determines where a control looks for its font information.
	ParentShowHint (see page 2507)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2507)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2512)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2514)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2515)	Determines if the user can tab to a control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 2515)	Determines whether the component appears onscreen.

7.1.12.2 TScrollBox.Create Constructor

Creates and initializes a new TScrollBox (see page 2483) object.

Pascal

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

C++

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

Description

Use Create to create and initialize a new TScrollBox (see page 2483) object.

AOwner is a component, typically the form, that is responsible for freeing the scroll box.

See Also

Owner

7.1.12.3 TScrollBox Methods

The methods of the TScrollBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2487)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	PaintWindow (see page 2487)	Renders the image of a windowed control.

7.1.12.3.1 TScrollBox.CreateParams Method

Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams calls the inherited method and uses that Window class style, removing the CS_HREDRAW and CS_VREDRAW styles from it.

By removing these styles the scrolling windowed control will only redraw the invalidated regions, not the entire window.

See Also

CreateWnd (see page 2559)

CreateWindowHandle

7.1.12.3.2 TScrollBox.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler

PaintTo

DoubleBuffered

7.1.12.4 TScrollBox Properties

The properties of the TScrollBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2488)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2492)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 2492)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

	AutoSize (see page 2493)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 2493)	Specifies which edges of the control are beveled.
	BevelInner (see page 2493)	Specifies the cut of the inner bevel.
	BevelKind (see page 2494)	Specifies the control's bevel style.
	BevelOuter (see page 2494)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2495)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2495)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2496)	Specifies whether the scroll box has a border.
	Color (see page 2496)	Specifies the background color of the control.
	Constraints (see page 2497)	Specifies the size constraints for the control.
	Ctl3D (see page 2497)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2498)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2499)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2501)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2501)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2501)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2503)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2503)	Controls the attributes of text written on or in the control.
	Padding (see page 2504)	Specifies the padding of a control.
	ParentBackground (see page 2505)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2505)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2505)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2506)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2506)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2506)	Determines where a control looks for its font information.
	ParentShowHint (see page 2507)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2507)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2512)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2514)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2515)	Determines if the user can tab to a control.
	Visible (see page 2515)	Determines whether the component appears onscreen.

7.1.12.4.1 TScrollBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
```

```
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
```

```
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
```

```
end;
```

See Also

Parent

Anchors

7.1.12.4.2 TScrollBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.12.4.3 TScrollBox.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.12.4.4 TScrollBox.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

7.1.12.4.5 TScrollBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

7.1.12.4.6 TScrollBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

7.1.12.4.7 TScrollBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

7.1.12.4.8 TScrollBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

7.1.12.4.9 TScrollBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

7.1.12.4.10 TScrollBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

7.1.12.4.11 TScrollBox.BorderStyle Property

Specifies whether the scroll box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set whether the scroll box has a border.

7.1.12.4.12 TScrollBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

7.1.12.4.13 TScrollBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

7.1.12.4.14 TScrollBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

7.1.12.4.15 TScrollBox.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
```

7.1.12.4.16 TScrollBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

7.1.12.4.17 TScrollBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

7.1.12.4.18 TScrollBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

7.1.12.4.19 TScrollBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

7.1.12.4.20 TScrollBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

7.1.12.4.21 TScrollBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

7.1.12.4.22 TScrollBox.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

7.1.12.4.23 TScrollBox.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

7.1.12.4.24 TScrollBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

7.1.12.4.25 TScrollBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

7.1.12.4.26 TScrollBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

7.1.12.4.27 TScrollBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

7.1.12.4.28 TScrollBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

7.1.12.4.29 TScrollBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

7.1.12.4.30 TScrollBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}
```

```
/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
```

```
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

7.1.12.4.31 TScrollBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

7.1.12.4.32 TScrollBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

7.1.12.4.33 TScrollBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

7.1.12.4.34 TScrollBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
```

```
    {
      TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this); // The owner will clean this up.
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

7.1.12.5 TScrollBox Events

The events of the TScrollBox class are listed here.

Published Events

	Name	Description
	OnCanResize (see page 2517)	Occurs when an attempt is made to resize the control.
	OnClick (see page 2518)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 2520)	Adjust resize constraints.
	OnContextPopup (see page 2520)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2521)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2521)	Occurs when another control is docked to the control.
	OnDockOver (see page 2523)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2525)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2528)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2531)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2532)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2533)	Occurs when a control receives the input focus.
	OnExit (see page 2535)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2538)	Returns the control's docking information.
	OnMouseActivate (see page 2540)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2541)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2543)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2544)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2544)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2546)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2548)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2549)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2549)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 2549)	Occurs immediately after the control is resized.
	OnStartDock (see page 2550)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2551)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2551)	Occurs when the application tries to undock a control that is docked to the windowed control.

7.1.12.5.1 TScrollBox.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize

SetBounds

OnResize

7.1.12.5.2 TScrollBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

7.1.12.5.3 TScrollBox.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize

SetBounds

Constraints

OnResize

CanAutoSize

7.1.12.5.4 TScrollBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by

a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

7.1.12.5.5 TScrollBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

7.1.12.5.6 TScrollBox.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
```

```
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
```

```
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockOver

7.1.12.5.7 TScrollBox.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
```

```
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
```

```
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

7.1.12.5.8 TScrollBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
```

```
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
```

```
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.12.5.9 TScrollBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.12.5.10 TScrollBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

7.1.12.5.11 TScrollBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

7.1.12.5.12 TScrollBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

7.1.12.5.13 TScrollBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
```

```
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

7.1.12.5.14 TScrollBox.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
```

```
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
```

```
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

7.1.12.5.15 TScrollBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.12.5.16 TScrollBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
```

```
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
```

```
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

7.1.12.5.17 TScrollBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

7.1.12.5.18 TScrollBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

7.1.12.5.19 TScrollBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

7.1.12.5.20 TScrollBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

7.1.12.5.21 TScrollBox.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel;
```

C++

```
__property OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or

OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown

OnMouseWheelUp

7.1.12.5.22 TScrollBox.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

7.1.12.5.23 TScrollBox.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

7.1.12.5.24 TScrollBox.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

C++ Examples:

```
/*
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
  Button1->Left = Form1->ClientWidth - Button1->Width;
}
```

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize

SetBounds

AutoSize

OnConstrainedResize

7.1.12.5.25 TScrollBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

7.1.12.5.26 TScrollBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

7.1.12.5.27 TScrollBox.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
```

```
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
```

```
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

7.1.13 TScrollingWinControl Class

TScrollingWinControl is the base class for controls that support scrolling.

Class Hierarchy

Pascal

```
TScrollingWinControl = class(TWinControl);
```

C++

```
class TScrollingWinControl : public TWinControl;
```

File

Forms

Description

Use TScrollingWinControl as a base class when defining a custom windowed control that supports scrolling.

Typically, a scrolling windowed control has horizontal and vertical scroll bars and scrolls a child control into view when the child control receives focus.

Examples of scrolling windowed controls include TForm (see page 2241) and TScrollBox (see page 2483) objects.

The scrolling window's BiDiMode determines whether the left or right side of the window's contents are displayed by default. The BiDiMode does not affect the window's coordinate system. The scroll bars must be adjusted programmatically according to the system locale.

See Also

TForm (see page 2241)

TScrollBox (see page 2483)

7.1.13.1 TScrollingWinControl Members

The following tables list the members exposed by TScrollingWinControl.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.
	DoFlipChildren (see page 2560)	Flips the control's children.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.

Protected Properties

	Name	Description
	AutoScroll (see page 2561)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Public Methods

	Name	Description
	Create (see page 2555)	Creates and initializes a new TScrollingWinControl (see page 2554) object.
	Destroy (see page 2556)	Removes the TScrollingWinControl (see page 2554) object from memory.
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.

7.1.13.2 TScrollingWinControl.Create Constructor

Creates and initializes a new TScrollingWinControl (see page 2554) object.

Pascal

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

C++

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

Description

Use Create to create and initialize a new TScrollingWinControl (see page 2554) object.

AOwner is a TComponent object that owns the scrolling windowed control.

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

Create

VertScrollBar (see page 2563)

7.1.13.3 TScrollingWinControl.Destroy Destructor

Removes the TScrollingWinControl (see page 2554) object from memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScrollingWinControl();
```

Description

Do not call Destroy.TScrollingWinControl (see page 2554) Instead, call Free, which calls DestroyWinControl if the TScrollingWinControl (see page 2554) object is not nil.

See Also

Destroy

System::TObject::Free

7.1.13.4 TScrollingWinControl Methods

The methods of the TScrollingWinControl class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2557)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 2557)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 2558)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 2558)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 2558)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 2559)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 2559)	Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.

	DoFlipChildren (see page 2560)	Flips the control's children.
	Resizing (see page 2561)	This is Resizing, a member of class TScrollingWinControl.

Public Methods

	Name	Description
	DisableAutoRange (see page 2559)	Disables automatic scrolling.
	EnableAutoRange (see page 2560)	Reenables automatic scrolling.
	ScrollInView (see page 2561)	Scrolls a control into the visible area of the scrolling windowed control.

7.1.13.4.1 TScrollingWinControl.AdjustClientRect Method

Adjusts the ClientRect property to exclude the scroll bars.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the control needs accurate information on where to place child controls within the client area. It indents Rect to exclude the area taken by the scroll bars, then calls the inherited AdjustClientRect method.

See Also

ClientRect

7.1.13.4.2 TScrollingWinControl.AlignControls Method

Aligns the controls in the scrolling windowed control.

Pascal

```
procedure AlignControls(AControl: TControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect ARect);
```

Description

Use AlignControls to align the controls in the scrolling control according to the values of their Align properties.

AControl specifies the control that takes precedence in alignment when other controls have the same Align value. AControl can be nil (Delphi) or NULL (C++).

ARect indicates the area in the scrolling control that contains the controls to align. ARect is automatically set to the entire area of the scrolling control.

If AutoScroll (see page 2561) is true and the scrolling control is not being resized, the ranges of the scroll bars on the scrolling control are automatically recalculated.

After recalculating the ranges of the scroll bars and setting the value of ARect, AlignControls calls the inherited AlignControls method, passing the AControl and ARect objects.

See Also

Align

AutoScroll (see page 2561)

AlignControls

7.1.13.4.3 TScrollingWinControl.AutoScrollEnabled Method

Indicates whether automatic scrolling is enabled.

Pascal

```
function AutoScrollEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AutoScrollEnabled();
```

Description

Call AutoScrollEnabled to determine whether automatic scrolling is enabled.

AutoScrollEnabled returns true if the control is not a dock site using the dock manager and the AutoSize property is false.

See Also

AutoScroll

AutoSize

UseDockManager

7.1.13.4.4 TScrollingWinControl.AutoScrollInView Method

Scrolls a control into the visible area of the scrolling windowed control.

Pascal

```
procedure AutoScrollInView(AControl: TControl); virtual;
```

C++

```
virtual __fastcall AutoScrollInView(TControl * AControl);
```

Description

Use AutoScrollInView to scroll the control specified by the AControl parameter into the visible area of the scrolling windowed control.

Unlike ScrollInView (see page 2561), AutoScrollInView does not perform scrolling when the scrolling windowed control or the specified control are being loaded from a stream.

AutoScrollInView calls ScrollInView (see page 2561) to perform any needed scrolling.

See Also

ScrollInView (see page 2561)

7.1.13.4.5 TScrollingWinControl.ChangeScale Method

Resizes the scrolling windowed control by a specific ratio.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

Use ChangeScale to resize the scrolling windowed control by a specific ratio.

M is the multiplier of the ratio; D is the divisor of the ratio. For example, to resize the scrolling windowed control to four-fifths of its current size, set M to four and D to five.

See Also

ChangeScale

7.1.13.4.6 TScrollingWinControl.CreateParams Method

Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2559) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams calls the inherited method and uses that Window class style, removing the CS_HREDRAW and CS_VREDRAW styles from it.

By removing these styles the scrolling windowed control will only redraw the invalidated regions, not the entire window.

See Also

CreateWnd (see page 2559)

CreateWindowHandle

7.1.13.4.7 TScrollingWinControl.CreateWnd Method

Creates a Windows control corresponding to the TScrollingWinControl (see page 2554) object.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the window must be destroyed and recreated to reflect property changes.

See Also

CreateWnd

DestroyWnd

7.1.13.4.8 TScrollingWinControl.DisableAutoRange Method

Disables automatic scrolling.

Pascal

```
procedure DisableAutoRange;
```

C++

```
__fastcall DisableAutoRange();
```

Description

Use DisableAutoRange to temporarily disable scrolling when rearranging controls in a scrolling windowed control. Use the EnableAutoRange (see page 2560) method to turn scrolling on again when finished rearranging the controls.

See Also

EnableAutoRange (see page 2560)

AutoScroll (see page 2561)

7.1.13.4.9 TScrollingWinControl.DoFlipChildren Method

Flips the control's children.

Pascal

```
procedure DoFlipChildren; override;
```

C++

```
virtual __fastcall DoFlipChildren();
```

Description

The FlipChildren method calls DoFlipChildren to reverse the positions of child controls. DoFlipChildren moves the control's children to opposite positions on the control; that is, children on the left side of the control are moved to the right side of the control and vice versa. In addition, DoFlipChildren moves the vertical scroll bars to the other side of the control.

Note: DoFlipChildren is a protected helper method for the public FlipChildren method. Unlike FlipChildren, DoFlipChildren does not adjust the alignment of controls that are aligned to one of the edges of the windowed control.

See Also

FlipChildren

BiDiMode

7.1.13.4.10 TScrollingWinControl.EnableAutoRange Method

Reenables automatic scrolling.

Pascal

```
procedure EnableAutoRange;
```

C++

```
__fastcall EnableAutoRange();
```

Description

Use EnableAutoRange to reenable scrolling after turning it off. This method can be used with the DisableAutoRange (see page 2559) method to turn scrolling on and off.

See Also

DisableAutoRange (see page 2559)

AutoScroll (see page 2561)

7.1.13.4.11 TScrollingWinControl.Resizing Method

Pascal

```
procedure Resizing(State: TWindowState); virtual;
```

C++

```
virtual __fastcall Resizing(TWindowState State);
```

Description

This is Resizing, a member of class TScrollingWinControl.

7.1.13.4.12 TScrollingWinControl.ScrollInView Method

Scrolls a control into the visible area of the scrolling windowed control.

Pascal

```
procedure ScrollInView(AControl: TControl);
```

C++

```
__fastcall ScrollInView(TControl * AControl);
```

Description

Use ScrollInView to ensure that a specified control is in the visible area of the scrolling windowed control. AControl specifies the TControl object to scroll into view.

Note: If the ComponentState property for AControl or the scrolling windowed control might be csLoading, use AutoScrollInView (see page 2558) instead.

See Also

AutoScrollInView (see page 2558)

ComponentState

7.1.13.5 TScrollingWinControl Properties

The properties of the TScrollingWinControl class are listed here.

Protected Properties

	Name	Description
	AutoScroll (see page 2561)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Published Properties

	Name	Description
	HorzScrollBar (see page 2562)	Represents the horizontal scroll bar for the scrolling windowed control.
	VertScrollBar (see page 2563)	Represents the vertical scroll bar for the scrolling windowed control.

7.1.13.5.1 TScrollingWinControl.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar (see page 2562) and VertScrollBar (see page 2563) properties to make scroll bars appear.

See Also

AutoScrollInView (see page 2558)

HorzScrollBar (see page 2562)

VertScrollBar (see page 2563)

7.1.13.5.2 TScrollingWinControl.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 2561)

VertScrollBar (see page 2563)

TControlScrollBar (see page 2009)

7.1.13.5.3 TScrollingWinControl.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
```

```
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 2561)

HorzScrollBar (see page 2562)

TControlScrollBar (see page 2009)

7.1.13.6 TScrollingWinControl Events

The events of the TScrollingWinControl class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2564)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2564)	Occurs when an object with custom alignment is aligned.

7.1.13.6.1 TScrollingWinControl.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

7.1.13.6.2 TScrollingWinControl.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

7.2 Interfaces

The following table lists interfaces in this documentation.

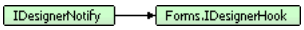
Interfaces

	Name	Description
	IDesignerHook (see page 2566)	IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.
	IOleForm (see page 2572)	IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

7.2.1 IDesignerHook Interface

IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.

Class Hierarchy



Pascal

```
IDesignerHook = interface(IDesignerNotify);
```

C++

```
__interface IDesignerHook : public IDesignerNotify;
```

File

Forms

Description

IDesignerHook provides access to the form currently under construction in the form designer, and influences the behavior of the designer by specifying whether the object under construction is a form (as opposed to a composite component), drawing the grid that assists in layout, and so on.

Component writers can access the current IDesignerHook interface using the Designer property of a form.

Note: C++ method declarations that use IDesignerHook use the _di_IDesignerHook type instead. This is a DelphiInterface wrapper around the IDesignerHook interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDesignerHook > _di_IDesignerHook;
```

See Also

TCustomForm (see page 2103)

TComponentEditor

TPropertyEditor

IFormDesigner

7.2.1.1 IDesignerHook Members

The following tables list the members exposed by IDesignerHook.

Public Methods

	Name	Description
	GetCustomForm (see page 2567)	Specifies the value returned when the Form (see page 2571) property is read.
	GetIsControl (see page 2568)	Specifies the value returned when the IsControl (see page 2571) property is read.
	GetRoot (see page 2568)	Returns the current entity being edited by the form designer.
	IsDesignMsg (see page 2568)	Determines when the designer should handle a Windows message.
	PaintGrid (see page 2569)	Paints the alignment grid on the form's canvas.
	PaintMenu (see page 2569)	This is PaintMenu, a member of class IDesignerHook.
	SetCustomForm (see page 2569)	Specifies what happens when the Form (see page 2571) property is set.
	SetIsControl (see page 2570)	Specifies what happens when the IsControl (see page 2571) property is set.
	UniqueName (see page 2570)	Generates a unique name from a specified base string.
	ValidateRename (see page 2570)	Restricts the names that can be given to components on the form that is being designed.

Public Properties

	Name	Description
	Form (see page 2571)	Provides a reference to the form currently being designed.
	IsControl (see page 2571)	Specifies whether the form is in its form state or its control state.

7.2.1.2 IDesignerHook Methods

The methods of the IDesignerHook class are listed here.

Public Methods

	Name	Description
	GetCustomForm (see page 2567)	Specifies the value returned when the Form (see page 2571) property is read.
	GetIsControl (see page 2568)	Specifies the value returned when the IsControl (see page 2571) property is read.
	GetRoot (see page 2568)	Returns the current entity being edited by the form designer.
	IsDesignMsg (see page 2568)	Determines when the designer should handle a Windows message.
	PaintGrid (see page 2569)	Paints the alignment grid on the form's canvas.
	PaintMenu (see page 2569)	This is PaintMenu, a member of class IDesignerHook.
	SetCustomForm (see page 2569)	Specifies what happens when the Form (see page 2571) property is set.
	SetIsControl (see page 2570)	Specifies what happens when the IsControl (see page 2571) property is set.
	UniqueName (see page 2570)	Generates a unique name from a specified base string.
	ValidateRename (see page 2570)	Restricts the names that can be given to components on the form that is being designed.

7.2.1.2.1 IDesignerHook.GetCustomForm Method

Specifies the value returned when the Form (see page 2571) property is read.

Pascal

```
function GetCustomForm: TCustomForm;
```

C++

```
__fastcall TCustomForm GetCustomForm();
```

Description

Implement GetCustomForm to specify the value returned when the Form (see page 2571) property is read.

See Also

Form (see page 2571)

SetCustomForm (see page 2569)

7.2.1.2.2 IDesignerHook.GetIsControl Method

Specifies the value returned when the IsControl (see page 2571) property is read.

Pascal

```
function GetIsControl: Boolean;
```

C++

```
__fastcall Boolean GetIsControl();
```

Description

Implement GetIsControl to specify the value returned when the IsControl (see page 2571) property is read.

See Also

IsControl (see page 2571)

SetIsControl (see page 2570)

7.2.1.2.3 IDesignerHook.GetRoot Method

Returns the current entity being edited by the form designer.

Pascal

```
function GetRoot: TComponent;
```

C++

```
__fastcall TComponent * GetRoot();
```

Description

Call GetRoot to obtain the object the form designer is working on. If this object is a descendant of TCustomForm (see page 2103), GetRoot returns the value of the Form (see page 2571) property.

Use GetRoot rather than the Form (see page 2571) property, to ensure that code works even when editing a data module, remote data module, active form, or other entity that is not a descendant of TCustomForm (see page 2103).

See Also

Form (see page 2571)

ContainerWindow

7.2.1.2.4 IDesignerHook.IsDesignMsg Method

Determines when the designer should handle a Windows message.

Pascal

```
function IsDesignMsg(Sender: TControl; var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean IsDesignMsg(TControl * Sender, TMessage Message);
```

Description

IsDesignMsg is called for each message sent to a component in the designer. This method returns true if the message is a design message, meaning one the designer should handle for the component.

See Also

WindowProc

WndProc

7.2.1.2.5 IDesignerHook.PaintGrid Method

Paints the alignment grid on the form's canvas.

Pascal

```
procedure PaintGrid;
```

C++

```
__fastcall PaintGrid();
```

Description

Forms (see page 1921) that are associated with a designer call the PaintGrid method in response to Windows paint messages. If there is no designer associated with the form, the form's Paint method is called instead.

See Also

Paint (see page 2131)

7.2.1.2.6 IDesignerHook.PaintMenu Method

Pascal

```
procedure PaintMenu;
```

C++

```
__fastcall PaintMenu();
```

Description

This is PaintMenu, a member of class IDesignerHook.

7.2.1.2.7 IDesignerHook.SetCustomForm Method

Specifies what happens when the Form (see page 2571) property is set.

Pascal

```
procedure SetCustomForm(Value: TCustomForm);
```

C++

```
__fastcall SetCustomForm(TCustomForm Value);
```

Description

Implement SetCustomForm to specify what happens when the Form (see page 2571) property is set.

See Also

Form (see page 2571)

GetCustomForm (see page 2567)

7.2.1.2.8 IDesignerHook.SetIsControl Method

Specifies what happens when the IsControl (see page 2571) property is set.

Pascal

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

C++

```
__fastcall SetIsControl(Boolean Value);
```

Description

Implement SetIsControl specify what happens when the IsControl (see page 2571) property is set.

See Also

IsControl (see page 2571)

GetIsControl (see page 2568)

7.2.1.2.9 IDesignerHook.UniqueName Method

Generates a unique name from a specified base string.

Pascal

```
function UniqueName(const BaseName: string): string;
```

C++

```
__fastcall UnicodeString UniqueName(const UnicodeString BaseName);
```

Description

Call UniqueName to automatically generate a unique name for a component. Specify the base string for the name by the BaseName parameter. UniqueName appends a number to BaseName to ensure that there are no name-space conflicts.

See Also

GetComponentName

7.2.1.2.10 IDesignerHook.ValidateRename Method

Restricts the names that can be given to components on the form that is being designed.

Pascal

```
procedure ValidateRename(AComponent: TComponent; const CurName: string; const NewName: string);
```

C++

```
__fastcall ValidateRename(TComponent * AComponent, const UnicodeString CurName, const
UnicodeString NewName);
```

Description

Call ValidateRename to determine if the component specified by AComponent, which currently has the name specified by CurName, can be renamed to the string specified by NewName. This method is usually called by the form's ValidateRename

method to provide tighter restrictions on renaming, such as prohibiting reserved words.

See Also

ValidateRename (see page 2141)

7.2.1.3 IDesignerHook Properties

The properties of the IDesignerHook class are listed here.

Public Properties

	Name	Description
	Form (see page 2571)	Provides a reference to the form currently being designed.
	IsControl (see page 2571)	Specifies whether the form is in its form state or its control state.

7.2.1.3.1 IDesignerHook.Form Property

Provides a reference to the form currently being designed.

Pascal

```
property Form: TCustomForm;
```

C++

```
__property TCustomForm Form;
```

Description

Use Form to access the form, if any, that is being designed. When the user specifies changes in the form designer, the changes can be made to the form specified by the Form property.

Note: Not every designer has a form. For example, when the designer is used for a data module, remote data module, or active form, the Form property should not be used. Instead, use the object that the GetRoot (see page 2568) method returns.

See Also

Designer (see page 2159)

IsControl (see page 2571)

GetRoot (see page 2568)

7.2.1.3.2 IDesignerHook.IsControl Property

Specifies whether the form is in its form state or its control state.

Pascal

```
property IsControl: Boolean;
```

C++

```
__property Boolean IsControl;
```

Description

If the designer is used to create a control, rather than a form, set IsControl to true when the user finishes designing the control.

See Also

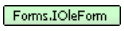
IsControl

Form (see page 2571)

7.2.2 IOleForm Interface

IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

Class Hierarchy



Pascal

```
IOleForm = interface;
```

C++

```
__interface IOleForm;
```

File

Forms

See Also

TOleControl (see page 6074)

7.2.2.1 IOleForm Members

The following tables list the members exposed by IOleForm.

Public Methods

	Name	Description
	OnDestroy (see page 2572)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 2573)	This is OnResize, a member of class IOleForm.

7.2.2.2 IOleForm Methods

The methods of the IOleForm class are listed here.

Public Methods

	Name	Description
	OnDestroy (see page 2572)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 2573)	This is OnResize, a member of class IOleForm.

7.2.2.2.1 IOleForm.OnDestroy Method

Pascal

```
procedure OnDestroy;
```

C++

```
__fastcall OnDestroy();
```

Description

This is OnDestroy, a member of class IOleForm.

7.2.2.2.2 IOleForm.OnResize Method

Pascal

```
procedure OnResize;
```

C++

```
__fastcall OnResize();
```

Description

This is OnResize, a member of class IOleForm.

7.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DisableTaskWindows (see page 2574)	This is function Forms.DisableTaskWindows.
	DoneCtl3D (see page 2574)	This is function Forms.DoneCtl3D.
	EnableTaskWindows (see page 2575)	This is function Forms.EnableTaskWindows.
	ForegroundTask (see page 2575)	Indicates whether the current thread is running in the foreground.
	GetParentForm (see page 2576)	Returns the form or property page that contains a specified control.
	InitCtl3D (see page 2576)	This is function Forms.InitCtl3D.
	IsAccel (see page 2576)	Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.
	KeyboardStateToShiftState (see page 2577)	Convert array of virtual keys to TShiftState.
	KeyboardStateToShiftState (see page 2577)	Convert array of virtual keys to TShiftState.
	KeyDataToShiftState (see page 2578)	Convert keyboard message data to TShiftState.
	KeysToShiftState (see page 2578)	Convert mouse event parameter to TShiftState.
	RestoreFocusState (see page 2579)	This is function Forms.RestoreFocusState.
	SaveFocusState (see page 2579)	This is function Forms.SaveFocusState.
	SetAutoSubClass (see page 2579)	This is function Forms.SetAutoSubClass.
	Subclass3DDlg (see page 2580)	This is function Forms.Subclass3DDlg.
	Subclass3DWnd (see page 2580)	This is function Forms.Subclass3DWnd.
	ValidParentForm (see page 2580)	Returns the form or property page that contains a specified control.

7.3.1 Forms.DisableTaskWindows Function

Pascal

```
function DisableTaskWindows(ActiveWindow: HWnd): TTaskWindowList;
```

C++

```
TTaskWindowList DisableTaskWindows(HWnd ActiveWindow);
```

File

Forms

Description

This is function Forms.DisableTaskWindows.

7.3.2 Forms.DoneCtl3D Function

Pascal

```
procedure DoneCtl3D; deprecated;
```

C++

```
DoneCtl3D();
```

File

Forms

Description

This is function Forms.DoneCtl3D.

7.3.3 Forms.EnableTaskWindows Function

Pascal

```
procedure EnableTaskWindows(WindowList: TTaskWindowList);
```

C++

```
EnableTaskWindows(TTaskWindowList WindowList);
```

File

Forms

Description

This is function Forms.EnableTaskWindows.

7.3.4 Forms.ForegroundTask Function

Indicates whether the current thread is running in the foreground.

Pascal

```
function ForegroundTask: Boolean;
```

C++

```
Boolean ForegroundTask();
```

File

Forms

Description

Call ForegroundTask to determine whether the current thread is executing in the Foreground. This is useful if you want to provide feedback to users only if they are currently interacting with a window on the same thread, but do not want to interrupt other processes.

ForegroundTask checks whether the window that currently has focus belongs to the thread that executes the function. It returns true if the focused window belongs to the current thread, false otherwise.

7.3.5 Forms.GetParentForm Function

Returns the form or property page that contains a specified control.

Pascal

```
function GetParentForm(Control: TControl; TopForm: Boolean = True): TCustomForm;
```

C++

```
TCustomForm GetParentForm(TControl * Control, Boolean TopForm = True);
```

File

Forms

Description

Call GetParentForm to obtain the descendant of TCustomForm (see page 2103) that contains the control specified by the **Control** parameter. If the **TopForm** parameter if true, this function goes all the way up the parent chain for **Control** until it finds a Form with no parent. If **TopForm** if false, this function goes up the parent chain until it finds a Form that is a TCustomForm (see page 2103).

If the control is not contained in a form (for example, an ActiveX control embedded in another type of window), GetParentForm returns **nil** (Delphi) or **NULL** (C++).

See Also

TControl

TCustomForm (see page 2103)

ValidParentForm (see page 2580)

7.3.6 Forms.InitCtl3D Function

Pascal

```
procedure InitCtl3D; deprecated;
```

C++

```
InitCtl3D();
```

File

Forms

Description

This is function Forms.InitCtl3D.

7.3.7 Forms.IsAccel Function

Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.

Pascal

```
function IsAccel(VK: Word; const Str: string): Boolean;
```

C++

```
Boolean IsAccel(Word VK, const UnicodeString Str);
```

File

Forms

Description

Call IsAccel to determine whether the character represented by the VK (virtual key) parameter appears as an accelerator in the string specified by the Str parameter. IsAccel only looks for accelerators that are identified by a preceding ampersand (&). Such accelerators appear underlined in the string when displayed as a control's caption.

Use IsAccel when processing key events to determine whether a key matches an accelerator character in a control's caption.

Note: IsAccel does not recognize hot key combinations in menu items that are identified by the HotKey property.

See Also

TCustomLabel

TCustomStaticText

TMenuItem

7.3.8 Forms.KeyboardStateToShiftState Function

Convert array of virtual keys to TShiftState.

Pascal

```
function KeyboardStateToShiftState(const KeyboardState: TKeyboardState): TShiftState; overload;
function KeyboardStateToShiftState: TShiftState; overload;
```

C++

```
TShiftState KeyboardStateToShiftState(const TKeyboardState KeyboardState);
TShiftState KeyboardStateToShiftState();
```

File

Forms

Description

Convert the given array of virtual keys **KeyboardState** to a TShiftState type. This array can be provided by the Windows function GetKeyboardState.

See Also

KeyDataToShiftState (see page 2578)

KeysToShiftState (see page 2578)

TShiftState

7.3.9 Forms.KeyDataToShiftState Function

Convert keyboard message data to TShiftState.

Pascal

```
function KeyDataToShiftState(KeyData: Longint): TShiftState;
```

C++

```
TShiftState KeyDataToShiftState(Longint KeyData);
```

File

Forms

Description

Convert the current keyboard state and given **KeyData** to a TShiftState type.

See Also

KeyboardStateToShiftState (see page 2577)

KeysToShiftState (see page 2578)

TShiftState

7.3.10 Forms.KeysToShiftState Function

Convert mouse event parameter to TShiftState.

Pascal

```
function KeysToShiftState(Keys: Word): TShiftState;
```

C++

```
TShiftState KeysToShiftState(Word Keys);
```

File

Forms

Description

Convert the parameter **Keys** to a TShiftState type. **Keys** contains flags that indicate the status of the mouse buttons and the CTRL and SHIFT keys at the time of the mouse event

See Also

KeyDataToShiftState (see page 2578)

KeyboardStateToShiftState (see page 2577)

TShiftState

7.3.11 Forms.RestoreFocusState Function

Pascal

```
procedure RestoreFocusState(FocusState: TFocusState);
```

C++

```
RestoreFocusState(TFocusState FocusState);
```

File

Forms

Description

This is function Forms.RestoreFocusState.

7.3.12 Forms.SaveFocusState Function

Pascal

```
function SaveFocusState: TFocusState;
```

C++

```
TFocusState SaveFocusState();
```

File

Forms

Description

This is function Forms.SaveFocusState.

7.3.13 Forms.SetAutoSubClass Function

Pascal

```
procedure SetAutoSubClass(Enable: Boolean); deprecated;
```

C++

```
SetAutoSubClass(Boolean Enable);
```

File

Forms

Description

This is function Forms.SetAutoSubClass.

7.3.14 Forms.Subclass3DDlg Function

Pascal

```
procedure Subclass3DDlg(Wnd: HWnd; Flags: Word); deprecated;
```

C++

```
Subclass3DDlg(HWnd Wnd, Word Flags);
```

File

Forms

Description

This is function Forms.Subclass3DDlg.

7.3.15 Forms.Subclass3DWnd Function

Pascal

```
function Subclass3DWnd(Wnd: HWnd): Boolean; deprecated;
```

C++

```
Boolean Subclass3DWnd(HWnd Wnd);
```

File

Forms

Description

This is function Forms.Subclass3DWnd.

7.3.16 Forms.ValidParentForm Function

Returns the form or property page that contains a specified control.

Pascal

```
function ValidParentForm(Control: TControl; TopForm: Boolean = True): TCustomForm;
```

C++

```
TCustomForm ValidParentForm(TControl * Control, Boolean TopForm = True);
```

File

Forms

Description

Call ValidParentForm to obtain the descendant of TCustomForm (see page 2103) that contains the control specified by the **Control** parameter. If the **TopForm** parameter if true, this function goes all the way up the parent chain for **Control** until it finds a Form with no parent. If **TopForm** if false, this function goes up the parent chain until it finds a Form that is a TCustomForm (see page 2103).

If the control is not contained in a form (for example, an ActiveX control embedded in another type of window), ValidParentForm raises an EInvalidOperation exception.

See Also

GetParentForm (see page 2576)

TControl

TCustomForm (see page 2103)

7.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActiveFormBorderStyle (see page 2582)	Specifies the style of an active form's border.
	TBorderIcon (see page 2583)	TBorderIcons (see page 2596) and TBorderIcons (see page 2596) specify the buttons that appear on the title bar of a form window.
	TCloseAction (see page 2584)	TCloseAction describes how a form should respond when it is closed.
	TDefaultMonitor (see page 2585)	Specifies the monitor on which the form appears.
	TFormBorderStyle (see page 2586)	TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 2597) is the type of the BorderStyle property for other controls in windows-only applications.
	TFormStyle (see page 2587)	TFormStyle describes form style.
	TMonitorDefaultTo (see page 2587)	TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.
	TPopupMode (see page 2588)	TPopupMode enumerates the possible values of a Form's PopupMode property.
	TPosition (see page 2590)	TPosition describes the positioning of a form.
	TPrintScale (see page 2591)	Represents the proportions of a printed form.
	TScrollBarKind (see page 2591)	TScrollBarKind indicates the orientation of a scrollbar.
	TScrollBarStyle (see page 2592)	TScrollBarStyle specifies a scroll bar style.
	TShowAction (see page 2592)	This is record Forms.TShowAction.
	TTileMode (see page 2593)	Represents how MDI child forms are arranged when the Tile method is called.
	TTimerMode (see page 2593)	This is record Forms.TTimerMode.
	TWindowState (see page 2594)	TWindowState indicates whether a form is maximized, minimized, or normally sized.

Records

	Name	Description
	TCMHintShowPause (see page 2584)	This is record Forms.TCMHintShowPause.
	TCursorRec (see page 2585)	This is record Forms.TCursorRec.
	TPopupForm (see page 2588)	This is record Forms.TPopupForm.
	TPopupWnd (see page 2589)	This is record Forms.TPopupWnd.

7.4.1 Forms.TActiveFormBorderStyle Enumeration

Specifies the style of an active form's border.

Pascal

```
TActiveFormBorderStyle = (
  afbNone,
  afbSingle,
  afbSunken,
  afbRaised
);
```

C++

```
enum TActiveFormBorderStyle {
  afbNone,
  afbSingle,
```

```
  afbSunken,
  afbRaised
};
```

File

Forms

Description

TActiveFormBorderStyle is an enumeration of possible active form border styles, and may have on of the following are possible values :

Value	Meaning
afbNone	The form has no border.
afbSingle	The form has a regular single-line border.
afbSunken	The form has a 3D look—that is, a border with a sunken edge.
afbRaised	The form has a border with a raised edge.

See Also

TCustomActiveForm (see page 2024)

7.4.2 Forms.TBorderIcon Enumeration

TBorderIcons (see page 2596) and TBorderIcons (see page 2596) specify the buttons that appear on the title bar of a form window.

Pascal

```
TBorderIcon = (
  biSystemMenu,
  biMinimize,
  biMaximize,
  biHelp
);
```

C++

```
enum TBorderIcon {
  biSystemMenu,
  biMinimize,
  biMaximize,
  biHelp
};
```

File

Forms

Description

TBorderIcons (see page 2596) can include any of the following TBorderIcon values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).
biMinimize	The form has a Minimize button
biMinimize	The form has a Maximize button

biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise, no question mark appears.

7.4.3 Forms.TCloseAction Enumeration

TCloseAction describes how a form should respond when it is closed.

Pascal

```
TCloseAction = (
  caNone,
  caHide,
  caFree,
  caMinimize
);
```

C++

```
enum TCloseAction {
  caNone,
  caHide,
  caFree,
  caMinimize
};
```

File

Forms

Description

TCloseAction consists of the following values:

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed.

7.4.4 Forms.TCMHintShowPause Record

Pascal

```
TCMHintShowPause = record
  Msg: Cardinal;
  WasActive: Integer;
  Pause: PInteger;
  Result: Integer;
end;
```

C++

```
struct TCMHintShowPause {
  unsigned Msg;
  int WasActive;
```

```
    PInteger Pause;
    int Result;
  };
```

File

Forms

Description

This is record Forms.TCMHintShowPause.

7.4.5 Forms.TCursorRec Record

Pascal

```
  TCursorRec = record
    Next: PCursorRec;
    Index: Integer;
    Handle: HCURSOR;
  end;
```

C++

```
  struct TCursorRec {
    PCursorRec Next;
    int Index;
    HCURSOR Handle;
  };
```

File

Forms

Description

This is record Forms.TCursorRec.

7.4.6 Forms.TDefaultMonitor Enumeration

Specifies the monitor on which the form appears.

Pascal

```
  TDefaultMonitor = (
    dmDesktop,
    dmPrimary,
    dmMainForm,
    dmActiveForm
  );
```

C++

```
  enum TDefaultMonitor {
    dmDesktop,
    dmPrimary,
    dmMainForm,
    dmActiveForm
  };
```

File

Forms

Description

Use TDefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

7.4.7 Forms.TFormBorderStyle Enumeration

TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 2597) is the type of the BorderStyle property for other controls in windows-only applications.

Pascal

```
TFormBorderStyle = (
  bsNone,
  bsSingle,
  bsSizeable,
  bsDialog,
  bsToolWindow,
  bsSizeToolWin
);
```

C++

```
enum TFormBorderStyle {
  bsNone,
  bsSingle,
  bsSizeable,
  bsDialog,
  bsToolWindow,
  bsSizeToolWin
};
```

File

Forms

Description

These are the possible values of the TFormBorderStyle type:

Value	Meaning
bsNone	No visible border. (not resizable)
bsSingle	Single-line border. (not resizable)
bsDialog	Standard dialog box border. (not resizable)
bsSizeable	Standard resizable border.
bsToolWindow	Like bsSingle but with a smaller caption.

bsSizeToolWin	Like bsSizeable but with a smaller caption.

Only the bsNone and bsSingle values are possible values of TBorderStyle (see page 2597).

7.4.8 Forms.TFormStyle Enumeration

TFormStyle describes form style.

Pascal

```
TFormStyle = (
  fsNormal,
  fsMDIChild,
  fsMDIForm,
  fsStayOnTop
);
```

C++

```
enum TFormStyle {
  fsNormal,
  fsMDIChild,
  fsMDIForm,
  fsStayOnTop
};
```

File

Forms

Description

TFormStyle consists of the following values.

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

See Also

TCustomForm (see page 2103)

7.4.9 Forms.TMonitorDefaultTo Enumeration

TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.

Pascal

```
TMonitorDefaultTo = (
  mdNearest,
  mdNull,
```

```
    mdPrimary
  );
```

C++

```
  enum TMonitorDefaultTo {
    mdNearest,
    mdNull,
    mdPrimary
  };
```

File

Forms

Description

TMonitorDefaultTo indicates which monitor should be used when the selection criteria do not match any of the monitors in the system. It can have any of the following values:

Value	Meaning
mdNearest	Choose the monitor that comes closest to matching the criteria (the monitor closest to a point, on which most of a rectangle lies, etc).
mdNull	Do not select any monitor if the criteria can't be met. Use nil (Delphi) or NULL (C++) to indicate that no monitor could be found.
mdPrimary	Choose the primary monitor if no monitor matches the criteria.

7.4.10 Forms.TPopupForm Record

Pascal

```
  TPopupForm = record
    FormID: Integer;
    Form: TCustomForm;
    WasPopup: Boolean;
  end;
```

C++

```
  struct TPopupForm {
    int FormID;
    TCustomForm Form;
    Boolean WasPopup;
  };
```

File

Forms

Description

This is record Forms.TPopupForm.

7.4.11 Forms.TPopupMode Enumeration

TPopupMode enumerates the possible values of a Form's PopupMode property.

Pascal

```
TPopupMode = (
  pmNone,
  pmAuto,
  pmExplicit
);
```

C++

```
enum TPopupMode {
  pmNone,
  pmAuto,
  pmExplicit
};
```

File

Forms

Description

The PopupMode property controls how the top-level form behaves with respect to Window's WS_POPUP style. The TPopupMode can have one of these values :

Value	Meaning
pmNone	Set the PopupMode property to pmNone if you want the popup window to have the pre-Delphi 8 behavior.
pmAuto	Popup handling is automatic.
pmExplicit	Set the PopupMode property to pmExplicit for non-modal design windows such as tool palettes and other floating tool windows. This causes the windows to always remain on top of the main form.

See Also

PopupMode (see page 2299)

7.4.12 Forms.TPopupWnd Record

Pascal

```
TPopupWnd = record
  ID: Integer;
  ControlWnd: HWND;
end;
```

C++

```
struct TPopupWnd {
  int ID;
  HWND ControlWnd;
};
```

File

Forms

Description

This is record Forms.TPopupWnd.

7.4.13 Forms.TPosition Enumeration

TPosition describes the positioning of a form.

Pascal

```
TPosition = (
  poDesigned,
  poDefault,
  poDefaultPosOnly,
  poDefaultSizeOnly,
  poScreenCenter,
  poDesktopCenter,
  poMainFormCenter,
  poOwnerFormCenter
);
```

C++

```
enum TPosition {
  poDesigned,
  poDefault,
  poDefaultPosOnly,
  poDefaultSizeOnly,
  poScreenCenter,
  poDesktopCenter,
  poMainFormCenter,
  poOwnerFormCenter
};
```

File

Forms

Description

TPosition consists of the following values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.
poDefault	The form appears in a position on the screen and with a height and width determined by the operating system.
poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. Note that in multi-monitor applications, this may result in a form that does not fall entirely on one monitor, because CLX does not include multi-monitor support.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

TCustomForm (see page 2103)

7.4.14 Forms.TPrintScale Enumeration

Represents the proportions of a printed form.

Pascal

```
TPrintScale = (
  poNone,
  poProportional,
  poPrintToFit
);
```

C++

```
enum TPrintScale {
  poNone,
  poProportional,
  poPrintToFit
};
```

File

Forms

Description

Use TPrintScale to get or set the proportions of the printed form. TPrintScale can have one of the following values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

7.4.15 Forms.TScrollBarKind Enumeration

TScrollBarKind indicates the orientation of a scrollbar.

Pascal

```
TScrollBarKind = (
  sbHorizontal,
  sbVertical
);
```

C++

```
enum TScrollBarKind {
  sbHorizontal,
  sbVertical
};
```

File

Forms

Description

TScrollBarKind indicates whether a scrollbar is horizontal or vertical. It includes the following values:

Value	Meaning
sbHorizontal	Horizontal scrollbar.
sbVertical	Vertical scrollbar.

7.4.16 Forms.TScrollBarStyle Enumeration

TScrollBarStyle specifies a scroll bar style.

Pascal

```
TScrollBarStyle = (
  ssRegular,
  ssFlat,
  ssHotTrack
);
```

C++

```
enum TScrollBarStyle {
  ssRegular,
  ssFlat,
  ssHotTrack
};
```

File

Forms

Description

TScrollBarStyle can have one of the following values:

Value	Meaning
ssRegular	The scroll bar is normal.
ssFlat	The scroll bar is flat.
ssHotTrack	The scroll bar uses hot tracking.

7.4.17 Forms.TShowAction Enumeration

Pascal

```
TShowAction = (
  saIgnore,
  saRestore,
  saMinimize,
  saMaximize
);
```

C++

```
enum TShowAction {
  saIgnore,
  saRestore,
  saMinimize,
  saMaximize
};
```

File

Forms

Description

This is record Forms.TShowAction.

7.4.18 Forms.TTileMode Enumeration

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
TTileMode = (
  tbHorizontal,
  tbVertical
);
```

C++

```
enum TTileMode {
  tbHorizontal,
  tbVertical
};
```

File

Forms

Description

TTileMode can have one of the following values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form.
tbVertical	Each form stretches along the height of the parent form.

Note: TTileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle property is set to fsMDIChild).

7.4.19 Forms.TTimerMode Enumeration

Pascal

```
TTimerMode = (
  tmShow,
  tmHide
);
```

C++

```
enum TTimerMode {
  tmShow,
  tmHide
};
```

File

Forms

Description

This is record Forms.TTimerMode.

7.4.20 Forms.TWindowState Enumeration

TWindowState indicates whether a form is maximized, minimized, or normally sized.

Pascal

```
TWindowState = (
  wsNormal,
  wsMinimized,
  wsMaximized
);
```

C++

```
enum TWindowState {
  wsNormal,
  wsMinimized,
  wsMaximized
};
```

File

Forms

Description

TWindowState describes the state of a form window. The following table lists the possible values:

Value	Meaning
wsNormal	The form is in its normal state (that is, neither minimized nor maximized).
wsMinimized	The form is minimized.
wsMaximized	The form is maximized.

See Also

TCustomForm (see page 2103)

7.5 Types

The following table lists types in this documentation.

Types

Name	Description
PCursorRec (see page 2595)	This is type Forms.PCursorRec.
PHintInfo (see page 2596)	PHintInfo declares a pointer to a Help Window appearance record.
TBorderIcons (see page 2596)	TBorderIcons specifies the buttons that appear on the title bar of a form window.
TBorderStyle (see page 2597)	TBorderStyle is the type of the BorderStyle property for framed controls.
TCloseEvent (see page 2597)	TCloseEvent is the type of a form's OnClose event handle.
TCloseQueryEvent (see page 2598)	TCloseQueryEvent is used for event handlers that are called when a window is about to close.
TCursorRecType (see page 2598)	This is type Forms.TCursorRecType.
TCustomFormClass (see page 2599)	TCustomFormClass defines the metaclass for TCustomForm (see page 2103).
TCustomFrameClass (see page 2599)	TCustomFrameClass defines the metaclass for TCustomFrame (see page 2237).
TExceptionEvent (see page 2599)	TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.
TFocusState (see page 2600)	This is type Forms.TFocusState.
TFormClass (see page 2600)	TFormClass defines the metaclass for TForm (see page 2241).
TFormState (see page 2601)	TFormState describes the possible state of a form.
TGetHandleEvent (see page 2601)	This is type Forms.TGetHandleEvent.
THelpEvent (see page 2602)	THelpEvent is the type of an OnHelp event handler.
THintInfo (see page 2602)	THintInfo is used to define the appearance and behavior of the Help window.
TIdleEvent (see page 2604)	TIdleEvent is the type of event handler that perform actions when the application is idle.
TMessageEvent (see page 2605)	TMessageEvent is the type of an OnMessage event handler.
TPopupFormArray (see page 2605)	This is type Forms.TPopupFormArray.
TPopupWndArray (see page 2605)	This is type Forms.TPopupWndArray.
TScrollBarInc (see page 2606)	TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 2009).
TSetLayeredWindowAttributes (see page 2606)	This is type Forms.TSetLayeredWindowAttributes.
TSettingChangeEvent (see page 2607)	TSettingChangeEvent is the type of an OnSettingChange event handler.
TShortCutEvent (see page 2609)	TShortCutEvent is the type of an OnShortCut event handler.
TShowHintEvent (see page 2610)	TShowHintEvent is the type of the OnShowHint event handler.
TTaskWindowList (see page 2610)	This is type Forms.TTaskWindowList.
TWindowHook (see page 2611)	TWindowHook is the type for a dialog procedure.

7.5.1 Forms.PCursorRec Type

Pascal

```
PCursorRec = ^TCursorRec;
```

C++

```
^TCursorRec PCursorRec;
```

File

Forms

Description

This is type Forms.PCursorRec.

7.5.2 Forms.PHintInfo Type

PHintInfo declares a pointer to a Help Window appearance record.

Pascal

```
PHintInfo = Controls.PHintInfo deprecated 'Use Controls.PHintInfo';
```

C++

```
Controls.PHintInfo deprecated 'Use Controls.PHintInfo' PHintInfo;
```

File

Forms

Description

The PHintInfo type declares a pointer to a THintInfo (see page 2602) variable. The THintInfo (see page 2602) type is used to define the appearance and behavior of the Help window in a TShowHintEvent (see page 2610) type OnShowHint event handler.

See Also

OnShowHint (see page 2008)

TShowHintEvent (see page 2610)

THintInfo (see page 2602)

7.5.3 Forms.TBorderIcons Type

TBorderIcons specifies the buttons that appear on the title bar of a form window.

Pascal

```
TBorderIcons = set of TBorderIcon;
```

C++

```
TBorderIcon TBorderIcons;
```

File

Forms

Description

TBorderIcons type declares a pointer to a TBorderIcon (see page 2583) variable. TBorderIcons can include any of the following TBorderIcon (see page 2583) values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).
biMinimize	The form has a Minimize button

biMinimize	The form has a Maximize button
biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise, no question mark appears.

See Also

TBorderIcon (see page 2583)

7.5.4 Forms.TBorderStyle Type

TBorderStyle is the type of the BorderStyle property for framed controls.

Pascal

```
TBorderStyle = bsNone..bsSingle;
```

C++

```
bsNone..bsSingle TBorderStyle;
```

File

Forms

Description

These are the possible values of the TBorderStyle type:

Value	Meaning
bsNone	No visible border.
bsSingle	Single-line border.

7.5.5 Forms.TCloseEvent Type

TCloseEvent is the type of a form's OnClose event handle.

Pascal

```
TCloseEvent = procedure (Sender: TObject; var Action: TCloseAction) of object;
```

C++

```
(Sender: TObject; var Action: TCloseAction) ( TCloseEvent)();
```

File

Forms

Description

The TCloseEvent type points to a method that handles the closing of a form. The **Sender** parameter is the form that the user tried to close. The **Action** parameter returns the action you want the form to take when it closes.

Setting **Action** controls how the form responds to the attempt to close it as shown below :

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

See Also

OnClose (see page 2321)

7.5.6 Forms.TCloseQueryEvent Type

TCloseQueryEvent is used for event handlers that are called when a window is about to close.

Pascal

```
TCloseQueryEvent = procedure (Sender: TObject; var CanClose: Boolean) of object;
```

C++

```
(Sender: TObject; var CanClose: Boolean) ( TCloseQueryEvent)();
```

File

Forms

Description

The TCloseQueryEvent type is the type for event handlers that allow an application to block the closing of a window (form or dialog).

Sender is the object whose event handler is called. This is the form that is about to close or the dialog whose window is about to close.

CanClose is set by the event handler to indicate whether the form or dialog can close (true) or whether the user should be blocked from closing it (false).

See Also

OnClose (see page 2321)

TCloseQueryEvent

7.5.7 Forms.TCursorRecType Type

Pascal

```
TCursorRecType = PCursorRec;
```

C++

```
PCursorRec TCursorRecType;
```

File

Forms

Description

This is type Forms.TCursorRecType.

7.5.8 Forms.TCustomFormClass Type

TCustomFormClass defines the metaclass for TCustomForm (see page 2103).

Pascal

```
TCustomFormClass = class of TCustomForm;
```

C++

```
TCustomForm TCustomFormClass;
```

File

Forms

Description

TCustomFormClass is the metaclass for TCustomForm (see page 2103). Its value is the class reference for TCustomForm (see page 2103) or for one of its descendants.

See Also

TCustomForm (see page 2103)

7.5.9 Forms.TCustomFrameClass Type

TCustomFrameClass defines the metaclass for TCustomFrame (see page 2237).

Pascal

```
TCustomFrameClass = class of TCustomFrame;
```

C++

```
TCustomFrame TCustomFrameClass;
```

File

Forms

Description

TCustomFrameClass is the metaclass for TCustomFrame (see page 2237). Its value is the class reference for TCustomFrame (see page 2237) or for one of its descendants.

See Also

TCustomFrame (see page 2237)

7.5.10 Forms.TExceptionEvent Type

TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.

Pascal

```
TExceptionEvent = procedure (Sender: TObject; E: Exception) of object;
```

C++

```
(Sender: TObject; E: Exception) ( TExceptionEvent)();
```

File

Forms

Description

The TExceptionEvent type is the type of the OnException event handler.

Sender is the object that receives the event notification. Typically, this is the application object.

E is the exception object that represents the exception.

See Also

OnException (see page 1993)

7.5.11 Forms.TFocusState Type

Pascal

```
TFocusState = type Pointer;
```

C++

```
type Pointer TFocusState;
```

File

Forms

Description

This is type Forms.TFocusState.

7.5.12 Forms.TFormClass Type

TFormClass defines the metaclass for TForm (see page 2241).

Pascal

```
TFormClass = class of TForm;
```

C++

```
TForm * TFormClass;
```

File

Forms

Description

TFormClass is the metaclass for TForm (see page 2241). Its value is the class reference for TForm (see page 2241) or for one of its descendants.

See Also

TForm (see page 2241)

7.5.13 Forms.TFormState Type

TFormState describes the possible state of a form.

Pascal

```
TFormState = set of (fsCreating, fsVisible, fsShowing, fsModal, fsCreatedMDIChild,
fsActivated);
```

C++

```
(fsCreating, fsVisible, fsShowing, fsModal, fsCreatedMDIChild, fsActivated) TFormState;
```

File

Forms

Description

The following table lists the values that can be included in a form's state:

Value	Meaning
fsCreating	The form's constructor is currently executing.
fsVisible	The form's window is visible. This state is used to update the Visible property.
fsShowing	The form's WindowState property is changing. This state is used to prevent WindowState changes from interfering with a transition that is in progress.
fsModal	The form was created as a modal window.
fsActivated	The form has received focus or the application became active but has not yet called the Activate method to generate an OnActivate event.

7.5.14 Forms.TGetHandleEvent Type

Pascal

```
TGetHandleEvent = procedure (var Handle: HWND) of object;
```

C++

```
(var Handle: HWND) ( TGetHandleEvent)();
```

File

Forms

Description

This is type Forms.TGetHandleEvent.

7.5.15 Forms.THelpEvent Type

THelpEvent is the type of an OnHelp event handler.

Pascal

```
THelpEvent = function (Command: Word; Data: Longint; var CallHelp: Boolean): Boolean of object;
```

C++

```
(Command: Word; Data: Longint; var CallHelp: Boolean): Boolean ( THelpEvent)();
```

File

Forms

Description

THelpEvent includes the following parameters:

Command indicates the type of help command that was issued. It's meaning depends on the current help system. For example, when using WinHelp, Command is the command used by the WinHelp API. For details on possible commands, see the documentation provided by the developer of the application's help system. For example, if the application uses WinHelp, the possible commands are listed in the WinHelp section of the Win32 Developer's Reference.

Data provides additional data to describe the desired action by the help system. Its value depends on the value of the Command property. Typically, Data specifies a keyword or context ID. For details, see the documentation provided by the application's help system.

CallHelp is set by the event handler. If CallHelp is true when the event handler exits, the application should forward the help command to the help system. If the event handler handles the command, it sets CallHelp to false so that the application does not forward the help command to the help system.

7.5.16 Forms.THintInfo Type

THintInfo is used to define the appearance and behavior of the Help window.

Pascal

```
THintInfo = Controls.THintInfo deprecated 'Use Controls.THintInfo';
```

C++

```
Controls.THintInfo deprecated 'Use Controls.THintInfo' THintInfo;
```

File

Forms

Description

The THintInfo type is used to define the appearance and behavior of the Help window in a TShowHintEvent (see page 2610) type OnShowHint event handler. The following table describes the meaning of its fields:

Field	Meaning
HintControl	The name of the control for which hint processing is occurring.

HintWindowClass (see page 2615)	The class of the hint-window control. The default is THintWindow, but you can specify any class derived from THintWindow. Use this field if you want to substitute a custom hint window for THintWindow.
HintPos	The default position in screen coordinates of the top-left corner of the hint window. Change where the window appears by changing this value.
HintMaxWidth	The maximum width of the hint window before word wrapping begins. By default, the value is the width of the screen (the Width property of the global Screen (see page 2615) variable).
HintColor	The background color of the Hint window.
CursorRect	The rectangle the user's mouse pointer must be in for the hint window to appear. The default value for CursorRect is the client rectangle of the control. Change this value so that a single control can be divided into several hint regions. When the user moves the mouse pointer outside the rectangle, the hint window disappears.
CursorPos	The location of the mouse pointer within the control.
ReshowTimeout	How long the hint system should wait before asking about the hint status again. By default, this field is zero, indicating that the hint status will not change. Setting it to a non-zero value will cause the hint to act, after the requested milliseconds have elapsed, as if the user moved the mouse outside the hint rectangle and back in. This can be used to either put off hint processing for a period of time, or to allow the hint to be updated periodically.
HideTimeout	The number of milliseconds to show the hint. By default, it is set to the value of the Application (see page 2612) variable's HintHidePause property.
HintStr	The string to be displayed in the hint window. This allows an OnHint event handler to modify the contents of a hint before it is displayed. By default, it contains the value returned by the GetShortHint function when passed the value of the Application (see page 2612) variable's Hint property.
HintData	Additional data to be passed to the hint-window control. Use this field in conjunction with HintWindowClass (see page 2615).

C++ Examples:

```
/*
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
*/
void __fastcall TForm1::DoShowHint(UnicodeString &HintStr, bool &CanShow, THintInfo &HintInfo)
{
  if (HintInfo.HintControl == SpeedButton3)
  {
    HintInfo.HintColor = clAqua;// Changes only for this hint
    HintInfo.HintMaxWidth = 120; // Hint text word wraps if width is greater than 120
    HintInfo.HintPos.x += SpeedButton3->Width; // Move hint to right edge
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->ShowHint = true;
  Application->OnShowHint = DoShowHint;
}
```

Delphi Examples:

```
{
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
}

procedure TForm1.DoShowHint(var HintStr: String; var CanShow: Boolean; var HintInfo:
THintInfo);
begin
  if HintInfo.HintControl = SpeedButton3 then
  begin
    with HintInfo do
    begin
      HintColor := clAqua;{ Changes only for this hint }
      HintMaxWidth := 120;{Hint text word wraps if width is greater than 120 }
      Inc(HintPos.X, SpeedButton3.Width); { Move hint to right edge }
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.ShowHint := True;
  Application.OnShowHint := DoShowHint;
end;
```

See Also

THintWindow

OnShowHint

7.5.17 Forms.TIdleEvent Type

TIdleEvent is the type of event handler that perform actions when the application is idle.

Pascal

```
TIdleEvent = procedure (Sender: TObject; var Done: Boolean) of object;
```

C++

```
(Sender: TObject; var Done: Boolean) ( TIdleEvent)();
```

File

Forms

Description

The TIdleEvent type is the type of the OnIdle event.

Sender is the object that receives the event notification. Typically, this is the application object.

Done indicates whether the event handler has finished its background processing.

If the event handler exits when **Done** is true, all action components are then given a chance to update themselves and, on Windows, the application yields control to Windows until a new message appears in the application's message queue. When

Done is false, any action components in the application are not allowed to update themselves when the event handler exits and, on Windows, the application does not explicitly yield control to the operating system.

See Also

OnIdle (see page 1998)

7.5.18 Forms.TMessageEvent Type

TMessageEvent is the type of an OnMessage event handler.

Pascal

```
TMessageEvent = procedure (var Msg: TMsg; var Handled: Boolean) of object;
```

C++

```
(var Msg: TMsg; var Handled: Boolean) ( TMessageEvent)();
```

File

Forms

Description

TMessageEvent includes the following parameters:

Msg identifies the Windows message that triggered the event.

Handled indicates whether the event handler responded to the message. If the event handler sets Handled to true, the application assumes that the message has been completely handled and halts any subsequent processing of the message.

7.5.19 Forms.TPopupFormArray Type

Pascal

```
TPopupFormArray = array of TPopupForm;
```

C++

```
array of TPopupForm TPopupFormArray;
```

File

Forms

Description

This is type Forms.TPopupFormArray.

7.5.20 Forms.TPopupWndArray Type

Pascal

```
TPopupWndArray = array of TPopupWnd;
```

C++

```
array of TPopupWnd TPopupWndArray;
```

File

Forms

Description

This is type Forms.TPopupWndArray.

7.5.21 Forms.TScrollBarInc Type

TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 2009).

Pascal

```
TScrollBarInc = 1..32767;
```

C++

```
1..32767 TScrollBarInc;
```

File

Forms

Description

TScrollBarInc has a value between 1 and 32767 (inclusive). It indicates the number of positions to scroll.

See Also

TControlScrollBar

7.5.22 Forms.TSetLayeredWindowAttributes Type

Pascal

```
[SuppressUnmanagedCodeSecurity]
TSetLayeredWindowAttributes = function (Hwnd: THandle; crKey: COLORREF; bAlpha: Byte; dwFlags:
DWORD): Boolean;
```

C++

```
[SuppressUnmanagedCodeSecurity]
(Hwnd: THandle; crKey: COLORREF; bAlpha: Byte; dwFlags: DWORD): Boolean (
TSetLayeredWindowAttributes)();
```

File

Forms

Description

This is type Forms.TSetLayeredWindowAttributes.

7.5.23 Forms.TSettingChangeEvent Type

TSettingChangeEvent is the type of an OnSettingChange event handler.

Pascal

```
TSettingChangeEvent = procedure (Sender: TObject; Flag: Integer; const Section: string; var
Result: Longint) of object;
```

C++

```
(Sender: TObject; Flag: Integer; const Section: string; var Result: Longint) (
TSettingChangeEvent)();
```

File

Forms

Description

TSettingChangeEvent has the following parameters:

Sender is the application object.

Section indicates the area that contains the system parameter that changed. Typically, this is a registry key or the name of a section in Win.ini. If the notification indicates a policy change, Section is the string 'Policy'. If the notification indicates a change in locale settings, Section is the string 'intl'.

Result is the value returned to Windows. It should be 0 if the event handler responded to the setting change, nonzero otherwise.

Flag indicates what setting changed. The following table lists the possible values:

Value	Meaning
0	A user policy was applied or a user locale setting changed.
1	A computer policy was applied.
SPI_SETDESKWALLPAPER	The bitmap for the desktop wallpaper changed.
SPI_SETCURSORS	The system cursors changed.
SPI_SETDESKPATTERN	The desktop pattern was reloaded from Win.ini.
SPI_SETFONTSMOOTING	The font smoothing feature was enabled or disabled.
SPI_SETWORKAREA	The size of the work area (the portion of the screen not obscured by the system taskbar or desktop toolbars) changed.
SPI_ICONHORIZONTALSPACING	The width, in pixels, of an icon cell changed. This affects the arrangement of icons in large icon view.
SPI_ICONVERTICALSPACING	The height, in pixels, of an icon cell changed. This affects the arrangement of icons in large icon view.
SPI_SETICONMETRICS	The metrics associated with system icons changed.
SPI_SETICONS	The system icons were reloaded.
SPI_SETICONTITLELOGFONT	The font for icon titles changed.
SPI_SETICONTITLEWRAP	Icon title-wrapping was turned on or off.
SPI_SETBEEP	The warning beeper was turned on or off.
SPI_SETDEFAULTINPUTLANG	The default input language for the system shell changed
SPI_SETDOUBLECLICKTIME	The double-click time for the mouse changed.

SPI_SETDOUBLECLICKHEIGHT	The height of the double-click rectangle changed.
SPI_SETDOUBLECLICKWIDTH	The width of the double-click rectangle changed.
SPI_SETKEYBOARDCUES	The setting for whether Menu access key letters appear always or only when the menu is activated from the keyboard changed.
SPI_SETKEYBOARDDELAY	The keyboard repeat-delay setting changed.
SPI_SETKEYBOARDPREF	The keyboard preference changed. This indicates whether the user wants applications to display hidden keyboard interfaces.
SPI_SETKEYBOARDSPEED	The keyboard repeat-speed setting changed.
SPI_SETLANGTOGGLE	The hotkey for switching input languages changed.
SPI_SETMOUSE	The mouse threshold values or acceleration changed.
SPI_SETMOUSEBUTTONSWRAP	The meaning of left and right mouse buttons changed.
SPI_SETMOUSEHOVERHEIGHT	The height of the rectangle in which the mouse must stay for mouse hover messages changed.
SPI_SETMOUSEHOVERTIME	The time the mouse must stay in the hover rectangle to generate mouse hover messages changed.
SPI_SETMOUSEHOVERWIDTH	The width of the rectangle in which the mouse must stay for mouse hover messages changed.
SPI_SETMOUSESPEED	The mouse speed changed.
SPI_SETMOUSETRAILS	Mouse trails were turned on or off.
SPI_SETSNAPTODEFBUTTON	The automatic snapping of the mouse to the default button was turned on or off.
SPI_SETWHEELSCROLLLINES	The number of lines to scroll when the mouse wheel rotates changed.
SPI_SETMENUDROPALIGNMENT	Popup menus changed between left- and right-alignment.
SPI_SETMENUFADE	Menu fade animation was turned on or off (Windows 2K)
SPI_SETMENUSHOWDELAY	The delay before displaying a shortcut menu when the mouse is over a submenu item changed.
SPI_SETLOWPOWERACTIVE	The low-power phase of screen saving was activated or deactivated.
SPI_SETLOWPOWERTIMEOUT	The time period before the low-power phase of screen saving is activated changed.
SPI_SETPOWEROFFACTIVE	The power-off phase of screen saving came on or off.
SPI_SETPOWEROFFTIMEOUT	The delay before activating the power-off phase of screen saving changed.
SPI_SETSCREENSAVEACTIVE	The screen saver was activated or deactivated.
SPI_SETSCREENSAVETIMEOUT	The delay before the screen saver appears changed.
SPI_SETCOMBOBOXANIMATION	The slide-open effect for combo boxes was turned on or off.
SPI_SETCURSORSHADOW	The cursor shadow was turned on or off (Windows 2K).
SPI_SETGRADIENTCAPTIONS	The gradient effect on window title bars was turned on or off.
SPI_SETHOTTRACKING	Hot-tracking was enabled or disabled.
SPI_SETLISTBOXSMOOTHSCROLLING	Smooth scrolling of list boxes was enabled or disabled.
SPI_SETMENUANIMATION	Menu animation was enabled or disabled.
SPI_SETMENUUNDERLINES	Same as SPI_SETKEYBOARDCUES.
SPI_SETSELECTIONFADE	The menu selection fade feature was enabled or disabled. (Windows 2K)
SPI_SETTOOLTIPANIMATION	ToolTip animation was enabled or disabled. (Windows 2K)
SPI_SETTOOLTIPFADE	ToolTip animation changed between a fade and a slide effect. (Windows 2K)

SPI_SETUIEFFECTS	UI effects were enabled or disabled. (Windows 2K)
SPI_SETACTIVEWINDOWTRACKING	Active window tracking (where the active window follows the mouse) was enabled or disabled.
SPI_SETACTIVEWNDTRKZORDER	Active window tracking changed whether the active window was brought to the top.
SPI_SETANIMATION	Parameters to the animation effects for user actions changed.
SPI_SETBORDER	The width of a window's sizing border changed.
SPI_SETCARETWIDTH	The width of the caret in edit controls changed (Windows 2K)
SPI_SETDRAGFULLWINDOWS	Dragging of full windows was turned on or off.
SPI_SETDRAGHEIGHT	The height of the rectangle used to detect drag operations changed.
SPI_SETDRAGWIDTH	The width of the rectangle used to detect drag operations changed.
SPI_SETFOREGROUNDFLASHCOUNT	The number of times a taskbar button flashes when rejecting a foreground switch request changed.
SPI_SETFOREGROUNDLOCKTIMEOUT	The time after user input during which applications can't force themselves into the foreground changed.
SPI_SETMINIMIZEDMETRICS	The metrics for minimized windows changed.
SPI_SETNONCLIENTMETRICS	The metrics for the nonclient area of nonminimized windows changed.
SPI_SETSHOWIMEUI	The ability of the IME window to be visible on a per-user basis changed.
SPI_SETPENWINDOWS	Pen windows was loaded or unloaded. (Windows 9x)
SPI_SETACCESSTIMEOUT	The time-out period for accessibility features changed.
SPI_SETFILTERKEYS	The parameters to the FilterKeys accessibility feature changed.
SPI_SETHIGHCONTRAST	The parameters to the HighContrast accessibility feature changed.
SPI_SETMOUSECLICKLOCK	The mouse ClickLock state was turned on or off. (Windows Me)
SPI_SETMOUSECLICKLOCKTIME	The time delay for locking the primary mouse button changed. (Windows Me)
SPI_SETMOUSEKEYS	The parameters for the MouseKeys accessibility feature changed.
SPI_SETMOUSESONAR	The mouse Sonar feature was turned on or off. (Windows Me)
SPI_SETMOUSEVANISH	The mouse Vanish feature was turned on or off. (Windows Me)
SPI_SETSCREENREADER	A screen reviewer utility (such as a speech synthesizer or Braille display utility) started or stopped running.
SPI_SETSERIALKEYS	The parameters to the SerialKeys accessibility feature changed. (Windows 9x)
SPI_SETSHOWSOUNDS	The Show Sounds accessibility feature toggled.
SPI_SETSOUNDSENTRY	The parameters to the SoundSentry accessibility feature changed.
SPI_SETSTICKYKEYS	The parameters to the StickyKeys feature changed.
SPI_SETTOGGLEKEYS	The parameters to the ToggleKeys feature changed.

7.5.24 Forms.TShortCutEvent Type

TShortCutEvent is the type of an OnShortCut event handler.

Pascal

```
TShortCutEvent = procedure (var Msg: TWMKey; var Handled: Boolean) of object;
```

C++

```
(var Msg: TWMKey; var Handled: Boolean) ( TShortCutEvent)();
```

File

Forms

Description

The TShortCutEvent type represents event handlers that respond to keystrokes before they enter the standard keystroke processing. These event handlers allow an application to bypass the standard keystroke processing for special shortcut keys.

The Msg parameter represents the Windows keystroke message. The virtual key code is given by the CharCode field, and the KeyData field contains a set of flags that indicate the repeat count, scan code, extended-key flag, context code, previous key-state flag and transition-state flag. See the Windows documentation for details on these fields.

The Handled parameter indicates whether the event handler processed the key. If Handled is set to true, then the application stops processing the keystroke. Otherwise (if Handled remains false) the application proceeds with normal keystroke processing.

7.5.25 Forms.TShowHintEvent Type

TShowHintEvent is the type of the OnShowHint event handler.

Pascal

```
TShowHintEvent = procedure (var HintStr: string; var CanShow: Boolean; var HintInfo:
THintInfo) of object;
```

C++

```
(var HintStr: string; var CanShow: Boolean; var HintInfo: THintInfo) ( TShowHintEvent)();
```

File

Forms

Description

TShowHintEvent has the following parameters:

HintStr sets the text of the Help Hint. You can change this string in the event handler to change the hint text that appears to the user.

CanShow indicates whether the Help Hint should display. If the event handler sets CanShow to true, the Help Hint displays. If it sets CanShow to false, the Help Hint does not appear.

HintInfo contains information about the appearance and behavior of the Help window. Change its fields to customize the way the Help Hint is displayed.

7.5.26 Forms.TTaskWindowList Type

Pascal

```
TTaskWindowList = Pointer;
```

C++

```
void * TTaskWindowList;
```

File

Forms

Description

This is type Forms.TTaskWindowList.

7.5.27 Forms.TWindowHook Type

TWindowHook is the type for a dialog procedure.

Pascal

```
TWindowHook = function (var Message: TMessage): Boolean of object;
```

C++

```
(var Message: TMessage): Boolean ( TWindowHook)();
```

File

Forms

Description

TWindowHook is the type for the window procedure of a non-VCL dialog box. Like most window procedures, TWindowHook takes a TMessage (see page 4997) value as an argument.

Except in response to the WM_INITDIALOG (see page 5174) message, the window hook should return true if it processes the message, and false if it does not. When the Msg field of the Message parameter is WM_INITDIALOG (see page 5174), the window hook should return false if it explicitly sets the focus to one of the controls in the dialog box. Otherwise, it should return true, in which case the system sets the focus to the first control in the dialog box that can be given the focus.

See Also

TApplication (see page 1924)

7.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 2612)	Represents application-level information.
Ctl3DBtnWndProc (see page 2614)	This is variable Forms.Ctl3DBtnWndProc.
Ctl3DCtlColorEx (see page 2614)	This is variable Forms.Ctl3DCtlColorEx.
Ctl3DDlgFramePaint (see page 2614)	This is variable Forms.Ctl3DDlgFramePaint.
HintWindowClass (see page 2615)	Specifies the class for the window used to display help hints.
Screen (see page 2615)	Represents a screen device.
SetLayeredWindowAttributes (see page 2617)	This is variable Forms.SetLayeredWindowAttributes.

7.6.1 Forms.Application Variable

Represents application-level information.

Pascal

```
Application: TApplication;
```

C++

```
TApplication Application;
```

File

Forms

Description

By default, when a new project is created, the IDE constructs an application object and assigns it to the Application variable in the Forms (see page 1921) unit. Application has several properties that can be used to get information about an application while it runs.

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
```

```
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
/*
This code displays the name of your project in an edit box:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->Text = Application->Title;
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  temp1 := Edit4.Text;
  temp2 := ExtractFileName(Edit1.Text);
  NewFileName := temp1 + temp2;
  OldFileName :=
    Edit3.Text + ExtractFileName(Edit2.Text);
  Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
  if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
  begin
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
```

```
  end;
end;
{
This code displays the name of your project in an edit box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := Application.Title;
end;
```

See Also

TApplication (see page 1924)

7.6.2 Forms.Ctl3DBtnWndProc Variable

Pascal

```
Ctl3DBtnWndProc: Pointer = nil;
```

C++

```
void * Ctl3DBtnWndProc = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DBtnWndProc.

7.6.3 Forms.Ctl3DCtlColorEx Variable

Pascal

```
Ctl3DCtlColorEx: function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall = nil;
```

C++

```
function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall Ctl3DCtlColorEx = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DCtlColorEx.

7.6.4 Forms.Ctl3DDlgFramePaint Variable

Pascal

```
Ctl3DDlgFramePaint: function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall =
nil;
```

C++

```
function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall Ctl3DDlgFramePaint = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DDlgFramePaint.

7.6.5 Forms.HintWindowClass Variable

Specifies the class for the window used to display help hints.

Pascal

```
HintWindowClass: THintWindowClass = THintWindow;
```

C++

```
THintWindowClass HintWindowClass = THintWindow;
```

File

Forms

Description

When the application displays a Help Hint, it creates an instance of HintWindowClass to represent the window used for displaying the hint. Applications can customize this window by creating a descendant of THintWindow and assigning it to the HintWindowClass variable at application startup.

See Also

THintWindow

TApplication (see page 1924)

7.6.6 Forms.Screen Variable

Represents a screen device.

Pascal

```
Screen: TScreen;
```

C++

```
TScreen Screen;
```

File

Forms

Description

The Screen variable is a TScreen (see page 2445) component that represents the screen of the system on which the application runs. By default, applications create a screen component based on information about the current screen device and assign it to Screen.

C++ Examples:

```
/*
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->FormCount; i++)
  {
    if (Screen->Forms[i]->Top < 0)
    {
      Screen->Forms[i]->Height += Screen->Forms[i]->Top;
      Screen->Forms[i]->Top = 0;
    }
    if (Screen->Forms[i]->Left < 0)
    {
      Screen->Forms[i]->Width += Screen->Forms[i]->Left;
      Screen->Forms[i]->Left = 0;
    }
    if (Screen->Forms[i]->Left + Screen->Forms[i]->Width > Screen->Width)
      Screen->Forms[i]->Width = Screen->Width - Screen->Forms[i]->Left;
    if (Screen->Forms[i]->Top + Screen->Forms[i]->Height > Screen->Height)
      Screen->Forms[i]->Height = Screen->Height - Screen->Forms[i]->Top;
  }
}
```

Delphi Examples:

```
{
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
}
procedure TForm1.Button1Click(Sender: TObject);
var I: Integer;
begin
with Screen do
  for I := 0 to FormCount-1 do
    begin
    if Forms[I].Top < 0 then
    begin
      Forms[I].Height := Forms[I].Height + Forms[I].Top;
      Forms[I].Top := 0;
    end;
    if Forms[I].Left < 0 then
    begin
      Forms[I].Width := Forms[I].Width + Forms[I].Left;
      Forms[I].Left := 0;
    end;
    if Forms[I].Left + Forms[I].Width > Width then
      Forms[I].Width := Width - Forms[I].Left;
    if Forms[I].Top + Forms[I].Height > Height then
      Forms[I].Height := Height - Forms[I].Top;
    end;
end;
```

See Also

Application (see page 2612)

7.6.7 Forms.SetLayeredWindowAttributes Variable

Pascal

```
SetLayeredWindowAttributes: TSetLayeredWindowAttributes = nil;
```

C++

```
TSetLayeredWindowAttributes SetLayeredWindowAttributes = nil;
```

File

Forms

Description

This is variable Forms.SetLayeredWindowAttributes.

8 Graphics Namespace

This is namespace Graphics.

Topics

Name	Description
Classes (see page 2627)	The following table lists classes in this documentation.
Interfaces (see page 2834)	The following table lists interfaces in this documentation.
Functions (see page 2836)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 2860)	The following table lists structs, records, enums in this documentation.
Types (see page 2875)	The following table lists types in this documentation.
Variables (see page 2890)	The following table lists variables in this documentation.
Constants (see page 2892)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidGraphic (see page 2627)	EInvalidGraphic is the exception class for unrecognized graphic files.
	EInvalidGraphicOperation (see page 2628)	EInvalidGraphicOperation is the exception class for illegal operations on graphics.
	TBitmap (see page 2628)	TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).
	TBitmapImage (see page 2662)	TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 2628) object.
	TBrush (see page 2664)	TBrush represents the color and pattern used to fill solid shapes.
	TBrushRecall (see page 2671)	TBrushRecall saves and restores a TBrush (see page 2664) object.
	TCanvas (see page 2673)	TCanvas provides an abstract drawing space for objects that must render their own images.
	TFont (see page 2730)	TFont encapsulates a system font.
	TFontRecall (see page 2747)	TFontRecall saves and restores the properties of a TFont (see page 2730) object.
	TGraphic (see page 2748)	TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.
	TGraphicsObject (see page 2764)	TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 2664), TFont (see page 2730), and TPen (see page 2801).
	TIcon (see page 2768)	TIcon is an encapsulation of a Windows icon.
	TIconImage (see page 2778)	TIconImage is the internal representation of the bitmap image for a TIcon (see page 2768) object.
	TMetafile (see page 2780)	TMetafile is an encapsulation of the Win32 Enhanced (see page 2793) metafile.

	TMetafileCanvas (see page 2794)	TMetafileCanvas allows applications to create metafile images from scratch.
	TMetafileImage (see page 2799)	TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 2780) object.
	TPen (see page 2801)	TPen is used to draw lines or outline shapes on a canvas.
	TPenRecall (see page 2811)	TPenRecall saves and restores the properties of a TPen (see page 2801) object.
	TPicture (see page 2812)	TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.
	TSharedImage (see page 2830)	TSharedImage is the base class for classes that contain images used for graphics objects.

Constants

Name	Description
cl3DDkShadow (see page 2896)	Represents the dark shadow of 3D components.
cl3DLight (see page 2896)	Represents the light part of 3D components.
clActiveBorder (see page 2897)	Represents the default border color of an active window.
clActiveCaption (see page 2897)	Represents the default color of the title bar of an active window.
clAppWorkSpace (see page 2898)	Represents the default Application WorkSpace color.
clAqua (see page 2898)	Represents the color Aqua.
clBackground (see page 2899)	Represents the default Windows background color.
clBlack (see page 2899)	Represents the color black.
clBlue (see page 2899)	Represents the color blue.
clBtnFace (see page 2900)	Represents the color of the face of a button.
clBtnHighlight (see page 2900)	Represents the color of the highlighting on a button.
clBtnShadow (see page 2901)	Represents the color of the shadow cast by of a button.
clBtnText (see page 2901)	Represents the default color of the text on a button.
clCaptionText (see page 2902)	Represents the default color of the text on the title bar of an active window.
clCream (see page 2902)	Represents the color cream.
clDefault (see page 2903)	Specifies the default color of a component.
clDkGray (see page 2903)	Represents the color dark gray.
clFuchsia (see page 2903)	Represents the color fuchsia.
clGradientActiveCaption (see page 2904)	Right side color in the color gradient of an active window's title bar.
clGradientInactiveCaption (see page 2904)	Right side color in the color gradient of an inactive window's title bar.
clGray (see page 2905)	Represents the color gray.
clGrayText (see page 2905)	Represents the color of text that is grayed out.
clGreen (see page 2906)	Represents the color green.
clHighlight (see page 2906)	Represents the color of the background of a block of selected text.
clHighlightText (see page 2907)	Represents the color of selected text.
clHotLight (see page 2907)	This is constant Graphics.clHotLight.
clInactiveBorder (see page 2908)	Represents the default border color of an inactive window.
clInactiveCaption (see page 2908)	Represents the default color of the title bar of an inactive window.
clInactiveCaptionText (see page 2908)	Represents the default color of the title text in an inactive window.
clInfoBk (see page 2909)	Represents the color of the background of a tool tip.
clInfoText (see page 2909)	Represents the color of tool tip text.
clLime (see page 2910)	Represents the color lime.
clLtGray (see page 2910)	Represents the color light gray.
clMaroon (see page 2911)	Represents the color maroon.
clMedGray (see page 2911)	Represents the color medium gray.
clMenu (see page 2912)	Represents the color of the background of an unselected menu sub item.
clMenuBar (see page 2912)	Represents the color of the background of an unselected main menu item.
clMenuHighlight (see page 2912)	Represents the color of the background of a selected menu item.
clMenuText (see page 2913)	Represents the default menu text color.
clMoneyGreen (see page 2913)	Represents the color money green.
clNavy (see page 2914)	Represents the color navy.
clNone (see page 2914)	Represents no selected color.
clOlive (see page 2915)	Represents the color olive.

clPurple (see page 2915)	Represents the color purple.
clRed (see page 2916)	Represents the color red.
clScrollBar (see page 2916)	This is constant Graphics.clScrollBar.
clSilver (see page 2916)	Represents the color silver.
clSkyBlue (see page 2917)	Represents the color sky blue.
clSystemColor (see page 2917)	Is a mask used to identify the set of system colors.
clTeal (see page 2918)	Represents the color teal.
clWebAliceBlue (see page 2918)	This is constant Graphics.clWebAliceBlue.
clWebAntiqueWhite (see page 2919)	This is constant Graphics.clWebAntiqueWhite.
clWebAqua (see page 2919)	This is constant Graphics.clWebAqua.
clWebAquamarine (see page 2919)	This is constant Graphics.clWebAquamarine.
clWebAzure (see page 2920)	This is constant Graphics.clWebAzure.
clWebBeige (see page 2920)	This is constant Graphics.clWebBeige.
clWebBisque (see page 2920)	This is constant Graphics.clWebBisque.
clWebBlack (see page 2921)	This is constant Graphics.clWebBlack.
clWebBlanchedAlmond (see page 2921)	This is constant Graphics.clWebBlanchedAlmond.
clWebBlue (see page 2921)	This is constant Graphics.clWebBlue.
clWebBlueViolet (see page 2922)	This is constant Graphics.clWebBlueViolet.
clWebBrown (see page 2922)	This is constant Graphics.clWebBrown.
clWebBurlywood (see page 2922)	This is constant Graphics.clWebBurlywood.
clWebCadetBlue (see page 2923)	This is constant Graphics.clWebCadetBlue.
clWebChartreuse (see page 2923)	This is constant Graphics.clWebChartreuse.
clWebChocolate (see page 2923)	This is constant Graphics.clWebChocolate.
clWebCoral (see page 2924)	This is constant Graphics.clWebCoral.
clWebCornFlowerBlue (see page 2924)	This is constant Graphics.clWebCornFlowerBlue.
clWebCornSilk (see page 2924)	This is constant Graphics.clWebCornSilk.
clWebCrimson (see page 2925)	This is constant Graphics.clWebCrimson.
clWebCyan (see page 2925)	This is constant Graphics.clWebCyan.
clWebDarkBlue (see page 2925)	This is constant Graphics.clWebDarkBlue.
clWebDarkCyan (see page 2926)	This is constant Graphics.clWebDarkCyan.
clWebDarkGoldenRod (see page 2926)	This is constant Graphics.clWebDarkGoldenRod.
clWebDarkGray (see page 2926)	This is constant Graphics.clWebDarkGray.
clWebDarkgreen (see page 2927)	This is constant Graphics.clWebDarkgreen.
clWebDarkKhaki (see page 2927)	This is constant Graphics.clWebDarkKhaki.
clWebDarkMagenta (see page 2927)	This is constant Graphics.clWebDarkMagenta.
clWebDarkOliveGreen (see page 2928)	This is constant Graphics.clWebDarkOliveGreen.
clWebDarkOrange (see page 2928)	This is constant Graphics.clWebDarkOrange.
clWebDarkOrchid (see page 2928)	This is constant Graphics.clWebDarkOrchid.
clWebDarkRed (see page 2929)	This is constant Graphics.clWebDarkRed.
clWebDarkSalmon (see page 2929)	This is constant Graphics.clWebDarkSalmon.
clWebDarkSeaGreen (see page 2929)	This is constant Graphics.clWebDarkSeaGreen.
clWebDarkSlateBlue (see page 2930)	This is constant Graphics.clWebDarkSlateBlue.
clWebDarkSlategray (see page 2930)	This is constant Graphics.clWebDarkSlategray.
clWebDarkTurquoise (see page 2930)	This is constant Graphics.clWebDarkTurquoise.
clWebDarkViolet (see page 2931)	This is constant Graphics.clWebDarkViolet.
clWebDeepPink (see page 2931)	This is constant Graphics.clWebDeepPink.
clWebDeepskyBlue (see page 2931)	This is constant Graphics.clWebDeepskyBlue.
clWebDimGray (see page 2932)	This is constant Graphics.clWebDimGray.
clWebDodgerBlue (see page 2932)	This is constant Graphics.clWebDodgerBlue.
clWebFirebrick (see page 2932)	This is constant Graphics.clWebFirebrick.
clWebFloralWhite (see page 2933)	This is constant Graphics.clWebFloralWhite.
clWebForestGreen (see page 2933)	This is constant Graphics.clWebForestGreen.
clWebFuchsia (see page 2933)	This is constant Graphics.clWebFuchsia.
clWebGainsboro (see page 2934)	This is constant Graphics.clWebGainsboro.

clWebGhostWhite (see page 2934)	This is constant Graphics.clWebGhostWhite.
clWebGold (see page 2934)	This is constant Graphics.clWebGold.
clWebGoldenRod (see page 2935)	This is constant Graphics.clWebGoldenRod.
clWebGray (see page 2935)	This is constant Graphics.clWebGray.
clWebGreen (see page 2935)	This is constant Graphics.clWebGreen.
clWebGreenYellow (see page 2936)	This is constant Graphics.clWebGreenYellow.
clWebHoneydew (see page 2936)	This is constant Graphics.clWebHoneydew.
clWebHotPink (see page 2936)	This is constant Graphics.clWebHotPink.
clWebIndianRed (see page 2937)	This is constant Graphics.clWebIndianRed.
clWebIndigo (see page 2937)	This is constant Graphics.clWebIndigo.
clWebIvory (see page 2937)	This is constant Graphics.clWebIvory.
clWebKhaki (see page 2938)	This is constant Graphics.clWebKhaki.
clWebLavender (see page 2938)	This is constant Graphics.clWebLavender.
clWebLavenderBlush (see page 2938)	This is constant Graphics.clWebLavenderBlush.
clWebLawnGreen (see page 2939)	This is constant Graphics.clWebLawnGreen.
clWebLemonChiffon (see page 2939)	This is constant Graphics.clWebLemonChiffon.
clWebLightBlue (see page 2939)	This is constant Graphics.clWebLightBlue.
clWebLightCoral (see page 2940)	This is constant Graphics.clWebLightCoral.
clWebLightCyan (see page 2940)	This is constant Graphics.clWebLightCyan.
clWebLightGoldenrodYellow (see page 2940)	This is constant Graphics.clWebLightGoldenrodYellow.
clWebLightGreen (see page 2941)	This is constant Graphics.clWebLightGreen.
clWebLightgrey (see page 2941)	This is constant Graphics.clWebLightgrey.
clWebLightPink (see page 2941)	This is constant Graphics.clWebLightPink.
clWebLightSalmon (see page 2942)	This is constant Graphics.clWebLightSalmon.
clWebLightSeaGreen (see page 2942)	This is constant Graphics.clWebLightSeaGreen.
clWebLightSkyBlue (see page 2942)	This is constant Graphics.clWebLightSkyBlue.
clWebLightSlateGray (see page 2943)	This is constant Graphics.clWebLightSlateGray.
clWebLightSteelBlue (see page 2943)	This is constant Graphics.clWebLightSteelBlue.
clWebLightYellow (see page 2943)	This is constant Graphics.clWebLightYellow.
clWebLime (see page 2944)	This is constant Graphics.clWebLime.
clWebLimeGreen (see page 2944)	This is constant Graphics.clWebLimeGreen.
clWebLinen (see page 2944)	This is constant Graphics.clWebLinen.
clWebMagenta (see page 2945)	This is constant Graphics.clWebMagenta.
clWebMaroon (see page 2945)	This is constant Graphics.clWebMaroon.
clWebMediumAquamarine (see page 2945)	This is constant Graphics.clWebMediumAquamarine.
clWebMediumBlue (see page 2946)	This is constant Graphics.clWebMediumBlue.
clWebMediumOrchid (see page 2946)	This is constant Graphics.clWebMediumOrchid.
clWebMediumPurple (see page 2946)	This is constant Graphics.clWebMediumPurple.
clWebMediumSeaGreen (see page 2947)	This is constant Graphics.clWebMediumSeaGreen.
clWebMediumSlateBlue (see page 2947)	This is constant Graphics.clWebMediumSlateBlue.
clWebMediumSpringGreen (see page 2947)	This is constant Graphics.clWebMediumSpringGreen.
clWebMediumTurquoise (see page 2948)	This is constant Graphics.clWebMediumTurquoise.
clWebMediumVioletRed (see page 2948)	This is constant Graphics.clWebMediumVioletRed.
clWebMidnightBlue (see page 2948)	This is constant Graphics.clWebMidnightBlue.
clWebMintcream (see page 2949)	This is constant Graphics.clWebMintcream.
clWebMistyRose (see page 2949)	This is constant Graphics.clWebMistyRose.
clWebMoccasin (see page 2949)	This is constant Graphics.clWebMoccasin.
clWebNavajoWhite (see page 2950)	This is constant Graphics.clWebNavajoWhite.
clWebNavy (see page 2950)	This is constant Graphics.clWebNavy.
clWebOldLace (see page 2950)	This is constant Graphics.clWebOldLace.
clWebOlive (see page 2951)	This is constant Graphics.clWebOlive.
clWebOliveDrab (see page 2951)	This is constant Graphics.clWebOliveDrab.
clWebOrange (see page 2951)	This is constant Graphics.clWebOrange.
clWebOrangeRed (see page 2952)	This is constant Graphics.clWebOrangeRed.

clWebOrchid (see page 2952)	This is constant Graphics.clWebOrchid.
clWebPaleGoldenrod (see page 2952)	This is constant Graphics.clWebPaleGoldenrod.
clWebPaleGreen (see page 2953)	This is constant Graphics.clWebPaleGreen.
clWebPaleTurquoise (see page 2953)	This is constant Graphics.clWebPaleTurquoise.
clWebPaleVioletRed (see page 2953)	This is constant Graphics.clWebPaleVioletRed.
clWebPapayaWhip (see page 2954)	This is constant Graphics.clWebPapayaWhip.
clWebPeachPuff (see page 2954)	This is constant Graphics.clWebPeachPuff.
clWebPeru (see page 2954)	This is constant Graphics.clWebPeru.
clWebPink (see page 2955)	This is constant Graphics.clWebPink.
clWebPlum (see page 2955)	This is constant Graphics.clWebPlum.
clWebPowderBlue (see page 2955)	This is constant Graphics.clWebPowderBlue.
clWebPurple (see page 2956)	This is constant Graphics.clWebPurple.
clWebRed (see page 2956)	This is constant Graphics.clWebRed.
clWebRosyBrown (see page 2956)	This is constant Graphics.clWebRosyBrown.
clWebRoyalBlue (see page 2957)	This is constant Graphics.clWebRoyalBlue.
clWebSaddleBrown (see page 2957)	This is constant Graphics.clWebSaddleBrown.
clWebSalmon (see page 2957)	This is constant Graphics.clWebSalmon.
clWebSandyBrown (see page 2958)	This is constant Graphics.clWebSandyBrown.
clWebSeaGreen (see page 2958)	This is constant Graphics.clWebSeaGreen.
clWebSeashell (see page 2958)	This is constant Graphics.clWebSeashell.
clWebSienna (see page 2959)	This is constant Graphics.clWebSienna.
clWebSilver (see page 2959)	This is constant Graphics.clWebSilver.
clWebSkyBlue (see page 2959)	This is constant Graphics.clWebSkyBlue.
clWebSlateBlue (see page 2960)	This is constant Graphics.clWebSlateBlue.
clWebSlateGray (see page 2960)	This is constant Graphics.clWebSlateGray.
clWebSnow (see page 2960)	This is constant Graphics.clWebSnow.
clWebSpringGreen (see page 2961)	This is constant Graphics.clWebSpringGreen.
clWebSteelBlue (see page 2961)	This is constant Graphics.clWebSteelBlue.
clWebTan (see page 2961)	This is constant Graphics.clWebTan.
clWebTeal (see page 2962)	This is constant Graphics.clWebTeal.
clWebThistle (see page 2962)	This is constant Graphics.clWebThistle.
clWebTomato (see page 2962)	This is constant Graphics.clWebTomato.
clWebTurquoise (see page 2963)	This is constant Graphics.clWebTurquoise.
clWebViolet (see page 2963)	This is constant Graphics.clWebViolet.
clWebWheat (see page 2963)	This is constant Graphics.clWebWheat.
clWebWhite (see page 2964)	This is constant Graphics.clWebWhite.
clWebWhiteSmoke (see page 2964)	This is constant Graphics.clWebWhiteSmoke.
clWebYellow (see page 2964)	This is constant Graphics.clWebYellow.
clWebYellowGreen (see page 2965)	This is constant Graphics.clWebYellowGreen.
clWhite (see page 2965)	Represents the color white.
clWindow (see page 2965)	Represents the default Window background color.
clWindowFrame (see page 2966)	Represents the default Window frame background color.
clWindowText (see page 2966)	Represents the default Window text color.
clYellow (see page 2967)	Represents the color yellow.
cmBlackness (see page 2967)	TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with black.
cmDstInvert (see page 2968)	TCopyMode (see page 2880) value : inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 2968)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 2969)	TCopyMode (see page 2880) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.
cmNotSrcCopy (see page 2969)	TCopyMode (see page 2880) value : copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 2970)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.

cmPatCopy (see page 2970)	TCopyMode (see page 2880) value : copies the source pattern to the canvas.
cmPatInvert (see page 2971)	TCopyMode (see page 2880) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.
cmPatPaint (see page 2971)	TCopyMode (see page 2880) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 2972)	TCopyMode (see page 2880) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 2972)	TCopyMode (see page 2880) value : copies the source bitmap to the canvas.
cmSrcErase (see page 2973)	TCopyMode (see page 2880) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 2973)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 2974)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 2974)	TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with white.
ExtendedColorsCount (see page 2975)	The total number of extended colors available
rc3_Cursor (see page 2976)	This is constant Graphics.rc3_Cursor.
rc3_Icon (see page 2976)	This is constant Graphics.rc3_Icon.
rc3_StockIcon (see page 2976)	This is constant Graphics.rc3_StockIcon.
StandardColorsCount (see page 2977)	The total number of standard colors available
WebColorsCount (see page 2978)	The total number of web colors available

Functions

	Name	Description
	AllocPatternBitmap (see page 2836)	Creates a TBitmap (see page 2628) filled with a pattern.
	BytesPerScanline (see page 2837)	This is function Graphics.BytesPerScanline.
	CharsetToIdent (see page 2837)	Obtains the name of a character set.
	ColorToIdent (see page 2838)	Provides the symbolic name of a TColor (see page 2878) value.
	ColorToRGB (see page 2838)	Converts a TColor (see page 2878) value into an RGB representation of the color.
	ColorToString (see page 2840)	Returns a string that represents the name of a TColor (see page 2878) value.
	CopyPalette (see page 2841)	Creates a new palette object that matches an existing palette.
	CreateGrayMappedBmp (see page 2841)	Remaps the standard gray colors in a bitmap with the system grays.
	CreateGrayMappedRes (see page 2842)	Remaps the standard gray colors in a bitmap resource with the system grays.
	CreateMappedBmp (see page 2842)	Changes the color table in a bitmap.
	CreateMappedRes (see page 2843)	Changes the color table in a bitmap resource.
	FreeMemoryContexts (see page 2844)	This is function Graphics.FreeMemoryContexts.
	GetCharsetValues (see page 2844)	Passes the name of every predefined character set string to a callback function.
	GetColorValues (see page 2848)	Passes the name of every predefined color constant to a callback function.
	GetDefFontCharSet (see page 2848)	Returns the character set of the default system font.
	GetDIB (see page 2849)	Returns the info header and image for a device independent bitmap.
	GetDIBSizes (see page 2849)	Returns the size of the info header and of the image for a device independent bitmap.
	GraphicExtension (see page 2850)	Returns the default file-name extension of a graphics object.
	GraphicFileMask (see page 2852)	Returns a file mask that specifies the valid extensions for a graphic class.
	GraphicFilter (see page 2854)	Returns a file filter compatible with the Filter property of an Open or Save dialog.
	IdentToCharset (see page 2857)	Translates the name of a character set constant to the corresponding character set.
	IdentToColor (see page 2857)	Translates the name of a color constant to the corresponding TColor (see page 2878) value.
	PaletteChanged (see page 2858)	This is function Graphics.PaletteChanged.
	StringToColor (see page 2858)	Translates a string representation of a color to the corresponding TColor (see page 2878) value.
	TransparentStretchBlt (see page 2859)	This is function Graphics.TransparentStretchBlt.

Interfaces

	Name	Description
	IChangeNotifier (see page 2834)	IChangeNotifier is the interface used for responding to status changes.

Structs, Records, Enums

	Name	Description
	TAlphaFormat (see page 2860)	This is record Graphics.TAlphaFormat.
	TBitmapHandleType (see page 2861)	TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.
	TBrushData (see page 2861)	This is record Graphics.TBrushData.
	TBrushStyle (see page 2862)	TBrushStyle specifies the pattern on a brush.
	TCanvasOrientation (see page 2863)	TCanvasOrientation specifies the pixel coordinate system that a canvas uses.
	TCanvasStates (see page 2863)	This is record Graphics.TCanvasStates.
	TCursorOrIcon (see page 2864)	This is record Graphics.TCursorOrIcon.
	TFillMode (see page 2864)	This is record Graphics.TFillMode.
	TFillStyle (see page 2864)	TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.
	TFontData (see page 2865)	This is record Graphics.TFontData.
	TFontPitch (see page 2866)	TFontPitch indicates whether the characters in a font have the same width.
	TFontStyle (see page 2866)	TFontStyle and TFontStyles (see page 2883) specify style characteristics of a font.
	TIconRec (see page 2867)	This is record Graphics.TIconRec.
	TPenData (see page 2868)	This is record Graphics.TPenData.
	TPenMode (see page 2868)	TPenMode indicates how the color of a pen interacts with the color of a canvas.
	TPenStyle (see page 2870)	TPenStyle indicates the type of line a pen draws.
	TPixelFormat (see page 2871)	TPixelFormat specifies the format of a bitmap.
	TProgressStage (see page 2872)	TProgressStage indicates the stage of a lengthy operation.
	TResData (see page 2872)	This is record Graphics.TResData.
	TResource (see page 2873)	This is record Graphics.TResource.
	TTextFormats (see page 2873)	This is record Graphics.TTextFormats.
	TTransparentMode (see page 2874)	TTransparentMode indicates how the transparent color of a bitmap is determined.

Types

Name	Description
HENHMETAFILE (see page 2875)	Meta file handle type.
HMETAFILE (see page 2876)	This is type Graphics.HMETAFILE.
PColor (see page 2876)	PColor is a pointer to a TColor (see page 2878) type.
PCursorOrIcon (see page 2876)	This is type Graphics.PCursorOrIcon.
PIconRec (see page 2877)	This is type Graphics.PIconRec.
PResource (see page 2877)	This is type Graphics.PResource.
TCanvasState (see page 2877)	This is type Graphics.TCanvasState.
TColor (see page 2878)	TColor is used to specify the color of a Windows-only control.
TCopyMode (see page 2880)	TCopyMode specifies how a graphical image is copied onto a canvas.
TFontCharset (see page 2881)	TFontCharset represents the character set of a font.
TFontDataName (see page 2882)	This is type Graphics.TFontDataName.
TFontName (see page 2882)	TFontName is a string that holds the typeface name of a font.
TFontStyles (see page 2883)	TFontStyle (see page 2866) and TFontStyles specify style characteristics of a font.
TFontStylesBase (see page 2887)	This is type Graphics.TFontStylesBase.
TGraphicClass (see page 2887)	TGraphicClass defines the metaclass for TGraphic (see page 2748).
TProgressEvent (see page 2888)	TProgressEvent is the type of event handlers that provide feedback during lengthy operations.
TTextFormat (see page 2889)	TTextFormat indicates how a text viewer control displays its contents.

Variables

Name	Description
DDBsOnly (see page 2890)	Determines whether bitmap handles are created as device-dependent bitmaps (DDB).
DefFontData (see page 2891)	This is variable Graphics.DefFontData.
SystemPalette16 (see page 2891)	Contains the 16 VGA system colors.

8.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidGraphic (see page 2627)	EInvalidGraphic is the exception class for unrecognized graphic files.
	EInvalidGraphicOperation (see page 2628)	EInvalidGraphicOperation is the exception class for illegal operations on graphics.
	TBitmap (see page 2628)	TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).
	TBitmapImage (see page 2662)	TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 2628) object.
	TBrush (see page 2664)	TBrush represents the color and pattern used to fill solid shapes.
	TBrushRecall (see page 2671)	TBrushRecall saves and restores a TBrush (see page 2664) object.
	TCanvas (see page 2673)	TCanvas provides an abstract drawing space for objects that must render their own images.
	TFont (see page 2730)	TFont encapsulates a system font.
	TFontRecall (see page 2747)	TFontRecall saves and restores the properties of a TFont (see page 2730) object.
	TGraphic (see page 2748)	TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.
	TGraphicsObject (see page 2764)	TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 2664), TFont (see page 2730), and TPen (see page 2801).
	TIcon (see page 2768)	TIcon is an encapsulation of a Windows icon.
	TIconImage (see page 2778)	TIconImage is the internal representation of the bitmap image for a TIcon (see page 2768) object.
	TMetafile (see page 2780)	TMetafile is an encapsulation of the Win32 Enhanced (see page 2793) metafile.
	TMetafileCanvas (see page 2794)	TMetafileCanvas allows applications to create metafile images from scratch.
	TMetafileImage (see page 2799)	TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 2780) object.
	TPen (see page 2801)	TPen is used to draw lines or outline shapes on a canvas.
	TPenRecall (see page 2811)	TPenRecall saves and restores the properties of a TPen (see page 2801) object.
	TPicture (see page 2812)	TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.
	TSharedImage (see page 2830)	TSharedImage is the base class for classes that contain images used for graphics objects.

8.1.1 EInvalidGraphic Class

EInvalidGraphic is the exception class for unrecognized graphic files.

Class Hierarchy

Pascal

```
EInvalidGraphic = class(Exception);
```

C++

```
class EInvalidGraphic : public Exception;
```

File

Graphics

Description

EInvalidGraphic is raised when

An application attempts to load a file that is not a valid bitmap, icon, metafile, or user-defined graphic type into a picture.

An application attempts to load a file with an unrecognized extension.

An image does not match the format specified in a call to LoadFromClipboardFormat or SaveToClipboardFormat.

An application attempts to set a bitmap's PixelFormat property to an unsupported value.

8.1.2 EInvalidGraphicOperation Class

EInvalidGraphicOperation is the exception class for illegal operations on graphics.

Class Hierarchy

Pascal

```
EInvalidGraphicOperation = class(Exception);
```

C++

```
class EInvalidGraphicOperation : public Exception;
```

File

Graphics

Description

EInvalidGraphicOperation is raised when an illegal operation is attempted on a graphic. For example, an EInvalidGraphicOperation exception is raised when an application attempts to resize an icon or copy an icon to the Clipboard.

See Also

TCanvas (see page 2673)

TIcon (see page 2768)

LoadFromClipboardFormat

SaveToClipboardFormat

TGraphic (see page 2748)

TGraphic (see page 2748)

8.1.3 TBitmap Class

TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).

Class Hierarchy

Pascal

```
TBitmap = class(TGraphic);
```

C++

```
class TBitmap : public TGraphic;
```

File

Graphics

Description

A bitmap is a powerful graphics object used to create, manipulate (scale, scroll, rotate, and paint), and store images in memory and as files on a disk.

TBitmap contains an internal image of the bitmap graphic and automatically manages realization of the palette when drawn.

To draw a bitmap on a canvas, call the Draw (see page 2636) or StretchDraw methods of a TCanvas (see page 2673) object, passing a TBitmap as a parameter.

Creating copies of a TBitmap is very fast since the handle is copied rather than the image. If the image is modified and the handle is shared by more than one TBitmap object, the image is copied before the modification is performed (that is, copy on write).

See Also

TCanvas (see page 2673)

TClipboard

TGraphic (see page 2748)

TPicture (see page 2812)

Using the Main VCL

8.1.3.1 TBitmap Members

The following tables list the members exposed by TBitmap.

Protected Methods

	Name	Description
	Changed (see page 2635)	This is Changed, a member of class TBitmap.
	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 2636)	This is Draw, a member of class TBitmap.
	DrawTransparent (see page 2637)	This is DrawTransparent, a member of class TBitmap.
	Equals (see page 2752)	This is the overview for the Equals method overload.
	GetEmpty (see page 2638)	This is GetEmpty, a member of class TBitmap.
	GetHeight (see page 2638)	This is GetHeight, a member of class TBitmap.
	GetPalette (see page 2639)	This is GetPalette, a member of class TBitmap.
	GetSupportsPartialTransparency (see page 2639)	This is GetSupportsPartialTransparency, a member of class TBitmap.
	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.

	GetWidth (see page 2639)	This is GetWidth, a member of class TBitmap.
	HandleNeeded (see page 2640)	This is HandleNeeded, a member of class TBitmap.
	MaskHandleNeeded (see page 2644)	This is MaskHandleNeeded, a member of class TBitmap.
	PaletteNeeded (see page 2645)	This is PaletteNeeded, a member of class TBitmap.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.
	ReadData (see page 2645)	This is ReadData, a member of class TBitmap.
	SetHeight (see page 2649)	This is SetHeight, a member of class TBitmap.
	SetPalette (see page 2649)	This is SetPalette, a member of class TBitmap.
	SetTransparent (see page 2759)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 2650)	This is SetWidth, a member of class TBitmap.
	WriteData (see page 2650)	This is WriteData, a member of class TBitmap.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	Assign (see page 2633)	Copies a new bitmap image to the bitmap object.
	Create (see page 2631)	Instantiates a bitmap object.
	Destroy (see page 2632)	Disposes of the instance of the bitmap object.
	Dormant (see page 2635)	Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.
	FreeImage (see page 2637)	Frees the cached file image stored in memory by the bitmap.
	HandleAllocated (see page 2639)	Indicates whether the TBitmap (see page 2628) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2640)	Loads a bitmap from the Clipboard into the bitmap object.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromResourceID (see page 2642)	Loads a bitmap from a resource into the bitmap object.
	LoadFromResourceName (see page 2642)	Loads a bitmap resource into the bitmap object.
	LoadFromStream (see page 2643)	Loads the bitmap from a stream into the bitmap object.
	Mask (see page 2644)	Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 2660) with white and every other color with black.
	ReleaseHandle (see page 2645)	Returns the handle to the bitmap so that the TBitmap (see page 2628) object no longer knows about the handle.
	ReleaseMaskHandle (see page 2646)	Returns the handle to the bitmap's mask so that the TBitmap (see page 2628) object no longer knows about the mask handle.
	ReleasePalette (see page 2647)	Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 2628) object.
	SaveToClipboardFormat (see page 2647)	Creates a local copy of the bitmap in Clipboard format.
	SaveToFile (see page 2757)	Saves a graphics image to a file.
	SaveToStream (see page 2649)	Saves the bitmap to a stream.
	SetSize (see page 2650)	Specifies the size of the bitmap.

Public Properties

	Name	Description
	AlphaFormat (see page 2651)	This is AlphaFormat, a member of class TBitmap.
R	Canvas (see page 2651)	Provides access to a drawing surface that represents the bitmap.
R	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Handle (see page 2652)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	HandleType (see page 2653)	Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	IgnorePalette (see page 2654)	Determines whether the bitmap realizes its palette when drawing its image.

	MaskHandle (see page 2654)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Monochrome (see page 2654)	Determines whether the bitmap displays its image in monochrome.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.
	PixelFormat (see page 2656)	Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.
R	ScanLine (see page 2658)	Provides indexed access to each line of pixels.
R	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	TransparentColor (see page 2660)	Determines which color of the bitmap is to be transparent when the bitmap is drawn.
	TransparentMode (see page 2661)	Determines whether the TransparentColor (see page 2660) property's value is automatically calculated or stored with the bitmap object.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

8.1.3.2 TBitmap.Create Constructor

Instantiates a bitmap object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TBitmap();
```

Description

Call Create to instantiate a bitmap object at runtime. Create is also called automatically when a bitmap image is loaded into a TImage.

Create allocates memory for a bitmap object, and calls the inherited Create. Then it creates a TBitmapImage (see page 2662) as the internal image that represents the bitmap.

C++ Examples:

```
/*
This example requires a resource file in the sources directory.
*/

#pragma resource "extrares.res"
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromResourceName((int)HInstance, "Live");
    Canvas->Draw(12,12,Bitmap1.get());
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example demonstrates how to access a resource
```

```
file.  Specify the resource file with the $R directive
below. The RC file associated with the resource file
relates the resource name with the bitmap.
}

{$R extrares.res}

procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1 : TBitMap;
begin
  BitMap1 := TBitMap.Create;
  try
    BitMap1.LoadFromResourceName(HInstance,'Live');
    Canvas.Draw(12,12,BitMap1);
    BitMap1.LoadFromResourceName(HInstance,'Dead');
    Canvas.Draw(12,102,BitMap1);
  finally
    BitMap1.Free;
  end;
end;
```

See Also

TBitmapImage (see page 2662)

8.1.3.3 TBitmap.Destroy Destructor

Disposes of the instance of the bitmap object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBitmap();
```

Description

Destroy is the destructor for a TBitmap (see page 2628) object.

Do not call the destructor directly in an application. Instead, call Free. Free verifies that the bitmap object is not nil before it calls Destroy.

Destroy releases the internal bitmap image (TBitmapImage (see page 2662) object) and frees the canvas before calling the inherited destructor.

See Also

System::TObject::Free

delete

TBitmapImage (see page 2662)

8.1.3.4 TBitmap Methods

The methods of the TBitmap class are listed here.

Protected Methods

	Name	Description
	Changed (see page 2635)	This is Changed, a member of class TBitmap.
	Draw (see page 2636)	This is Draw, a member of class TBitmap.
	DrawTransparent (see page 2637)	This is DrawTransparent, a member of class TBitmap.
	GetEmpty (see page 2638)	This is GetEmpty, a member of class TBitmap.
	GetHeight (see page 2638)	This is GetHeight, a member of class TBitmap.
	GetPalette (see page 2639)	This is GetPalette, a member of class TBitmap.
	GetSupportsPartialTransparency (see page 2639)	This is GetSupportsPartialTransparency, a member of class TBitmap.
	GetWidth (see page 2639)	This is GetWidth, a member of class TBitmap.
	HandleNeeded (see page 2640)	This is HandleNeeded, a member of class TBitmap.
	MaskHandleNeeded (see page 2644)	This is MaskHandleNeeded, a member of class TBitmap.
	PaletteNeeded (see page 2645)	This is PaletteNeeded, a member of class TBitmap.
	ReadData (see page 2645)	This is ReadData, a member of class TBitmap.
	SetHeight (see page 2649)	This is SetHeight, a member of class TBitmap.
	SetPalette (see page 2649)	This is SetPalette, a member of class TBitmap.
	SetWidth (see page 2650)	This is SetWidth, a member of class TBitmap.
	WriteData (see page 2650)	This is WriteData, a member of class TBitmap.

Public Methods

	Name	Description
	Assign (see page 2633)	Copies a new bitmap image to the bitmap object.
	Dormant (see page 2635)	Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.
	FreeImage (see page 2637)	Frees the cached file image stored in memory by the bitmap.
	HandleAllocated (see page 2639)	Indicates whether the TBitmap (see page 2628) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2640)	Loads a bitmap from the Clipboard into the bitmap object.
	LoadFromResourceID (see page 2642)	Loads a bitmap from a resource into the bitmap object.
	LoadFromResourceName (see page 2642)	Loads a bitmap resource into the bitmap object.
	LoadFromStream (see page 2643)	Loads the bitmap from a stream into the bitmap object.
	Mask (see page 2644)	Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 2660) with white and every other color with black.
	ReleaseHandle (see page 2645)	Returns the handle to the bitmap so that the TBitmap (see page 2628) object no longer knows about the handle.
	ReleaseMaskHandle (see page 2646)	Returns the handle to the bitmap's mask so that the TBitmap (see page 2628) object no longer knows about the mask handle.
	ReleasePalette (see page 2647)	Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 2628) object.
	SaveToClipboardFormat (see page 2647)	Creates a local copy of the bitmap in Clipboard format.
	SaveToStream (see page 2649)	Saves the bitmap to a stream.
	SetSize (see page 2650)	Specifies the size of the bitmap.

8.1.3.4.1 TBitmap.Assign Method

Copies a new bitmap image to the bitmap object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the bitmap image contained in Source to the bitmap object. If Source is not a bitmap, Assign calls the inherited Assign method, which can copy an image from any class that knows how to copy to a TBitmap (see page 2628) object. If the

bitmap needs to be changed, the actual bitmap image is copied before the changes are made (copy on write).

Note: An object of one type can always be assigned to another object of the same type. Also, the Source can be of type TPicture (see page 2812) if the Graphic property of the picture is a bitmap.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
```

```
    BitMap2.Free;
  end;
end;
```

See Also

Assign

TPicture (see page 2812)

Graphic (see page 2827)

8.1.3.4.2 TBitmap.Changed Method

Pascal

```
procedure Changed(Sender: TObject); override;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TBitmap.

8.1.3.4.3 TBitmap.Dormant Method

Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.

Pascal

```
procedure Dormant;
```

C++

```
__fastcall Dormant();
```

Description

Use Dormant to change the format of the bitmap in memory thereby reducing the amount of GDI resources used by the application.

Dormant creates a bitmap image in memory using a memory stream object. This preserves the image so that the bitmap can then free the HBITMAP (accessed through the Handle (see page 2652) property) that was assigned to it.

DIB handles may use fewer Win95 GDI resources than DDB, but DIBs may also use more memory than DDBs, depending on the current video driver and mode.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
```

```
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

Handle (see page 2652)

TBitmapImage (see page 2662)

TMemoryStream

8.1.3.4.4 TBitmap.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TBitmap.

8.1.3.4.5 TBitmap.DrawTransparent Method

Pascal

```
procedure DrawTransparent(ACanvas: TCanvas; const Rect: TRect; Opacity: Byte); override;
```

C++

```
virtual __fastcall DrawTransparent(TCanvas ACanvas, const TRect Rect, Byte Opacity);
```

Description

This is DrawTransparent, a member of class TBitmap.

8.1.3.4.6 TBitmap.FreeImage Method

Frees the cached file image stored in memory by the bitmap.

Pascal

```
procedure FreeImage;
```

C++

```
__fastcall FreeImage();
```

Description

Use FreeImage to reduce the memory requirements of an application when color depth and pixel format are not an issue. Freeing the image releases the memory allocated for the bitmap image when it was originally loaded to disk. Consequently, some of the original pixel format of the bitmap is lost (for example, if you changed its format to a DIB) as well as the color depth of the bitmap.

When a bitmap is loaded into a bitmap object, the bitmap object creates an image of the loaded bitmap in memory. If the bitmap isn't changed, the memory image is used when saving the bitmap, to verify that the bitmap has not lost color depth or changed the pixel format.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
```

```
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

8.1.3.4.7 TBitmap.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TBitmap.

8.1.3.4.8 TBitmap.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TBitmap.

8.1.3.4.9 TBitmap.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TBitmap.

8.1.3.4.10 TBitmap.GetSupportsPartialTransparency Method

Pascal

```
function GetSupportsPartialTransparency: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetSupportsPartialTransparency();
```

Description

This is GetSupportsPartialTransparency, a member of class TBitmap.

8.1.3.4.11 TBitmap.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TBitmap.

8.1.3.4.12 TBitmap.HandleAllocated Method

Indicates whether the TBitmap (see page 2628) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 2652) property is set to an HBITMAP value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 2652) property. When an application reads the Handle (see page 2652) property, it automatically creates a bitmap and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 2652)

8.1.3.4.13 TBitmap.HandleNeeded Method

Pascal

```
procedure HandleNeeded;
```

C++

```
__fastcall HandleNeeded();
```

Description

This is HandleNeeded, a member of class TBitmap.

8.1.3.4.14 TBitmap.LoadFromClipboardFormat Method

Loads a bitmap from the Clipboard into the bitmap object.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is called if the bitmap is registered with the TPicture (see page 2812) object using the RegisterClipboardFormat method.

LoadFromClipboardFormat replaces the current image with the data pointed to by the AData parameter. The palette for the bitmap is specified by the APalette parameter.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5, 5, Bitmap.get());
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
```

```
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
```

```
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TClipboard

TPicture (see page 2812)

RegisterClipboardFormat (see page 2819)

8.1.3.4.15 TBitmap.LoadFromResourceID Method

Loads a bitmap from a resource into the bitmap object.

Pascal

```
procedure LoadFromResourceID(Instance: THandle; ResID: Integer);
```

C++

```
__fastcall LoadFromResourceID(THandle Instance, int ResID);
```

Description

Use LoadFromResourceID to load a specified bitmap resource along with palette information from a module's executable file.

Instance is the handle of the module that contains the resource.

ResID is the resource ID for the bitmap.

See Also

LoadFromResourceName (see page 2642)

8.1.3.4.16 TBitmap.LoadFromResourceName Method

Loads a bitmap resource into the bitmap object.

Pascal

```
procedure LoadFromResourceName(Instance: THandle; const ResName: String);
```

C++

```
__fastcall LoadFromResourceName(THandle Instance, const UnicodeString ResName);
```

Description

LoadFromResourceName loads the specified bitmap resource along with palette information from a module's executable file.

Instance is the handle of the module that contains the resource.

ResName is the name of the resource to load.

Note: Use this routine to load bitmaps from RES files instead of the LoadBitmap API. LoadBitmap does not support 256-color

images.

C++ Examples:

```
/*
This example requires a resource file in the sources directory.
*/

#pragma resource "extrares.res"
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromResourceName((int)HInstance, "Live");
    Canvas->Draw(12,12,Bitmap1.get());
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example demonstrates how to access a resource
file.  Specify the resource file with the $R directive
below. The RC file associated with the resource file
relates the resource name with the bitmap.
}

{$R extrares.res}

procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1 : TBitMap;
begin
  BitMap1 := TBitMap.Create;
  try
    BitMap1.LoadFromResourceName(HInstance,'Live');
    Canvas.Draw(12,12,BitMap1);
    BitMap1.LoadFromResourceName(HInstance,'Dead');
    Canvas.Draw(12,102,BitMap1);
  finally
    BitMap1.Free;
  end;
end;
```

See Also

LoadFromResourceID (see page 2642)

HInstance

8.1.3.4.17 TBitmap.LoadFromStream Method

Loads the bitmap from a stream into the bitmap object.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to load the bitmap image from a stream. The Stream parameter specifies the stream from which to read the bitmap image. For example, use LoadFromStream to load a bitmap image from a BLOB stream that reads from a graphic field in a dataset.

See Also

LoadFromClipboardFormat (see page 2640)

LoadFromResourceName (see page 2642)

LoadFromResourceID (see page 2642)

LoadFromFile (see page 2755)

SaveToStream (see page 2649)

8.1.3.4.18 TBitmap.Mask Method

Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 2660) with white and every other color with black.

Pascal

```
procedure Mask(TransparentColor: TColor);
```

C++

```
__fastcall Mask(TColor TransparentColor);
```

Description

Use Mask to produce a monochrome mask bitmap based on a given transparent color. Mask produces the same image that MaskHandle (see page 2654) does, except that the resulting mask image replaces the image in the TBitmap (see page 2628) object, so it only makes sense to call Mask once. Use MaskHandle (see page 2654) when only a bitmap handle of the mask image is needed.

See Also

ReleaseMaskHandle (see page 2646)

8.1.3.4.19 TBitmap.MaskHandleNeeded Method

Pascal

```
procedure MaskHandleNeeded;
```

C++

```
__fastcall MaskHandleNeeded();
```

Description

This is MaskHandleNeeded, a member of class TBitmap.

8.1.3.4.20 TBitmap.PaletteNeeded Method

Pascal

```
procedure PaletteNeeded;
```

C++

```
__fastcall PaletteNeeded();
```

Description

This is PaletteNeeded, a member of class TBitmap.

8.1.3.4.21 TBitmap.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); override;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TBitmap.

8.1.3.4.22 TBitmap.ReleaseHandle Method

Returns the handle to the bitmap so that the TBitmap (see page 2628) object no longer knows about the handle.

Pascal

```
function ReleaseHandle: HBITMAP;
```

C++

```
__fastcall HBITMAP ReleaseHandle();
```

Description

Use ReleaseHandle to disassociate the bitmap from the bitmap handle. Use it when you need to give a bitmap handle to a routine or object that will assume ownership (or destroy) the bitmap handle.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
```

```
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

ReleaseMaskHandle (see page 2646)

Handle (see page 2652)

8.1.3.4.23 TBitmap.ReleaseMaskHandle Method

Returns the handle to the bitmap's mask so that the TBitmap (see page 2628) object no longer knows about the mask handle.

Pascal

```
function ReleaseMaskHandle: HBITMAP;
```

C++

```
__fastcall HBITMAP ReleaseMaskHandle();
```

Description

Use ReleaseMaskHandle to disassociate the bitmap object from the bitmap mask handle.

See Also

ReleaseHandle (see page 2645)

Handle (see page 2652)

Mask (see page 2644)

8.1.3.4.24 TBitmap.ReleasePalette Method

Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 2628) object.

Pascal

```
function ReleasePalette: HPALETTE;
```

C++

```
__fastcall HPALETTE ReleasePalette();
```

Description

Use ReleasePalette to disassociate the palette from the bitmap image.

8.1.3.4.25 TBitmap.SaveToClipboardFormat Method

Creates a local copy of the bitmap in Clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var Format: Word; var Data: THandle; var APalette: HPALETTE);
override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word Format, THandle Data, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the bitmap to a Clipboard format. The bitmap's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the bitmap can be saved, an application must have registered the format with the TBitmap (see page 2628) object using the RegisterClipboardFormat method.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5, 5, Bitmap.get());
    }
    catch (...)
    {
      MessageBeep(0);
```

```
      ShowMessage("Error reading image from clipboard!");
    }
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
```

```
    AData : THandle;
    APalette : HPALETTE;
  begin
    Bitmap := TBitmap.Create;
    try
      Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
  Shared\Images\Splash\256color\factory.bmp');
      Bitmap.SaveToClipBoardFormat(
        MyFormat,
        AData,
        APalette);
      ClipBoard.SetAsHandle(MyFormat,AData);
    finally
      Bitmap.Free;
    end;
  end;
```

See Also

RegisterClipboardFormat (see page 2819)

LoadFromClipboardFormat (see page 2640)

8.1.3.4.26 TBitmap.SaveToStream Method

Saves the bitmap to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to write the bitmap image to the stream specified by the Stream parameter. For example, specify a TBlobStream object as the Stream parameter to save the bitmap image to a graphic field in a dataset.

See Also

LoadFromStream (see page 2643)

8.1.3.4.27 TBitmap.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TBitmap.

8.1.3.4.28 TBitmap.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPALETTE); override;
```

C++

```
virtual __fastcall SetPalette(HPALETTE Value);
```

Description

This is SetPalette, a member of class TBitmap.

8.1.3.4.29 TBitmap.SetSize Method

Specifies the size of the bitmap.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the bitmap. This results in better performance than setting the height and width separately.

8.1.3.4.30 TBitmap.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TBitmap.

8.1.3.4.31 TBitmap.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); override;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TBitmap.

8.1.3.5 TBitmap Properties

The properties of the TBitmap class are listed here.

Public Properties

	Name	Description
	AlphaFormat (see page 2651)	This is AlphaFormat, a member of class TBitmap.
R	Canvas (see page 2651)	Provides access to a drawing surface that represents the bitmap.

	Handle (see page 2652)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	HandleType (see page 2653)	Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.
	IgnorePalette (see page 2654)	Determines whether the bitmap realizes its palette when drawing its image.
	MaskHandle (see page 2654)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	Monochrome (see page 2654)	Determines whether the bitmap displays its image in monochrome.
	PixelFormat (see page 2656)	Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.
R	ScanLine (see page 2658)	Provides indexed access to each line of pixels.
	TransparentColor (see page 2660)	Determines which color of the bitmap is to be transparent when the bitmap is drawn.
	TransparentMode (see page 2661)	Determines whether the TransparentColor (see page 2660) property's value is automatically calculated or stored with the bitmap object.

8.1.3.5.1 TBitmap.AlphaFormat Property

Pascal

```
property AlphaFormat: TAlphaFormat;
```

C++

```
__property TAlphaFormat AlphaFormat;
```

Description

This is AlphaFormat, a member of class TBitmap.

8.1.3.5.2 TBitmap.Canvas Property

Provides access to a drawing surface that represents the bitmap.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas allows drawing on the bitmap by providing a T Canvas object for this purpose. Drawing on the canvas effectively modifies the underlying bitmap image pixels. Any canvas operation is valid on a bitmap (not just Draw (see page 2636) and StretchDraw) including line drawing, rectangles, and circles. The bitmap object is passed as a parameter to these methods.

A canvas object is created automatically for the bitmap and the property is read-only.

C++ Examples:

```
/*
An examples of TBitmap Color, TGraphic Transparent, TBitmap
TransparentMode, TBitmap TransparentColor and TCanvas Draw.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bitmap(new Graphics::TBitmap);
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
```

```
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap.get());
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
}
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TCanvas (see page 2673)

Draw (see page 2685)

StretchDraw (see page 2704)

8.1.3.5.3 TBitmap.Handle Property

Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.

Pascal

```
property Handle: HBITMAP;
```

C++

```
__property HBITMAP Handle;
```

Description

Use Handle to call a Windows API function that requires the handle of a bitmap object. Pass Handle as the bitmap handle parameter to these functions.

Handle is the HBITMAP encapsulated by the bitmap object. Avoid grabbing the handle directly since it causes the HBITMAP to be copied if more than one TBitmap (see page 2628) shares the handle.

Warning: Be careful when giving the handle to an object or routine. If the receiver takes ownership (and destroys) the bitmap handle, call TBitmap (see page 2628).::ReleaseHandle (see page 2645).

Delphi Examples:

```
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

  Screen.Cursor := crMyCursor;

  bmpMask.Free;
  bmpColor.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  DestroyIcon(Screen.Cursors[crMyCursor]);
end;
```

See Also

HandleType (see page 2653)

MaskHandle (see page 2654)

HandleAllocated (see page 2639)

8.1.3.5.4 TBitmap.HandleType Property

Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.

Pascal

```
property HandleType: TBitmapHandleType;
```

C++

```
__property TBitmapHandleType HandleType;
```

Description

HandleType is used to find or modify the type of the bitmap. Device-dependent bitmaps (DDBs) may be faster to draw on the screen, but may be stored in memory in a device-specific format. Device-independent bitmaps (DIBs) are stored in memory in a

standard format supported by all devices (including printers).

See Also

Handle (see page 2652)

8.1.3.5.5 TBitmap.IgnorePalette Property

Determines whether the bitmap realizes its palette when drawing its image.

Pascal

```
property IgnorePalette: Boolean;
```

C++

```
__property Boolean IgnorePalette;
```

Description

Use IgnorePalette when speed of drawing is a priority. When IgnorePalette is true, the bitmap does not realize its palette when drawing itself, resulting in lower picture quality on 256-color video drivers, but faster drawing of the bitmap image.

Note:

See Also

TBitmapImage (see page 2662)

TDBCtrlGrid

TDBImage

8.1.3.5.6 TBitmap.MaskHandle Property

Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.

Pascal

```
property MaskHandle: HBITMAP;
```

C++

```
__property HBITMAP MaskHandle;
```

Description

Use MaskHandle to call a Windows API function that requires the handle of a bitmap object. Pass MaskHandle as the bitmap handle parameter to these functions. MaskHandle is the HBITMAP encapsulated by the bitmap object's mask.

See Also

Handle (see page 2652)

ReleaseMaskHandle (see page 2646)

8.1.3.5.7 TBitmap.Monochrome Property

Determines whether the bitmap displays its image in monochrome.

Pascal

```
property Monochrome: Boolean;
```

C++

```
__property Boolean Monochrome;
```

Description

If Monochrome is true if the bitmap displays as a monochrome bitmap. Monochrome is false if the bitmap displays in color, that is, if it contains more than one bit per pixel.

Note:

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
```

```
    Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
    BitMap2.Monochrome := true;
    Canvas.Draw(80,80,BitMap2);
    BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
  finally
    BitMap1.Free;
    BitMap2.Free;
  end;
end;
```

8.1.3.5.8 TBitmap.PixelFormat Property

Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.

Pascal

```
property PixelFormat: TPixelFormat;
```

C++

```
__property TPixelFormat PixelFormat;
```

Description

Use PixelFormat to change a TBitmap (see page 2628)'s internal image to a particular memory format and color depth, or to find out what memory format and color depth a TBitmap (see page 2628) is using.

For example, PixelFormat can be used to set the pixel format of the bitmap image to 8-bit for video drivers that cannot display the native format of a bitmap image.

Note: The PixelFormat of a JPEG image object applies to the bitmap if the JPEG image is copied to it.

Changing the pixel format is most commonly used with ScanLine (see page 2658), because your code must decode the pixel data accessed by ScanLine (see page 2658). Image-editing tools usually use one pixel for all internal image operations and copy the results to the screen (in whatever format) as the last step.

C++ Examples:

```
/*
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap);
  std::auto_ptr<Graphics::TBitmap> BigBitmap(new Graphics::TBitmap);
  TRGBTriple *ptr, *bigPtr;   // use a (byte *) for pf8bit color
  TPixelFormat pixForm, bigpixForm;
  try
  {
    Bitmap->LoadFromFile("../littlefac.bmp");
    pixForm = Bitmap->PixelFormat;
    bigpixForm  = BigBitmap->PixelFormat;
    Bitmap->PixelFormat = pf24bit;
    BigBitmap->PixelFormat = pf24bit;
    BigBitmap->Height = Bitmap->Height * 2;
    BigBitmap->Width = Bitmap->Width * 2;
    for (int y = 0; y < Bitmap->Height; y++)
    {
      ptr = reinterpret_cast<TRGBTriple *>(Bitmap->ScanLine[y]);
```

```
      for (int x = 0; x < Bitmap->Width; x++)
      {
        int bx = x * 2;
        int by = y * 2;
        bigPtr = reinterpret_cast<TRGBTriple *>(BigBitmap->ScanLine[by]);
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
        bigPtr = reinterpret_cast<TRGBTriple *>(BigBitmap->ScanLine[by + 1]);
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
      }
    }
    Canvas->Draw(0, 0, Bitmap.get());
    Canvas->Draw(200, 200, BigBitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or alter bitmap");
  }
}
```

Delphi Examples:

```
{
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  TRGBTripleArray = ARRAY[Word] of TRGBTriple;
  pRGBTripleArray = ^TRGBTripleArray; // use a PByteArray for pf8bit color
var
  x,y : Integer;
  bx, by : Integer;
  BitMap, BigBitMap : TBitMap;
  P, bigP : pRGBTripleArray;
  pixForm, bigpixForm : TPixelFormat;
begin
  BitMap := TBitMap.create;
  BigBitMap := TBitMap.create;
  try
    BitMap.LoadFromFile('littlefac.bmp');
    pixForm := BitMap.PixelFormat;
    bigpixForm := BigBitMap.PixelFormat;
    BitMap.PixelFormat := pf24bit;
    BigBitMap.PixelFormat := pf24bit;
    BigBitMap.Height := BitMap.Height * 2;
    BigBitMap.Width := BitMap.Width * 2;
    for y := 0 to BitMap.Height - 1 do
    begin
      P := BitMap.ScanLine[y];
      for x := 0 to BitMap.Width - 1 do
      begin
        bx := x * 2;
        by := y * 2;
        bigP := BigBitMap.ScanLine[by];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
        bigP := BigBitMap.ScanLine[by + 1];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
      end;
    end;
    Canvas.Draw(0, 0, BitMap);
    Canvas.Draw(200, 200, BigBitMap);
```

```
    finally
      BitMap.Free;
      BigBitMap.Free;
    end;
  end;
```

See Also

PixelFormat

ScanLine (see page 2658)

HandleType (see page 2653)

8.1.3.5.9 TBitmap.ScanLine Property

Provides indexed access to each line of pixels.

Pascal

```
property ScanLine [Row: Integer]: Pointer;
```

C++

```
__property void * ScanLine[int Row];
```

Description

ScanLine is used only with DIBs (Device Independent Bitmaps) for image editing tools that do low-level pixel work.

C++ Examples:

```
/*
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap);
  std::auto_ptr<Graphics::TBitmap> BigBitmap(new Graphics::TBitmap);
  TRGBTriple *ptr, *bigPtr;   // use a (byte *) for pf8bit color
  TPixelFormat pixForm, bigpixForm;
  try
  {
    Bitmap->LoadFromFile("../littlefac.bmp");
    pixForm = Bitmap->PixelFormat;
    bigpixForm  = BigBitmap->PixelFormat;
    Bitmap->PixelFormat = pf24bit;
    BigBitmap->PixelFormat = pf24bit;
    BigBitmap->Height = Bitmap->Height * 2;
    BigBitmap->Width = Bitmap->Width * 2;
    for (int y = 0; y < Bitmap->Height; y++)
    {
      ptr = reinterpret_cast<TRGBTriple *>(Bitmap->ScanLine[y]);
      for (int x = 0; x < Bitmap->Width; x++)
      {
        int bx = x * 2;
        int by = y * 2;
        bigPtr = reinterpret_cast<TRGBTriple *>(BigBitmap->ScanLine[by]);
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
        bigPtr = reinterpret_cast<TRGBTriple *>(BigBitmap->ScanLine[by + 1]);
```

```
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
      }
    }
    Canvas->Draw(0, 0, Bitmap.get());
    Canvas->Draw(200, 200, BigBitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or alter bitmap");
  }
}
```

Delphi Examples:

```
{
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  TRGBTripleArray = ARRAY[Word] of TRGBTriple;
  pRGBTripleArray = ^TRGBTripleArray; // use a PByteArray for pf8bit color
var
  x,y : Integer;
  bx, by : Integer;
  BitMap, BigBitMap : TBitMap;
  P, bigP : pRGBTripleArray;
  pixForm, bigpixForm : TPixelFormat;
begin
  BitMap := TBitMap.create;
  BigBitMap := TBitMap.create;
  try
    BitMap.LoadFromFile('littlefac.bmp');
    pixForm := BitMap.PixelFormat;
    bigpixForm := BigBitMap.PixelFormat;
    BitMap.PixelFormat := pf24bit;
    BigBitMap.PixelFormat := pf24bit;
    BigBitMap.Height := BitMap.Height * 2;
    BigBitMap.Width := BitMap.Width * 2;
    for y := 0 to BitMap.Height - 1 do
    begin
      P := BitMap.ScanLine[y];
      for x := 0 to BitMap.Width - 1 do
      begin
        bx := x * 2;
        by := y * 2;
        bigP := BigBitMap.ScanLine[by];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
        bigP := BigBitMap.ScanLine[by + 1];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
      end;
    end;
    Canvas.Draw(0, 0, BitMap);
    Canvas.Draw(200, 200, BigBitMap);
  finally
    BitMap.Free;
    BigBitMap.Free;
  end;
end;
```

See Also

PixelFormat (see page 2656)

8.1.3.5.10 TBitmap.TransparentColor Property

Determines which color of the bitmap is to be transparent when the bitmap is drawn.

Pascal

```
property TransparentColor: TColor;
```

C++

```
__property TColor TransparentColor;
```

Description

Use TransparentColor to determine how to draw the bitmap transparently. When the TransparentMode (see page 2661) property is set to tmAuto (default), TransparentColor returns the color of the first pixel in the bitmap image data. For "bottom-up" bitmaps, the first pixel is the bottom leftmost pixel shown onscreen. For "top-down" bitmaps (less common), the first pixel is in the top left corner shown onscreen.

If TransparentColor is assigned, the TransparentMode (see page 2661) is automatically set to tmFixed so that the new transparent color can be used later. If you want TransparentColor to disregard any assignments and return the bottom leftmost pixel color again, set TransparentMode (see page 2661) to tmAuto.

C++ Examples:

```
/*
An examples of TBitmap Color, TGraphic Transparent, TBitmap
TransparentMode, TBitmap TransparentColor and TCanvas Draw.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bitmap(new Graphics::TBitmap);
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap.get());
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
}
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
```

```
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TransparentMode (see page 2661)

8.1.3.5.11 TBitmap.TransparentMode Property

Determines whether the TransparentColor (see page 2660) property's value is automatically calculated or stored with the bitmap object.

Pascal

```
property TransparentMode: TTransparentMode;
```

C++

```
__property TTransparentMode TransparentMode;
```

Description

When TransparentMode is set to tmAuto (the default), the TransparentColor (see page 2660) property returns the color of the bottom-leftmost pixel of the bitmap image. When TransparentMode is set to tmFixed, the TransparentColor (see page 2660) property refers to the color stored in the bitmap object.

C++ Examples:

```
/*
An examples of TBitmap Color, TGraphic Transparent, TBitmap
TransparentMode, TBitmap TransparentColor and TCanvas Draw.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bitmap(new Graphics::TBitmap);
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap.get());
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
}
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
```

```
    try
      with Bitmap do begin
        LoadFromFile('factory.bmp');
        Transparent := True;
        TransParentColor := BitMap.canvas.pixels[50,50];
        Form1.Canvas.Draw(0,0,BitMap);
        TransparentMode := tmAuto;
        Form1.Canvas.Draw(50,50,BitMap);
      end;
    finally
      Bitmap.Free;
    end;
  end;
```

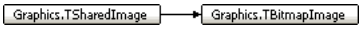
See Also

TransparentColor (see page 2660)

8.1.4 TBitmapImage Class

TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 2628) object.

Class Hierarchy



Pascal

```
TBitmapImage = class(TSharedImage);
```

C++

```
class TBitmapImage : public TSharedImage;
```

File

Graphics

Description

TBitmapImage is used only for internal implementation of the TBitmap (see page 2628) component that is defined in the Graphics (see page 2619) unit. It represents the internal image of the bitmap. All of the data and methods introduced in TBitmapImage are private. They contain information about the specific Windows bitmap resource, including pointers to the standard HBITMAP structures defined by Windows. TBitmapImage is referenced in the constructor of the TBitmap (see page 2628) object when the bitmap image is created.

Note: TBitmapImage is how TBitmap (see page 2628) does reference counting and handle sharing. Multiple instances can refer to the same TBitmap (see page 2628) image. TBitmapImage is the actual owner of the bitmap handle.

See Also

TBitmap (see page 2628)

8.1.4.1 TBitmapImage Members

The following tables list the members exposed by TBitmapImage.

Protected Methods

	Name	Description
	FreeHandle (see page 2663)	This is FreeHandle, a member of class TBitmapImage.

	Reference (see page 2832)	This is Reference, a member of class TSharedImage.
	Release (see page 2832)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
R	RefCount (see page 2832)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
V	Destroy (see page 2663)	Frees the memory associated with the TBitmapImage (see page 2662) object.

8.1.4.2 TBitmapImage.Destroy Destructor

Frees the memory associated with the TBitmapImage (see page 2662) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBitmapImage();
```

Description

Do not call Destroy. The bitmap object automatically destroys the bitmap image.

See Also

System::TObject::Free

delete

8.1.4.3 TBitmapImage Methods

The methods of the TBitmapImage class are listed here.

Protected Methods

	Name	Description
V	FreeHandle (see page 2663)	This is FreeHandle, a member of class TBitmapImage.

8.1.4.3.1 TBitmapImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TBitmapImage.

8.1.5 TBrush Class

TBrush represents the color and pattern used to fill solid shapes.

Class Hierarchy

Pascal

```
TBrush = class(TGraphicsObject);
```

C++

```
class TBrush : public TGraphicsObject;
```

File

Graphics

Description

TBrush encapsulates the Windows brush object (HBRUSH) and is used to fill solid shapes, such as rectangles and ellipses, with a color or pattern.

See Also

TCanvas (see page 2673)

TPen (see page 2801)

Shapes

Using the Main VCL

8.1.5.1 TBrush Members

The following tables list the members exposed by TBrush.

Protected Methods

	Name	Description
	Changed (see page 2765)	Generates an OnChange (see page 2768) event when the properties of the graphics object change.
	GetBitmap (see page 2666)	This is GetBitmap, a member of class TBrush.
	GetColor (see page 2666)	This is GetColor, a member of class TBrush.
	GetHandle (see page 2667)	This is GetHandle, a member of class TBrush.
	GetStyle (see page 2667)	This is GetStyle, a member of class TBrush.
	Lock (see page 2766)	Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.
	SetBitmap (see page 2667)	This is SetBitmap, a member of class TBrush.
	SetColor (see page 2667)	This is SetColor, a member of class TBrush.
	SetHandle (see page 2668)	This is SetHandle, a member of class TBrush.
	SetStyle (see page 2668)	This is SetStyle, a member of class TBrush.
	Unlock (see page 2767)	Releases the lock on execution threads set by the Lock (see page 2766) method.

Public Events

	Name	Description
	OnChange (see page 2768)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	Assign (see page 2666)	Copies the contents of the brush to another brush.
	Create (see page 2665)	Creates and initializes a TBrush (see page 2664) instance.
	Destroy (see page 2665)	Destroys the TBrush (see page 2664) instance and frees its memory.
	HandleAllocated (see page 2766)	Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

Public Properties

	Name	Description
	Bitmap (see page 2668)	Specifies an external bitmap image that defines a pattern for the brush.
	Handle (see page 2670)	Provides access to the handle of a system brush.
	OwnerCriticalSection (see page 2767)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Published Properties

	Name	Description
	Color (see page 2669)	Indicates the color of the brush.
	Style (see page 2670)	Specifies the pattern for the brush.

8.1.5.2 TBrush.Create Constructor

Creates and initializes a TBrush (see page 2664) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall TBrush();
```

Description

Create generates a new TBrush (see page 2664) instance.

8.1.5.3 TBrush.Destroy Destructor

Destroys the TBrush (see page 2664) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBrush();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TBrush (see page 2664) instance is not nil before calling Destroy.

See Also

System::TObject::Free

8.1.5.4 TBrush Methods

The methods of the TBrush class are listed here.

Protected Methods

	Name	Description
	GetBitmap (see page 2666)	This is GetBitmap, a member of class TBrush.
	GetColor (see page 2666)	This is GetColor, a member of class TBrush.
	GetHandle (see page 2667)	This is GetHandle, a member of class TBrush.
	GetStyle (see page 2667)	This is GetStyle, a member of class TBrush.
	SetBitmap (see page 2667)	This is SetBitmap, a member of class TBrush.
	SetColor (see page 2667)	This is SetColor, a member of class TBrush.
	SetHandle (see page 2668)	This is SetHandle, a member of class TBrush.
	SetStyle (see page 2668)	This is SetStyle, a member of class TBrush.

Public Methods

	Name	Description
	Assign (see page 2666)	Copies the contents of the brush to another brush.

8.1.5.4.1 TBrush.Assign Method

Copies the contents of the brush to another brush.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the brush's properties from another TBrush (see page 2664) instance.

See Also

Assign

8.1.5.4.2 TBrush.GetBitmap Method

Pascal

```
function GetBitmap: TBitmap;
```

C++

```
__fastcall TBitmap GetBitmap();
```

Description

This is GetBitmap, a member of class TBrush.

8.1.5.4.3 TBrush.GetColor Method

Pascal

```
function GetColor: TColor;
```

C++

```
__fastcall TColor GetColor();
```

Description

This is GetColor, a member of class TBrush.

8.1.5.4.4 TBrush.GetHandle Method

Pascal

```
function GetHandle: HBrush;
```

C++

```
__fastcall HBrush GetHandle();
```

Description

This is GetHandle, a member of class TBrush.

8.1.5.4.5 TBrush.GetStyle Method

Pascal

```
function GetStyle: TBrushStyle;
```

C++

```
__fastcall TBrushStyle GetStyle();
```

Description

This is GetStyle, a member of class TBrush.

8.1.5.4.6 TBrush.SetBitmap Method

Pascal

```
procedure SetBitmap(Value: TBitmap);
```

C++

```
__fastcall SetBitmap(TBitmap Value);
```

Description

This is SetBitmap, a member of class TBrush.

8.1.5.4.7 TBrush.SetColor Method

Pascal

```
procedure SetColor(Value: TColor);
```

C++

```
__fastcall SetColor(TColor Value);
```

Description

This is SetColor, a member of class TBrush.

8.1.5.4.8 TBrush.SetHandle Method

Pascal

```
procedure SetHandle(Value: HBrush);
```

C++

```
__fastcall SetHandle(HBrush Value);
```

Description

This is SetHandle, a member of class TBrush.

8.1.5.4.9 TBrush.SetStyle Method

Pascal

```
procedure SetStyle(Value: TBrushStyle);
```

C++

```
__fastcall SetStyle(TBrushStyle Value);
```

Description

This is SetStyle, a member of class TBrush.

8.1.5.5 TBrush Properties

The properties of the TBrush class are listed here.

Public Properties

	Name	Description
	Bitmap (see page 2668)	Specifies an external bitmap image that defines a pattern for the brush.
	Handle (see page 2670)	Provides access to the handle of a system brush.

Published Properties

	Name	Description
	Color (see page 2669)	Indicates the color of the brush.
	Style (see page 2670)	Specifies the pattern for the brush.

8.1.5.5.1 TBrush.Bitmap Property

Specifies an external bitmap image that defines a pattern for the brush.

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Bitmap points to a TBitmap (see page 2628) object that holds a bitmap image. If Bitmap is nonempty, the bitmap image (rather than the Style (see page 2670) property) defines the brush's pattern. If the image is larger than eight pixels by eight pixels, only the top left eight-by-eight region is used.

Changing the image does not affect the brush until the TBitmap (see page 2628) is reassigned to the Bitmap property. Be sure

to free the TBitmap (see page 2628) after finishing with the brush, since TBrush (see page 2664) will not free it.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> BrushBmp(new Graphics::TBitmap);
  BrushBmp->LoadFromFile("../bm1.BMP");
  Form1->Canvas->Brush->Bitmap = BrushBmp.get();
  Form1->Canvas->FillRect(Rect(0,0,100,100));
  Form1->Canvas->Brush->Bitmap = NULL;
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

TBitmap (see page 2628)

Style (see page 2670)

8.1.5.5.2 TBrush.Color Property

Indicates the color of the brush.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the color of the brush. This is the color that is used to draw the pattern represented by the Style (see page 2670) property, not the background color of the brush (unless Style (see page 2670) is bsSolid).

Note: If the value of the Style (see page 2670) property is bsClear, the Color property is ignored. Further, any value assigned to the Color property is lost when Style (see page 2670) is set to bsClear.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Style (see page 2670)

8.1.5.5.3 TBrush.Handle Property

Provides access to the handle of a system brush.

Pascal

```
property Handle: HBrush;
```

C++

```
__property HBrush Handle;
```

Description

The Handle property provides access to the Windows GDI object handle. TBrush (see page 2664) encapsulates the Windows brush object (HBRUSH) and is used to fill solid shapes, such as rectangles and ellipses, with a color or pattern.

8.1.5.5.4 TBrush.Style Property

Specifies the pattern for the brush.

Pascal

```
property Style: TBrushStyle;
```

C++

```
__property TBrushStyle Style;
```

Description

The Style property determines the pattern painted by the brush, unless a value is assigned to Bitmap (see page 2668).

Tip: Set the Style property to bsClear to eliminate flicker when the object repaints.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Bitmap (see page 2668)

Color (see page 2669)

8.1.6 TBrushRecall Class

TBrushRecall saves and restores a TBrush (see page 2664) object.

Class Hierarchy

Pascal

```
TBrushRecall = class(TRecall);
```

C++

```
class TBrushRecall : public TRecall;
```

File

Graphics

Description

Use TBrushRecall to store the current state of a TBrush (see page 2664) object. You are then free to change the current state of the brush object. To restore the brush object to its original state, all that is necessary is to free the TBrushRecall object and the brush object is automatically restored to the saved property values.

You can update the TBrushRecall instance to reflect the current properties of the TBrush (see page 2664) object by calling the Store method. You can prevent the TBrushRecall destructor from updating the TBrush (see page 2664) object by calling the Forget method.

See Also

TRecall

TBrush (see page 2664)

Using Brushes

8.1.6.1 TBrushRecall Members

The following tables list the members exposed by TBrushRecall.

Public Methods

	Name	Description
	Create (see page 2672)	Instantiates a TBrushRecall (see page 2671) object.

8.1.6.2 TBrushRecall.Create Constructor

Instantiates a TBrushRecall (see page 2671) object.

Pascal

```
constructor Create(ABrush: TBrush);
```

C++

```
__fastcall TBrushRecall(TBrush ABrush);
```

Description

Call Create to instantiate a brush recall object at runtime. Create saves the current properties of the TBrush (see page 2664) object specified by ABrush, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of ABrush and save them, replacing the values saved by this constructor. When you destroy the TBrushRecall (see page 2671) instance, the most recently saved properties are assigned to ABrush, restoring it to a previous state.

See Also

Destroy

Reference

Store

8.1.7 TCanvas Class

TCanvas provides an abstract drawing space for objects that must render their own images.

Class Hierarchy

Pascal

```
TCanvas = class(TPersistent);
```

C++

```
class TCanvas : public TPersistent;
```

File

Graphics

Description

Use TCanvas as a drawing surface for objects that draw an image of themselves. Standard window controls such as edit controls or list boxes do not require a canvas, as they are drawn by the system.

TCanvas provides properties, events and methods that assist in creating an image by

Specifying the type of brush, pen and font to use.

Drawing and filling a variety of shapes and lines.

Writing text.

Rendering graphic images.

Enabling a response to changes in the current image.

TCanvas has two descendants, TControlCanvas and TMetafileCanvas (see page 2794), which assist in drawing images of controls and in creating metafile images for objects.

See Also

TControlCanvas

TMetafileCanvas (see page 2794)

8.1.7.1 TCanvas Members

The following tables list the members exposed by TCanvas.

Protected Methods

	Name	Description
	Changed (see page 2681)	This is Changed, a member of class TCanvas.
	Changing (see page 2682)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 2685)	This is CreateHandle, a member of class TCanvas.
	RequiredState (see page 2702)	This is RequiredState, a member of class TCanvas.

Public Events

	Name	Description
	OnChange (see page 2729)	Occurs when the image has just changed.
	OnChanging (see page 2730)	Occurs just before a change is made to the image.

Public Methods

	Name	Description
	Arc (see page 2679)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	BrushCopy (see page 2680)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 2682)	Draws a closed figure represented by the intersection of a line and an ellipse.
	CopyRect (see page 2683)	Copies part of an image from another canvas into the canvas.
	Create (see page 2675)	Creates an instance of TCanvas (see page 2673).
	Destroy (see page 2678)	Destroys an instance of TCanvas (see page 2673).
	Draw (see page 2685)	This is the overview for the Draw method overload.
	DrawFocusRect (see page 2687)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 2688)	This is the overview for the Ellipse method overload.
	FillRect (see page 2689)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 2690)	Fills an area of the canvas using the current brush.
	FrameRect (see page 2692)	Draws a rectangle using the Brush (see page 2716) of the canvas to draw the border.
	HandleAllocated (see page 2693)	Indicates whether the TCanvas (see page 2673) object has acquired a handle to a device context.
	LineTo (see page 2693)	Draws a line on the canvas from PenPos (see page 2727) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 2694)	Prevents other threads from drawing on the canvas.
	MoveTo (see page 2695)	Changes the current drawing position to the point (X,Y).
	Pie (see page 2695)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 2696)	Draws a set of Bezier curves.
	PolyBezierTo (see page 2697)	Draws a set of Bezier curves and updates the value of PenPos (see page 2727).
	Polygon (see page 2697)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 2698)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 2700)	This is the overview for the Rectangle method overload.
	Refresh (see page 2701)	Deselects the Pen (see page 2725), Brush (see page 2716), and Font (see page 2721) from the device context.
	RoundRect (see page 2702)	This is the overview for the RoundRect method overload.
	StretchDraw (see page 2704)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 2705)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 2706)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 2707)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 2727) to the end of the string.
	TextRect (see page 2713)	This is the overview for the TextRect method overload.
	TextWidth (see page 2714)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 2715)	Locks the canvas if it is currently unlocked.
	Unlock (see page 2716)	Decreases the LockCount (see page 2725) property, allowing other threads to execute when LockCount (see page 2725) reaches 0.

Public Properties

	Name	Description
R	CanvasOrientation (see page 2718)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 2718)	Read-only property that specifies the boundaries of the clipping rectangle.

	Handle (see page 2722)	Specifies the handle for this canvas.
R	LockCount (see page 2725)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 2727)	Specifies the current drawing position of the Pen (see page 2725).
	Pixels (see page 2727)	Specifies the color of the pixels within the current ClipRect (see page 2718).
	TextFlags (see page 2728)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 2716)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 2719)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 2721)	Specifies the font to use when writing text on the image.
	Pen (see page 2725)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

8.1.7.2 TCanvas.Create Constructor

Creates an instance of TCanvas (see page 2673).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCanvas();
```

Description

Call Create to instantiate a TCanvas (see page 2673) object at runtime. Create allocates memory for the instance of TCanvas (see page 2673) and creates the TFont (see page 2730), TBrush (see page 2664) and TPen (see page 2801) objects for the Font (see page 2721), Brush (see page 2716), and Pen (see page 2725) properties. Create also initializes the CopyMode (see page 2719) to cmSrcCopy (see page 2972).

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/

#include <memory>       //for STL auto_ptr class

class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyControl::TMyControl(TComponent* Owner)
```

```
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
{
  std::auto_ptr<TCanvas> tempCanvas(new TCanvas);
  HWND notUsed;
  tempCanvas->Handle = GetDeviceContext(notUsed);
  std::auto_ptr<TImage> image2save(new TImage(Form1)); // The owner will clean this up.
  image2save->Height = Height;
  image2save->Width = Width;
  TRect destRect = Rect(0,0,Width,Height);
  TRect sourceRect = destRect;
  image2save->Canvas->CopyRect(destRect, tempCanvas.get(), sourceRect);
  LogPal SysPal;
  SysPal.lpal.palVersion = 0x300;
  SysPal.lpal.palNumEntries = 256;
  GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
  image2save->Picture->Bitmap->Palette =
    CreatePalette(dynamic_cast<const tagLOGPALETTE *>(&SysPal.lpal));
  image2save->Picture->SaveToFile(fileName);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static_cast<TMyControl*>(MyControl1)->SaveAsBmp("../foo.bmp");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TMyControl* control = new TMyControl(Form1); // The owner will clean this up.

  MyControl1 = control;
  control->Parent = this;
  control->Visible = true;
}
```

Delphi Examples:

```
{
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and then
double click on the newly created foo.bmp file to view.
}
type
  TMyControl = class(TColorListBox)
    procedure SaveAsBmp(fileName: TFileName);
  end;
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;
    MyControl1: TMyControl;
  implementation

  {$R *.dfm}

  type
    LogPal = record
    lpal : TLogPalette;
    dummy:Array[0..255] of TPaletteEntry;
    end;

  procedure TMyControl.SaveAsBmp(fileName: TFileName);
  var
    Source: TComponent;
    SysPal : LogPal;
    tempCanvas: TCanvas;
    sourceRect, destRect: TRect;
    image2save: TImage;
    notUsed: HWND;
  begin
    tempCanvas := TCanvas.Create;
    try
      tempCanvas.Handle := GetDeviceContext(notUsed);
      image2save:=TImage.create(self);
      try
        with image2save do
        begin
          Height := Self.Height;
          Width :=  Self.Width;
          destRect := Rect(0,0,Width,Height);
          sourceRect := destRect;
          Canvas.CopyRect(destRect,tempCanvas,sourceRect);
          SysPal.lPal.palVersion:=$300;
          SysPal.lPal.palNumEntries:=256;
          GetSystemPaletteEntries(
            tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
          Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
        end;
        image2save.Picture.SaveToFile(fileName);
      finally
       image2save.Free;
      end;
    finally
      tempCanvas.Free;
    end;
  end;
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    MyControl1.SaveAsBmp('foo.bmp');
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyControl1:= TMyControl.Create(Form1);
    MyControl1.Parent:= Form1;
    MyControl1.visible := true;
  end;
```

See Also

Brush (see page 2716)

CopyMode (see page 2719)

Font (see page 2721)

Pen (see page 2725)

TBrush (see page 2664)

TFont (see page 2730)

TPen (see page 2801)

8.1.7.3 TCanvas.Destroy Destructor

Destroys an instance of TCanvas (see page 2673).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCanvas();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCanvas (see page 2673) object is not nil and only then calls Destroy.

See Also

Brush (see page 2716)

Font (see page 2721)

System::TObject::Free

Pen (see page 2725)

TBrush (see page 2664)

TFont (see page 2730)

TPen (see page 2801)

8.1.7.4 TCanvas Methods

The methods of the TCanvas class are listed here.

Protected Methods

	Name	Description
	Changed (see page 2681)	This is Changed, a member of class TCanvas.
	Changing (see page 2682)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 2685)	This is CreateHandle, a member of class TCanvas.
	RequiredState (see page 2702)	This is RequiredState, a member of class TCanvas.

Public Methods

	Name	Description
	Arc (see page 2679)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	BrushCopy (see page 2680)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 2682)	Draws a closed figure represented by the intersection of a line and an ellipse.
	CopyRect (see page 2683)	Copies part of an image from another canvas into the canvas.
	Draw (see page 2685)	This is the overview for the Draw method overload.

	DrawFocusRect (see page 2687)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 2688)	This is the overview for the Ellipse method overload.
	FillRect (see page 2689)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 2690)	Fills an area of the canvas using the current brush.
	FrameRect (see page 2692)	Draws a rectangle using the Brush (see page 2716) of the canvas to draw the border.
	HandleAllocated (see page 2693)	Indicates whether the TCanvas (see page 2673) object has acquired a handle to a device context.
	LineTo (see page 2693)	Draws a line on the canvas from PenPos (see page 2727) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 2694)	Prevents other threads from drawing on the canvas.
	MoveTo (see page 2695)	Changes the current drawing position to the point (X,Y).
	Pie (see page 2695)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 2696)	Draws a set of Bezier curves.
	PolyBezierTo (see page 2697)	Draws a set of Bezier curves and updates the value of PenPos (see page 2727).
	Polygon (see page 2697)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 2698)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 2700)	This is the overview for the Rectangle method overload.
	Refresh (see page 2701)	Deselects the Pen (see page 2725), Brush (see page 2716), and Font (see page 2721) from the device context.
	RoundRect (see page 2702)	This is the overview for the RoundRect method overload.
	StretchDraw (see page 2704)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 2705)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 2706)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 2707)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 2727) to the end of the string.
	TextRect (see page 2713)	This is the overview for the TextRect method overload.
	TextWidth (see page 2714)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 2715)	Locks the canvas if it is currently unlocked.
	Unlock (see page 2716)	Decreases the LockCount (see page 2725) property, allowing other threads to execute when LockCount (see page 2725) reaches 0.

8.1.7.4.1 TCanvas.Arc Method

Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.

Pascal

```
procedure Arc(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Arc(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Arc to draw an elliptically curved line with the current Pen (see page 2725). The arc traverses the perimeter of an ellipse that is bounded by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse, counterclockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X3,Y3). The ending point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X4, Y4).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

On NT or Windows 2000, the drawing direction can be changed to clockwise using the Windows API call SetArcDirection.

C++ Examples:

```
/*
The following lines of code draw the top quarter of an arc
bounded by the current window:
*/
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TRect R = GetClientRect();  // Gets the rectangular
                              // coordinates of
                              // the current window.
  Canvas->Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
}
```

Delphi Examples:

```
{
The following lines of code draw the top quarter of an arc
bounded by the current window:
}
procedure TForm1.FormPaint(Sender: TObject);
var
  R: TRect;
begin
  R := GetClientRect;   { Gets the boundaries of the current window }
  Canvas.Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
end;
```

See Also

Ellipse (see page 2688)

Pen (see page 2725)

8.1.7.4.2 TCanvas.BrushCopy Method

Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.

Pascal

```
procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);
```

C++

```
__fastcall BrushCopy(const TRect Dest, TBitmap Bitmap, const TRect Source, TColor Color);
```

Description

Use BrushCopy to achieve special effects such as making the copied image partially transparent. BrushCopy is provided mainly for backward compatibility. Use a TImageList instead of BrushCopy.

Dest specifies the rectangular portion of the canvas that will receive the copy of the bitmap. Bitmap specifies the graphic to copy from. Source specifies the rectangular area of Bitmap to copy. Color specifies the color in Bitmap to replace with the Brush (see page 2716) of the canvas.

To use BrushCopy to make the copied image partially transparent, specify the color of the surface of the canvas (clBackground (see page 2899) for example) as the Color of the Brush (see page 2716) property, then call BrushCopy.

C++ Examples:

```
/*
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
```

```
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect MyRect = Rect(10,10,100,100);
  TRect MyOther = Rect(10,111,100, 201);
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap);
  Bitmap->LoadFromFile("c:/Program Files/Common Files/CodeGear
Shared/Images/Splash/256color/factory.bmp");
  Form1->Canvas->BrushCopy(MyRect, Bitmap.get(), MyRect, clBlack);
  Form1->Canvas->CopyRect(MyOther, Bitmap->Canvas, MyRect);
}
```

Delphi Examples:

```
{
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap: TBitmap;
  MyRect, MyOther: TRect;
begin
  MyRect := Rect(10,10,150,150);
  MyOther := Rect(10,161,150,301);
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile(
    'c:\Program Files\Common Files\CodeGear Shared\Images\Splash\256color\factory.bmp');
  Form1.Canvas.BrushCopy(MyRect, Bitmap, MyRect, clBlack);
  Form1.Canvas.CopyRect(MyOther,Bitmap.Canvas,MyRect);
  Bitmap.Free;
end;
```

See Also

Brush (see page 2716)

CopyMode (see page 2719)

StretchDraw (see page 2704)

TBitmap (see page 2628)

Color (see page 2669)

TImageList

8.1.7.4.3 TCanvas.Changed Method

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TCanvas.

8.1.7.4.4 TCanvas.Changing Method

Pascal

```
procedure Changing; virtual;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TCanvas.

8.1.7.4.5 TCanvas.Chord Method

Draws a closed figure represented by the intersection of a line and an ellipse.

Pascal

```
procedure Chord(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Chord(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Chord to create a shape that is defined by an arc and a line that joins the endpoints of the arc. The chord consists of a portion of an ellipse that is bounded by the points (X1,Y1) and (X2,Y2). The ellipse is bisected by a line that runs between the points (X3,Y3) and (X4,Y4).

The perimeter of the chord runs counter clockwise from (X3, Y3), counterclockwise along the ellipse to (X4,Y4), and straight back to (X3,Y3). If (X3,Y3) and (X4,Y4) are not on the surface of the ellipse, the corresponding corners on the chord are the closest points on the perimeter that intersect the line. The outline of the chord is drawn using the value of Pen (see page 2725), and the shape is filled using the value of Brush (see page 2716).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

On NT, the drawing direction can be changed to clockwise using the Windows API call SetArcDirection.

C++ Examples:

```
/*
This code draws a chord on the top of an ellipse bounded by
the current window.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect R = GetClientRect();  //Get coordinates of the current window.
  Canvas->Chord(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
}
```

Delphi Examples:

```
{
This code draws a chord on the top of an ellipse bounded by
the current window.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R: TRect;
begin
  R := GetClientRect;  { Get coordinates of the current window. }
  Canvas.Chord(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
end;
```

See Also

Arc (see page 2679)

Brush (see page 2716)

Ellipse (see page 2688)

Pen (see page 2725)

8.1.7.4.6 TCanvas.CopyRect Method

Copies part of an image from another canvas into the canvas.

Pascal

```
procedure CopyRect(const Dest: TRect; Canvas: TCanvas; const Source: TRect);
```

C++

```
__fastcall CopyRect(const TRect Dest, TCanvas Canvas, const TRect Source);
```

Description

Use CopyRect to transfer part of the image on another canvas to the image of the TCanvas (see page 2673) object. Dest specifies the rectangle on the canvas where the source image will be copied. The Canvas parameter specifies the canvas with the source image. Source specifies a rectangle bounding the portion of the source canvas that will be copied.

The portion of the source canvas is copied using the mode specified by CopyMode (see page 2719).

C++ Examples:

```
/*
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect MyRect = Rect(10,10,100,100);
  TRect MyOther = Rect(10,111,100, 201);
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap);
  Bitmap->LoadFromFile("c:/Program Files/Common Files/CodeGear
Shared/Images/Splash/256color/factory.bmp");
  Form1->Canvas->BrushCopy(MyRect, Bitmap.get(), MyRect, clBlack);
  Form1->Canvas->CopyRect(MyOther, Bitmap->Canvas, MyRect);
}
/*
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
```

```
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    TRect SrcRect, DstRect;
    Image2->Canvas->CopyMode = cmSrcCopy;
    DstRect = Rect(0, 0, Image2->Width, Image2->Height);
    SrcRect = Rect(0, 0, Image1->Width, Image1->Height);
    Image2->Canvas->CopyRect(DstRect, Image1->Canvas, SrcRect);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
 TRect r;
 Copy1Click(Sender); // Do the same thing as the copy menu item.
 Image1->Canvas->CopyMode = cmWhiteness;
 r = Rect(0, 0, Image1->Width, Image1->Height);
 Image1->Canvas->CopyRect(r, Image1->Canvas, r);
 Image1->Canvas->CopyMode = cmSrcCopy; // Restore the copy mode.
}
```

Delphi Examples:

```
{
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap: TBitmap;
  MyRect, MyOther: TRect;
begin
  MyRect := Rect(10,10,150,150);
  MyOther := Rect(10,161,150,301);
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile(
    'c:\Program Files\Common Files\CodeGear Shared\Images\Splash\256color\factory.bmp');
  Form1.Canvas.BrushCopy(MyRect, Bitmap, MyRect, clBlack);
  Form1.Canvas.CopyRect(MyOther,Bitmap.Canvas,MyRect);
  Bitmap.Free;
end;
{
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
}
procedure TForm1.Copy1Click(Sender: TObject);
var
  DstRect, SrcRect: TRect;
begin
  with Image2.Canvas do
  begin
    CopyMode := cmSrcCopy;
    DstRect := Rect(0, 0, Image2.Width, Image2.Height);
    SrcRect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(DstRect, Image1.Canvas, SrcRect);
  end;
end;

procedure TForm1.Cut1Click(Sender: TObject);
var
  ARect: TRect;
```

```
begin
  Copy1Click(Sender); { do the same thing as the copy menu item }
  with Image1.Canvas do
  begin
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(ARect, Image1.Canvas, ARect);
    CopyMode := cmSrcCopy; { restore the copy mode }
  end;
end;
```

See Also

BrushCopy (see page 2680)

CopyMode (see page 2719)

StretchDraw (see page 2704)

8.1.7.4.7 TCanvas.CreateHandle Method

Pascal

```
procedure CreateHandle; virtual;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

This is CreateHandle, a member of class TCanvas.

8.1.7.4.8 Draw Method

This is the overview for the Draw method overload.

Overload List

	Name	Description
	TCanvas.Draw (Integer, Integer, TGraphic) (see page 2685)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).
	TCanvas.Draw (Integer, Integer, TGraphic, Byte) (see page 2685)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).

8.1.7.4.8.1 TCanvas.Draw Method (Integer, Integer, TGraphic)

Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).

Pascal

```
procedure Draw(X: Integer; Y: Integer; Graphic: TGraphic); overload;
procedure Draw(X: Integer; Y: Integer; Graphic: TGraphic; Opacity: Byte); overload;
```

C++

```
__fastcall Draw(int X, int Y, TGraphic Graphic);
__fastcall Draw(int X, int Y, TGraphic Graphic, Byte Opacity);
```

Description

Call Draw to draw a graphic on the canvas. Draw calls the Draw method of the graphic. The image is rendered into a rectangle determined by the size of the graphic, with the upper left corner at the point (X, Y).

Graphics (see page 2619) can be bitmaps, icons, or metafiles. If the graphic is a TBitmap (see page 2628) object, the bitmap is rendered using the value of CopyMode (see page 2719).

C++ Examples:

```
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
/*
An examples of TBitmap Color, TGraphic Transparent, TBitmap
TransparentMode, TBitmap TransparentColor and TCanvas Draw.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bitmap(new Graphics::TBitmap);
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap.get());
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
}
```

Delphi Examples:

```
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
```

```
end;
```

See Also

BrushCopy (see page 2680)

CopyMode (see page 2719)

CopyRect (see page 2683)

StretchDraw (see page 2704)

8.1.7.4.9 TCanvas.DrawFocusRect Method

Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.

Pascal

```
procedure DrawFocusRect(const Rect: TRect);
```

C++

```
__fastcall DrawFocusRect(const TRect Rect);
```

Description

Call DrawFocusRect for the perimeter of the image of a control when the control gets or loses input focus. Because DrawFocusRect uses an XOR function, calling it a second time while specifying the same rectangle removes the rectangle from the screen.

The rectangle this function draws cannot be scrolled. To scroll an area containing a rectangle drawn by DrawFocusRect, call DrawFocusRect to remove the rectangle from the screen, scroll the area, and then call DrawFocusRect to draw the rectangle in the new position.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ActiveControl (see page 2145)

ActiveControl (see page 2452)

8.1.7.4.10 Ellipse Method

This is the overview for the Ellipse method overload.

Overload List

	Name	Description
	TCanvas.Ellipse (Integer, Integer, Integer, Integer) (see page 2688)	Draws the ellipse defined by a bounding rectangle on the canvas.
	TCanvas.Ellipse (TRect) (see page 2688)	Draws the ellipse defined by a bounding rectangle on the canvas.

8.1.7.4.10.1 TCanvas.Ellipse Method (Integer, Integer, Integer, Integer)

Draws the ellipse defined by a bounding rectangle on the canvas.

Pascal

```
procedure Ellipse(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer); overload;
procedure Ellipse(const Rect: TRect); overload;
```

C++

```
__fastcall Ellipse(int X1, int Y1, int X2, int Y2);
```

```
__fastcall Ellipse(const TRect Rect);
```

Description

Call Ellipse to draw a circle or ellipse on the canvas. Specify the bounding rectangle either by giving

The top left point at pixel coordinates (X1, Y1) and the bottom right point at (X2, Y2).

A TRect value.

If the bounding rectangle is a square, a circle is drawn.

The ellipse is outlined using the value of Pen (see page 2725), and filled using the value of Brush (see page 2716).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Arc (see page 2679)

Brush (see page 2716)

Chord (see page 2682)

Pen (see page 2725)

8.1.7.4.11 TCanvas.FillRect Method

Fills the specified rectangle on the canvas using the current brush.

Pascal

```
procedure FillRect(const Rect: TRect);
```

C++

```
__fastcall FillRect(const TRect Rect);
```

Description

Use FillRect to fill a rectangular region using the current brush. The region is filled including the top and left sides of the rectangle, but excluding the bottom and right edges.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> BrushBmp(new Graphics::TBitmap);
  BrushBmp->LoadFromFile("../bm1.BMP");
  Form1->Canvas->Brush->Bitmap = BrushBmp.get();
  Form1->Canvas->FillRect(Rect(0,0,100,100));
  Form1->Canvas->Brush->Bitmap = NULL;
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

FloodFill (see page 2690)

8.1.7.4.12 TCanvas.FloodFill Method

Fills an area of the canvas using the current brush.

Pascal

```
procedure FloodFill(X: Integer; Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

C++

```
__fastcall FloodFill(int X, int Y, TColor Color, TFillStyle FillStyle);
```

Description

Use FloodFill to fill a possibly non-rectangular region of the image with the value of Brush (see page 2716). The boundaries of the region to be filled are determined by moving outward from the point (X,Y) until a color boundary involving the Color parameter is encountered.

X and X are the coordinates on the canvas where filling starts.

Color is the color that defines the boundary of the region to fill. Its interpretation depends on the value of FillStyle.

FillStyle specifies whether the region is defined by all pixels with the same value as Color, or all points with a different value.

Tip: Use the Pixels (see page 2727) property to get the exact value of the color at the point (X,Y) when using a FillStyle of fsSurface. Similarly, when FillStyle is fsBorder, use Pixels (see page 2727) to get the exact value of the boundary color if a point on the boundary is known.

C++ Examples:

```
/*
The following code floodfills from the center point of
Form1's client area until the color black is encountered:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Canvas->Brush->Color = clBlue;
  Canvas->Brush->Style = bsDiagCross;
  Canvas->FloodFill(
    ClientWidth/2, ClientHeight/2, clBlack, fsBorder);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int ewidth = 400;
  int eheight = 200;
  Canvas->Brush->Color = clRed;
  Canvas->Pen->Color = clBlack;
  Canvas->Ellipse(
    (ClientWidth - ewidth)/2,
    (ClientHeight - eheight)/2,
    ewidth, eheight);
}
```

Delphi Examples:

```
{
The following code floodfills from the center point of
Form1's client area until the color black is encountered:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.Brush.Color := clBlue;
  Canvas.Brush.Style := bsDiagCross;
  Canvas.FloodFill(
    ClientWidth div 2, ClientHeight div 2, clBlack, fsBorder);
end;

procedure TForm1.Button2Click(Sender: TObject);
var ewidth, eheight: integer;
begin
  ewidth := 400;
  eheight := 200;
  Canvas.Brush.Color := clRed;
  Canvas.Pen.Color := clBlack;
  Canvas.Ellipse(
```

```
    (ClientWidth - ewidth) div 2,
    (ClientHeight - eheight) div 2,
    ewidth, eheight);
end;
```

See Also

FillRect (see page 2689)

Pixels (see page 2727)

8.1.7.4.13 TCanvas.FrameRect Method

Draws a rectangle using the Brush (see page 2716) of the canvas to draw the border.

Pascal

```
procedure FrameRect(const Rect: TRect);
```

C++

```
__fastcall FrameRect(const TRect Rect);
```

Description

Use FrameRect to draw a 1 pixel wide border around a rectangular region. FrameRect does not fill the interior of the rectangle with the Brush (see page 2716) pattern.

To draw a boundary using the Pen (see page 2725) instead, use the Polygon (see page 2697) method.

C++ Examples:

```
/*
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect TheRect = Rect(10,10,100,100);
  Canvas->TextRect(TheRect, 10, 10, "Hello World");
  Canvas->Brush->Color = clBlack;
  Canvas->FrameRect(TheRect);
}
```

Delphi Examples:

```
{
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
}
procedure TForm1.Button1Click(Sender: TObject);
var TheRect: TRect;
begin
  TheRect := Rect(10,10,100,100);
  Form1.Canvas.TextRect(TheRect,10,10,'Hello, world!');
  Form1.Canvas.Brush.Color := clBlack;
  Form1.Canvas.FrameRect(TheRect);
end;
```

See Also

Brush (see page 2716)

FillRect (see page 2689)

Pen (see page 2725)

Polygon (see page 2697)

Rectangle (see page 2700)

RoundRect (see page 2702)

8.1.7.4.14 TCanvas.HandleAllocated Method

Indicates whether the TCanvas (see page 2673) object has acquired a handle to a device context.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 2722) property is set to an HDC value. HandleAllocated indicates whether the canvas has acquired a device context. Use HandleAllocated rather than reading the Handle (see page 2722) property. When an application reads the Handle (see page 2722) property, TCanvas (see page 2673) automatically allocates a new device context and returns its handle if the property had not been previously set.

See Also

Handle (see page 2722)

8.1.7.4.15 TCanvas.LineTo Method

Draws a line on the canvas from PenPos (see page 2727) to the point specified by X and Y, and sets the pen position to (X, Y).

Pascal

```
procedure LineTo(X: Integer; Y: Integer);
```

C++

```
__fastcall LineTo(int X, int Y);
```

Description

Use LineTo to draw a line from PenPos (see page 2727) up to, but not including the point (X,Y). LineTo changes the value of PenPos (see page 2727) to (X,Y).

The line is drawn using Pen (see page 2725).

Note: If the current Pen (see page 2725) does not have a style of psSolid, the line is drawn with a background specified by the current brush.

C++ Examples:

```
/*
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
*/
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  /* first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) */
```

```
  Canvas->FillRect(ClientRect);
  Canvas->MoveTo(0, 0);
  Canvas->LineTo(X, Y);
}
```

Delphi Examples:

```
{
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
}
procedure TForm1.FormMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  { first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) }
  Canvas.FillRect(ClientRect);
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

See Also

Pen (see page 2725)

PenPos (see page 2727)

Polyline (see page 2698)

Style (see page 2808)

Brush (see page 2716)

8.1.7.4.16 TCanvas.Lock Method

Prevents other threads from drawing on the canvas.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock in a multithreaded application to prevent other threads from drawing on the canvas. Lock prevents other cooperating threads in the application from executing until the Unlock (see page 2716) method is called. Nested calls to Lock increment the LockCount (see page 2725) property so that the canvas is not unlocked until the last lock is released.

In multi-threaded applications that use Lock to protect a canvas, all calls that use the canvas must be protected by a call to Lock. Any thread that does not lock the canvas before using it will introduce potential bugs.

Because Lock prevents other threads from executing, it can adversely affect performance. Do not call Lock unless there is a danger that another thread might interfere with drawing on the canvas.

See Also

LockCount (see page 2725)

TryLock (see page 2715)

Unlock (see page 2716)

Locking Objects

8.1.7.4.17 TCanvas.MoveTo Method

Changes the current drawing position to the point (X,Y).

Pascal

```
procedure MoveTo(X: Integer; Y: Integer);
```

C++

```
__fastcall MoveTo(int X, int Y);
```

Description

Use MoveTo to set the value of PenPos (see page 2727) before calling LineTo (see page 2693). Calling MoveTo is equivalent to setting the PenPos (see page 2727) property.

C++ Examples:

```
/*
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
*/
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  /* first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) */
  Canvas->FillRect(ClientRect);
  Canvas->MoveTo(0, 0);
  Canvas->LineTo(X, Y);
}
```

Delphi Examples:

```
{
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
}
procedure TForm1.FormMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  { first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) }
  Canvas.FillRect(ClientRect);
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

See Also

LineTo (see page 2693)

PenPos (see page 2727)

8.1.7.4.18 TCanvas.Pie Method

Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.

Pascal

```
procedure Pie(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Pie(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Pie to draw a pie-shaped wedge on the image. The wedge is defined by the ellipse bounded by the rectangle determined by the points (X1, Y1) and X2, Y2). The section drawn is determined by two lines radiating from the center of the ellipse through the points (X3, Y3) and (X4, Y4).

The wedge is outlined using Pen (see page 2725), and filled using Brush (see page 2716).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

See Also

Arc (see page 2679)

Brush (see page 2716)

Chord (see page 2682)

Pen (see page 2725)

8.1.7.4.19 TCanvas.PolyBezier Method

Draws a set of Bezier curves.

Pascal

```
procedure PolyBezier(const Points: array of TPoint);
```

C++

```
__fastcall PolyBezier(const array of TPoint Points);
```

Description

Use PolyBezier to draw cubic Bezier curves using the endpoints and control points specified by the Points parameter. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The Points parameter gives the endpoints to use when generating the Bezier curves.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

This procedure draws lines by using the current pen.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the PolyBezier method. For example, to form a Bezier curve using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.PolyBezier(Slice(PointArray, 10));
```

Control points after a[Index+3] are ignored. Nothing happens if there aren't enough control points.

See Also

Polyline (see page 2698)

PolyBezierTo (see page 2697)

Pen (see page 2725)

8.1.7.4.20 TCanvas.PolyBezierTo Method

Draws a set of Bezier curves and updates the value of PenPos (see page 2727).

Pascal

```
procedure PolyBezierTo(const Points: array of TPoint);
```

C++

```
__fastcall PolyBezierTo(const array of TPoint Points);
```

Description

Use PolyBezierTo to draw cubic Bezier curves using the endpoints and control points specified by the Points parameter. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The Points parameter gives the endpoints to use when generating the Bezier curves.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

This procedure draws lines by using the current pen and updates the value of PenPos (see page 2727) to the value of the last endpoint.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the PolyBezierTo method. For example, to form a Bezier curve using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.PolyBezierTo(Slice(PointArray, 10));
```

See Also

Polyline (see page 2698)

PolyBezier (see page 2696)

Pen (see page 2725)

PenPos (see page 2727)

8.1.7.4.21 TCanvas.Polygon Method

Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.

Pascal

```
procedure Polygon(const Points: array of TPoint);
```

C++

```
__fastcall Polygon(const array of TPoint Points);
```

Description

Use Polygon to draw a closed, many-sided shape on the canvas, using the value of Pen (see page 2725). After drawing the complete shape, Polygon fills the shape using the value of Brush (see page 2716).

The Points parameter is an array of points that give the vertices of the polygon.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

The first point is always connected to the last point.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the Polygon method. For example, to form a polygon using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.Polygon(Slice(PointArray, 10));
```

To draw a polygon on the canvas, without filling it, use the Polyline (see page 2698) method, specifying the first point a second time at the end.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Brush->Color = clTeal;
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Polygon(points, 3);
}
```

Delphi Examples:

```
{
This example draws a polygon in the specified shape, and
fills it with the color teal.
}
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Brush.Color := clTeal;
  Canvas.Polygon([Point(10, 10), Point(30, 10),
    Point(130, 30), Point(240, 120)]);
end;
```

See Also

Brush (see page 2716)

Pen (see page 2725)

Polyline (see page 2698)

Slice

8.1.7.4.22 TCanvas.Polyline Method

Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.

Pascal

```
procedure Polyline(const Points: array of TPoint);
```

C++

```
__fastcall Polyline(const array of TPoint Points);
```

Description

Use Polyline to connect a set of points on the canvas. If you specify only two points, Polyline draws a single line.

The Points parameter is an array of points to be connected.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the Polyline method. For example, to form a line connecting the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.Polyline(Slice(PointArray, 10));
```

Calling the MoveTo (see page 2695) function with the value of the first point, and then repeatedly calling LineTo (see page 2693) with all subsequent points will draw the same image on the canvas. However, unlike LineTo (see page 2693), Polyline does not change the value of PenPos (see page 2727).

C++ Examples:

```
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = dynamic_cast<TPaintBox *>(Sender);
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
This example paints a white five-pointed star in a paintbox
control:
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TPaintBox do
  begin
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
    Point(10, 30), Point(60, 60), Point(40, 10)]);
  end;
end;
```

See Also

LineTo (see page 2693)

MoveTo (see page 2695)

Pen (see page 2725)

PenPos (see page 2727)

Polygon (see page 2697)

Slice

8.1.7.4.23 Rectangle Method

This is the overview for the Rectangle method overload.

Overload List

	Name	Description
	TCanvas.Rectangle (Integer, Integer, Integer, Integer) (see page 2700)	Draws a rectangle on the canvas.
	TCanvas.Rectangle (TRect) (see page 2700)	Draws a rectangle on the canvas.

8.1.7.4.23.1 TCanvas.Rectangle Method (Integer, Integer, Integer, Integer)

Draws a rectangle on the canvas.

Pascal

```
procedure Rectangle(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer); overload;
procedure Rectangle(const Rect: TRect); overload;
```

C++

```
__fastcall Rectangle(int X1, int Y1, int X2, int Y2);
__fastcall Rectangle(const TRect Rect);
```

Description

Use Rectangle to draw a rectangle using Pen (see page 2725) and fill it with Brush (see page 2716). Specify the rectangle's coordinates in one of two ways:

Giving four coordinates that define the upper left corner at the point (X1, Y1) and the lower right corner at the point (X2, Y2).

Using a TRect type.

To fill a rectangular region without drawing the boundary in the current pen, use FillRect (see page 2689). To outline a rectangular region without filling it, use FrameRect (see page 2692) or Polygon (see page 2697). To draw a rectangle with rounded corners, use RoundRect (see page 2702).

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
```

```
    y = random(Screen->Height - 10);
    Canvas->Pen->Color = (Graphics::TColor) random(65535);
    switch (random(5))
    {
      case 0: Canvas->Pen->Style = psSolid; break;
      case 1: Canvas->Pen->Style = psDash; break;
      case 2: Canvas->Pen->Style = psDot; break;
      case 3: Canvas->Pen->Style = psDashDot; break;
      case 4: Canvas->Pen->Style = psDashDotDot; break;
    }
    Canvas->Rectangle(x, y, x + random(400), y + random(400));
  }
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

Brush (see page 2716)

FillRect (see page 2689)

FrameRect (see page 2692)

Pen (see page 2725)

Polygon (see page 2697)

RoundRect (see page 2702)

8.1.7.4.24 TCanvas.Refresh Method

Deselects the Pen (see page 2725), Brush (see page 2716), and Font (see page 2721) from the device context.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to return the HDC used by the canvas to a default state. Refresh does not change the values of the Font (see page 2721), Brush (see page 2716), or Pen (see page 2725) properties. Calling any of the drawing methods of the canvas after calling Refresh will reselect the Font (see page 2721), Brush (see page 2716), or Pen (see page 2725) into the HDC if it is used by the drawing method.

See Also

Brush (see page 2716)

Font (see page 2721)

Handle (see page 2722)

Pen (see page 2725)

8.1.7.4.25 TCanvas.RequiredState Method

Pascal

```
procedure RequiredState(ReqState: TCanvasState);
```

C++

```
__fastcall RequiredState(TCanvasState ReqState);
```

Description

This is RequiredState, a member of class TCanvas.

8.1.7.4.26 RoundRect Method

This is the overview for the RoundRect method overload.

Overload List

	Name	Description
	TCanvas.RoundRect (Integer, Integer, Integer, Integer, Integer, Integer) (see page 2702)	Draws a rectangle with rounded corners on the canvas.
	TCanvas.RoundRect (TRect, Integer, Integer) (see page 2702)	Draws a rectangle with rounded corners on the canvas.

8.1.7.4.26.1 TCanvas.RoundRect Method (Integer, Integer, Integer, Integer, Integer, Integer)

Draws a rectangle with rounded corners on the canvas.

Pascal

```
procedure RoundRect(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3:
Integer); overload;
procedure RoundRect(const Rect: TRect; CX: Integer; CY: Integer); overload;
```

C++

```
__fastcall RoundRect(int X1, int Y1, int X2, int Y2, int X3, int Y3);
__fastcall RoundRect(const TRect Rect, int CX, int CY);
```

Description

Use RoundRect to draw a rounded rectangle using Pen (see page 2725) and fill it with Brush (see page 2716). The rectangle will have edges defined by the points (X1,Y1), (X2,Y1), (X2,Y2), (X1,Y2), but the corners will be shaved to create a rounded appearance. The curve of the rounded corners matches the curvature of an ellipse with width X3 and height Y3.

To draw an ellipse instead, use Ellipse (see page 2688). To draw a true rectangle, use Rectangle (see page 2700).

C++ Examples:

```
/*
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  Canvas->Pen->Width = random(7);
  int Dx = random(400);
  int Dy = random(400);
  Canvas->RoundRect(x, y, x + Dx, y + Dy, Dx/2, Dy/2);
}
```

Delphi Examples:

```
{
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
}
var
  X, Y, DX, DY: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  Canvas.Pen.Width := Random(7);
  DX := Random(400);
  DY := Random(400);
  Canvas.RoundRect(X, Y, X + DX, Y + DY, DX div 2, DY div 2);
end;
```

See Also

Brush (see page 2716)

Ellipse (see page 2688)

Pen (see page 2725)

Rectangle (see page 2700)

8.1.7.4.27 TCanvas.StretchDraw Method

Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.

Pascal

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

C++

```
__fastcall StretchDraw(const TRect Rect, TGraphic Graphic);
```

Description

Call StretchDraw to draw a graphic on the canvas so that the image fits in the specified rectangle. StretchDraw calls the Draw (see page 2685) method of the graphic. The graphic object determines how to fit into the rectangle. This may involve changing magnification and/or aspect ratio.

To render the graphic in its natural size, use the Draw (see page 2685) method, instead.

If the graphic is a TBitmap (see page 2628) object, the bitmap is rendered using the value of CopyMode (see page 2719).

Note: If the graphic is an icon, it is not stretched.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
```

```
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

Draw (see page 2685)

8.1.7.4.28 TCanvas.TextExtent Method

Returns the width and height, in pixels, of a string rendered in the current font.

Pascal

```
function TextExtent(const Text: string): TSize;
```

C++

```
__fastcall TSize TextExtent(const UnicodeString Text);
```

Description

Use TextExtent to determine the space a string will occupy in the image. Other elements in the image such as lines, boxes, or additional lines of text can be positioned to accommodate the size of the text. TextExtent returns the width and height of the rectangle that bounds the text on the canvas.

To check only the height, call TextHeight (see page 2706). To check only the width, call TextWidth (see page 2714).

See Also

Font (see page 2721)

TextHeight (see page 2706)

TextOut (see page 2707)

TextWidth (see page 2714)

8.1.7.4.29 TCanvas.TextHeight Method

Returns the height, in pixels, of a string rendered in the current font.

Pascal

```
function TextHeight(const Text: string): Integer;
```

C++

```
__fastcall int TextHeight(const UnicodeString Text);
```

Description

Use TextHeight to determine the height a string will occupy in the image. Other elements in the image such as lines, boxes, or additional lines of text can be positioned to accommodate the height of the text. TextHeight returns the height of the font, the ascenders plus the descenders, not the actual height of the specified string. If Font.Height is negative, then TextHeight is equal to abs(Font.Height) plus the internal leading of the font. If Font.Height is positive, then TextHeight equals Font.Height.

TextHeight returns the same value as TextExtent (see page 2705)(Text).cy.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
```

```
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

Font (see page 2721)

TextExtent (see page 2705)

TextOut (see page 2707)

TextWidth (see page 2714)

8.1.7.4.30 TCanvas.TextOut Method

Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 2727) to the end of the string.

Pascal

```
procedure TextOut(X: Integer; Y: Integer; const Text: string);
```

C++

```
__fastcall TextOut(int X, int Y, const UnicodeString Text);
```

Description

Use TextOut to write a string onto the canvas. The string will be written using the current value of Font (see page 2721). Use the TextExtent (see page 2705) method to determine the space occupied by the text in the image. To write only the text that fits within a clipping rectangle, use TextRect (see page 2713) instead.

After a call to TextOut, the PenPos (see page 2727) property indicates the point at the top right of the text on the canvas.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
```

```
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
```

```
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
/*
The following code adds an object to MyList if it isn't
already in the list.
*/

#include <memory>       //for STL auto_ptr class

typedef struct AList
{
  int I;
  char C;
} TAList;

typedef AList* PAList;

TList *MyList;
PAList ARecord1, ARecord2;

void __fastcall DisplayTList(TList *theList)
{
  PAList AStruct;
  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < theList->Count; i++)
  {
    AStruct = (PAList) theList->Items[i];
    Form1->PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    Form1->PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord1) == -1) MyList->Add(ARecord1);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord2) == -1) MyList->Add(ARecord2);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (MyList->Count != 0)
  {
    for (int B = MyList->Count - 1; B >= 0; B--)
    {
```

```
      PAList ARecord;
      ARecord = PAList(MyList->Items[B]);
      MyList->Remove(ARecord);
    }
  }
  Form1->PaintBox1->Repaint();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TList> _MyListCleaner(MyList = new TList);
  static std::auto_ptr<TAList> _ARecord1Cleaner(ARecord1 = new TAList);
  ARecord1->I = 100;
  ARecord1->C = 'Z';
  static std::auto_ptr<TAList> _ARecord2Cleaner(ARecord2 = new TAList);
  ARecord2->I = 200;
  ARecord2->C = 'X';
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
```

```
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
  ARecord: PMyList;
  B: Byte;
```

```
  Y: Word;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      ARecord := TheList.Items[B];
      Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
  DisplayTList(MyList);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
  DisplayTList(MyList);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  ARecord: PMyList;
  B: Integer;
begin
  if (MyList.Count <> 0) then
  begin
    for B := (MyList.Count - 1) downto 0 do
    begin
      ARecord := MyList.Items[B];
      MyList.Remove(ARecord);
    end;
  end;
  Form1.PaintBox1.Repaint;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TList.Create;
  New(ARecord1);
  ARecord1^.I := 100;
  ARecord1^.C := 'Z';
  New(ARecord2);
  ARecord2^.I := 200;
  ARecord2^.C := 'X';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(ARecord1);
  Dispose(ARecord2);
  MyList.Free;
end;
```

See Also

Font (see page 2721)

TextExtent (see page 2705)

TextRect (see page 2713)

TextFlags (see page 2728)

PenPos (see page 2727)

8.1.7.4.31 TextRect Method

This is the overview for the TextRect method overload.

Overload List

	Name	Description
	TCanvas.TextRect (TRect, Integer, Integer, string) (see page 2713)	Writes a string inside a clipping rectangle.
	TCanvas.TextRect (TRect, string, TTextFormat) (see page 2713)	Writes a string inside a clipping rectangle.

8.1.7.4.31.1 TCanvas.TextRect Method (TRect, Integer, Integer, string)

Writes a string inside a clipping rectangle.

Pascal

```
procedure TextRect(var Rect: TRect; var Text: string; TextFormat: TTextFormat = []); overload;
procedure TextRect(Rect: TRect; X: Integer; Y: Integer; const Text: string); overload;
```

C++

```
__fastcall TextRect(TRect Rect, UnicodeString Text, TTextFormat TextFormat = []);
__fastcall TextRect(TRect Rect, int X, int Y, const UnicodeString Text);
```

Description

Use TextRect to write a string within a limited rectangular region. Any portions of the string that fall outside the rectangle passed in the Rect parameter are clipped and don't appear. The upper left corner of the text is placed at the point (X, Y).

C++ Examples:

```
/*
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect TheRect = Rect(10,10,100,100);
  Canvas->TextRect(TheRect, 10, 10, "Hello World");
  Canvas->Brush->Color = clBlack;
  Canvas->FrameRect(TheRect);
}
```

Delphi Examples:

```
{
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
}
procedure TForm1.Button1Click(Sender: TObject);
var TheRect: TRect;
begin
  TheRect := Rect(10,10,100,100);
```

```
  Form1.Canvas.TextRect(TheRect,10,10,'Hello, world!');
  Form1.Canvas.Brush.Color := clBlack;
  Form1.Canvas.FrameRect(TheRect);
end;
```

See Also

ClipRect (see page 2718)

Font (see page 2721)

TextOut (see page 2707)

TextFlags (see page 2728)

8.1.7.4.32 TCanvas.TextWidth Method

Returns the width, in pixels, of a string rendered in the current font.

Pascal

```
function TextWidth(const Text: string): Integer;
```

C++

```
__fastcall int TextWidth(const UnicodeString Text);
```

Description

Use TextWidth to determine the length a string will occupy in the image. TextWidth indicates whether a given string will fit in the available space. Other graphical elements in the image such as lines, or additional strings can be positioned to accommodate the width of the text.

TextWidth returns the same value as TextExtent (see page 2705)(Text).cx.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
```

```
  number of pixels required to fit 46 characters.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    with PageControl1 do
    begin
      for i := 0 to PageCount - 1 do
      begin
        if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
          TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
        if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
          TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
      end;
    end;
  end;
```

See Also

Font (see page 2721)

TextExtent (see page 2705)

TextHeight (see page 2706)

TextOut (see page 2707)

TextRect (see page 2713)

8.1.7.4.33 TCanvas.TryLock Method

Locks the canvas if it is currently unlocked.

Pascal

```
function TryLock: Boolean;
```

C++

```
__fastcall Boolean TryLock();
```

Description

Call TryLock to prevent other threads from writing on the surface of the canvas until the Unlock (see page 2716) method is called. Unlike calls to the Lock (see page 2694) method, which nest so that every call to Lock (see page 2694) must be matched with a subsequent call to Unlock (see page 2716), TryLock need only be matched by a call to Unlock (see page 2716) if the canvas was not already locked.

If the canvas is unlocked, TryLock locks the canvas, sets the LockCount (see page 2725) property to 1, and returns true. If the canvas is already locked, TryLock returns false without making any changes.

Use TryLock before attempting changes to the canvas that could interfere with other code that is protected by a Lock (see page 2694)/Unlock (see page 2716) pair of method calls.

See Also

Lock (see page 2694)

LockCount (see page 2725)

Unlock (see page 2716)

8.1.7.4.34 TCanvas.Unlock Method

Decreases the LockCount (see page 2725) property, allowing other threads to execute when LockCount (see page 2725) reaches 0.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a critical section begun with a call to the Lock (see page 2694) method. Every call to Lock (see page 2694) must be matched by a call to Unlock.

See Also

Lock (see page 2694)

LockCount (see page 2725)

TryLock (see page 2715)

TThread

Locking Objects

8.1.7.5 TCanvas Properties

The properties of the TCanvas class are listed here.

Public Properties

	Name	Description
R	CanvasOrientation (see page 2718)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 2718)	Read-only property that specifies the boundaries of the clipping rectangle.
	Handle (see page 2722)	Specifies the handle for this canvas.
R	LockCount (see page 2725)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 2727)	Specifies the current drawing position of the Pen (see page 2725).
	Pixels (see page 2727)	Specifies the color of the pixels within the current ClipRect (see page 2718).
	TextFlags (see page 2728)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 2716)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 2719)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 2721)	Specifies the font to use when writing text on the image.
	Pen (see page 2725)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

8.1.7.5.1 TCanvas.Brush Property

Determines the color and pattern for filling graphical shapes and backgrounds.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

Set the Brush property to specify the color and pattern to use when drawing the background or filling in graphical shapes. The value of Brush is a TBrush (see page 2664) object. Set the properties of the TBrush (see page 2664) object to specify the color and pattern or bitmap to use when filling in spaces on the canvas.

Note: Setting the Brush property assigns the specified TBrush (see page 2664) object, rather than replacing the current TBrush (see page 2664) object.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> BrushBmp(new Graphics::TBitmap);
  BrushBmp->LoadFromFile("../bm1.BMP");
  Form1->Canvas->Brush->Bitmap = BrushBmp.get();
  Form1->Canvas->FillRect(Rect(0,0,100,100));
  Form1->Canvas->Brush->Bitmap = NULL;
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

Chord (see page 2682)

Ellipse (see page 2688)

FillRect (see page 2689)

FloodFill (see page 2690)

FrameRect (see page 2692)

Pie (see page 2695)

Rectangle (see page 2700)

RoundRect (see page 2702)

8.1.7.5.2 TCanvas.CanvasOrientation Property

Determines the orientation of the canvas as left-to-right or right-to-left.

Pascal

```
property CanvasOrientation: TCanvasOrientation;
```

C++

```
__property TCanvasOrientation CanvasOrientation;
```

Description

CanvasOrientation is a read-only property used for bi-directional support. It indicates whether the orientation of the canvas is left-to-right (coLeftToRight) or right-to-left (coRightToLeft). The default is coLeftToRight, meaning that the 0,0 coordinate is at the top left corner of the canvas. In Middle eastern locales the orientation may be coRightToLeft, meaning that the 0,0 coordinate is at the top right corner of the canvas. This property is important for grid controls because it changes the grid's canvas orientation during drawing.

See Also

BiDiMode

8.1.7.5.3 TCanvas.ClipRect Property

Read-only property that specifies the boundaries of the clipping rectangle.

Pascal

```
property ClipRect: TRect;
```

C++

```
__property TRect ClipRect;
```

Description

Use ClipRect to determine where the canvas needs painting. ClipRect limits the drawing region of the canvas so that any drawing that occurs at coordinates outside the ClipRect is clipped and does not appear in the image.

When handling a form's OnPaint event, the canvas' ClipRect property is set to the rectangle that needs to be painted. Portions of the image that do not overlap the ClipRect do not need to be drawn. Thus, OnPaint routines can use the value of ClipRect to optimize painting, speeding the overall performance of the application.

Note: ClipRect can be modified using the SelectClipRgn Win32 API.

C++ Examples:

```
/*
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    HRGN MyRgn;

    MyRgn = ::CreateRectRgn(100,100,200,200);
    ::SelectClipRgn(Image1->Canvas->Handle,MyRgn);
    Image1->Canvas->Brush->Color = clRed;
    Image1->Canvas->FillRect(Image1->Canvas->ClipRect);
    Image1->Invalidate();
    ::SelectClipRgn(Image1->Canvas->Handle,NULL);
    ::DeleteObject(MyRgn);
}
```

Delphi Examples:

```
{
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
    MyRgn: HRGN;
begin
    MyRgn := CreateRectRgn(100,100,200,200);
    SelectClipRgn(Image1.Canvas.Handle,MyRgn);
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
    Image1.Invalidate;
    SelectClipRgn(Image1.Canvas.Handle, HRGN(nil));
    DeleteObject(MyRgn);
end;
```

See Also

OnPaint (see page 2233)

8.1.7.5.4 TCanvas.CopyMode Property

Specifies how a graphical image is copied onto the canvas.

Pascal

```
property CopyMode: TCopyMode;
```

C++

```
__property TCopyMode CopyMode;
```

Description

Set CopyMode to affect the way graphical images are drawn onto the canvas. The CopyMode is used when copying an image from another canvas using the CopyRect (see page 2683) method. CopyMode is also used by TBitmap (see page 2628) objects when they draw themselves to a canvas.

Use CopyMode to achieve a variety of affects when rendering an image. Achieve special effects like merged images and making parts of a bitmap transparent by combining multiple images with different CopyModes.

C++ Examples:

```
/*
The following example uses the CopyMode of an image’s canvas
```

```
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    TRect SrcRect, DstRect;
    Image2->Canvas->CopyMode = cmSrcCopy;
    DstRect = Rect(0, 0, Image2->Width, Image2->Height);
    SrcRect = Rect(0, 0, Image1->Width, Image1->Height);
    Image2->Canvas->CopyRect(DstRect, Image1->Canvas, SrcRect);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
 TRect r;
 Copy1Click(Sender); // Do the same thing as the copy menu item.
 Image1->Canvas->CopyMode = cmWhiteness;
 r = Rect(0, 0, Image1->Width, Image1->Height);
 Image1->Canvas->CopyRect(r, Image1->Canvas, r);
 Image1->Canvas->CopyMode = cmSrcCopy; // Restore the copy mode.
}
```

Delphi Examples:

```
{
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
}
procedure TForm1.Copy1Click(Sender: TObject);
var
  DstRect, SrcRect: TRect;
begin
  with Image2.Canvas do
  begin
    CopyMode := cmSrcCopy;
    DstRect := Rect(0, 0, Image2.Width, Image2.Height);
    SrcRect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(DstRect, Image1.Canvas, SrcRect);
  end;
end;

procedure TForm1.Cut1Click(Sender: TObject);
var
  ARect: TRect;
begin
  Copy1Click(Sender); { do the same thing as the copy menu item }
  with Image1.Canvas do
  begin
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(ARect, Image1.Canvas, ARect);
    CopyMode := cmSrcCopy; { restore the copy mode }
  end;
end;
```

See Also

BrushCopy (see page 2680)

CopyRect (see page 2683)

Draw (see page 2685)

StretchDraw (see page 2704)

8.1.7.5.5 TCanvas.Font Property

Specifies the font to use when writing text on the image.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

Set Font to specify the font to use for writing text on the image. The value of Font is a T Font object. Set the properties of the T Font object to specify the font face, color, size, style, and any other aspects of the font.

The Canvas.Font property is only guaranteed to equal the Font property if you have an owner-drawn listbox (where you are expected to use the canvas). Even in that case, it is only guaranteed to equal the Font property once the first paint message is underway. If you use a canvas inside an ownerdraw event, everything should work as expected. Using the canvas for a reason outside the scope of its intended usage may give unpredictable results.

Note: Setting the Font property assigns the specified T Font object, rather than replacing the current T Font object.

C++ Examples:

```
/*
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    LOGFONT lf; // Windows native font structure

    Canvas->Brush->Style = bsClear; // set the brush style to transparent
    ZeroMemory(&lf, sizeof(LOGFONT));

    lf.lfHeight = 20;
    lf.lfEscapement = 10 * 45; // degrees to rotate
    lf.lfOrientation = 10 * 45;
    lf.lfCharSet = DEFAULT_CHARSET;
    strcpy(lf.lfFaceName, "Tahoma");

    Canvas->Font->Handle = CreateFontIndirect(&lf);
    Canvas->TextOut(10, 100, "Rotated text"); // output the rotated font
}
```

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
```

```
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
```

See Also

TextOut (see page 2707)

TextRect (see page 2713)

TFont (see page 2730)

Assign (see page 2733)

8.1.7.5.6 TCanvas.Handle Property

Specifies the handle for this canvas.

Pascal

```
property Handle: HDC;
```

C++

```
__property HDC Handle;
```

Description

The Handle property specifies the Windows GDI handle to the device context for this canvas.

Set Handle to the HDC for the device context the canvas must draw into. When a windowed control responds to a Windows paint message, the HDC for drawing is passed in to the PaintWindow method. In other cases, an HDC can be obtained for a window by calling the GetDeviceContext method of a control. Additionally, Windows provides API calls to obtain an HDC for a printer or for a memory image.

Read the Handle property to supplement the drawing services provided by the TCanvas (see page 2673) object with API calls that require a handle to a device context. Most of the Windows GDI calls require an HDC.

TCanvas (see page 2673) does not own the HDC. Applications must create an HDC and set the Handle property. Applications must release the HDC when the canvas no longer needs it. Setting the Handle property of a canvas that already has a valid HDC will not automatically release the initial HDC.

Note: Some descendants of TCanvas (see page 2673), such as TControlCanvas, do own the HDC. Do not set the Handle property for these objects. They fetch and free their own Handle.

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
```

```
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/

#include <memory>       //for STL auto_ptr class

class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyControl::TMyControl(TComponent* Owner)
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
{
  std::auto_ptr<TCanvas> tempCanvas(new TCanvas);
  HWND notUsed;
  tempCanvas->Handle = GetDeviceContext(notUsed);
  std::auto_ptr<TImage> image2save(new TImage(Form1)); // The owner will clean this up.
  image2save->Height = Height;
  image2save->Width = Width;
  TRect destRect = Rect(0,0,Width,Height);
  TRect sourceRect = destRect;
  image2save->Canvas->CopyRect(destRect, tempCanvas.get(), sourceRect);
  LogPal SysPal;
  SysPal.lpal.palVersion = 0x300;
  SysPal.lpal.palNumEntries = 256;
  GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
  image2save->Picture->Bitmap->Palette =
    CreatePalette(dynamic_cast<const tagLOGPALETTE *>(&SysPal.lpal));
  image2save->Picture->SaveToFile(fileName);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static_cast<TMyControl*>(MyControl1)->SaveAsBmp("../foo.bmp");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TMyControl* control = new TMyControl(Form1); // The owner will clean this up.

  MyControl1 = control;
  control->Parent = this;
```

```
  control->Visible = true;
}
```

Delphi Examples:

```
{
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and then
double click on the newly created foo.bmp file to view.
}
type
  TMyControl = class(TColorListBox)
    procedure SaveAsBmp(fileName: TFileName);
  end;
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyControl1: TMyControl;
implementation

{$R *.dfm}

type
  LogPal = record
  lpal : TLogPalette;
  dummy:Array[0..255] of TPaletteEntry;
  end;

procedure TMyControl.SaveAsBmp(fileName: TFileName);
var
  Source: TComponent;
  SysPal : LogPal;
  tempCanvas: TCanvas;
  sourceRect, destRect: TRect;
  image2save: TImage;
  notUsed: HWND;
begin
  tempCanvas := TCanvas.Create;
  try
    tempCanvas.Handle := GetDeviceContext(notUsed);
    image2save:=TImage.create(self);
    try
      with image2save do
      begin
        Height := Self.Height;
        Width :=  Self.Width;
        destRect := Rect(0,0,Width,Height);
        sourceRect := destRect;
        Canvas.CopyRect(destRect,tempCanvas,sourceRect);
        SysPal.lPal.palVersion:=$300;
        SysPal.lPal.palNumEntries:=256;
        GetSystemPaletteEntries(
          tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
        Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
      end;
      image2save.Picture.SaveToFile(fileName);
```

```
      finally
       image2save.Free;
      end;
    finally
      tempCanvas.Free;
    end;
  end;
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    MyControl1.SaveAsBmp('foo.bmp');
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyControl1:= TMyControl.Create(Form1);
    MyControl1.Parent:= Form1;
    MyControl1.visible := true;
  end;
```

See Also

GetDeviceContext

PaintWindow

8.1.7.5.7 TCanvas.LockCount Property

Indicates the number of times the canvas has been locked to prevent interference from other threads.

Pascal

```
property LockCount: Integer;
```

C++

```
__property int LockCount;
```

Description

Read LockCount to determine whether the canvas is locked. Whenever the Lock (see page 2694) method is called to block other threads from interfering with the canvas, LockCount increases. Whenever the Unlock (see page 2716) method is called at the end of a critical section, LockCount decreases. When LockCount is 0, other threads may draw on the canvas.

See Also

Lock (see page 2694)

TThread

Unlock (see page 2716)

8.1.7.5.8 TCanvas.Pen Property

Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

Pascal

```
property Pen: TPen;
```

C++

```
__property TPen Pen;
```

Description

Set Pen to specify the pen to use for drawing lines and outlining shapes in the image. The value of Pen is a TPen (see page

2801) object. Set the properties of the TPen (see page 2801) object to specify the color, style, width, and mode of the pen.

Note: Setting the Pen property assigns the specified TPen (see page 2801) object, rather than replacing the current TPen (see page 2801) object.

C++ Examples:

```
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = dynamic_cast<TPaintBox *>(Sender);
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
This example paints a white five-pointed star in a paintbox
control:
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TPaintBox do
  begin
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
    Point(10, 30), Point(60, 60), Point(40, 10)]);
  end;
end;
```

See Also

Arc (see page 2679)

Chord (see page 2682)

Ellipse (see page 2688)

LineTo (see page 2693)

MoveTo (see page 2695)

PenPos (see page 2727)

Pie (see page 2695)

Polygon (see page 2697)

Polyline (see page 2698)

Rectangle (see page 2700)

RoundRect (see page 2702)

TPen (see page 2801)

Assign (see page 2803)

8.1.7.5.9 TCanvas.PenPos Property

Specifies the current drawing position of the Pen (see page 2725).

Pascal

```
property PenPos: TPoint;
```

C++

```
__property TPoint PenPos;
```

Description

Read PenPos to learn the current drawing position of the Pen (see page 2725). This is the starting point of a line drawn by the LineTo (see page 2693) method. Setting the PenPos property is equivalent to calling the MoveTo (see page 2695) method.

See Also

LineTo (see page 2693)

MoveTo (see page 2695)

Pen (see page 2725)

8.1.7.5.10 TCanvas.Pixels Property

Specifies the color of the pixels within the current ClipRect (see page 2718).

Pascal

```
property Pixels [X, Y: Integer]: TColor;
```

C++

```
__property TColor Pixels[int X, Y];
```

Description

Read Pixels to learn the color on the drawing surface at a specific pixel position within the current clipping region. If the position is outside the clipping rectangle, reading the value of Pixels returns -1.

Write Pixels to change the color of individual pixels on the drawing surface. Use Pixels for detailed effects on an image. Pixels may also be used to determine the color that should be used for the FillRect (see page 2689) method.

Not every device context supports the Pixels property. Reading the Pixels property for such a device context will return a value of -1. Setting the Pixels property for such a device context does nothing.

C++ Examples:

```
/*
This example draws a red line (very slowly) when the user
presses a button. Attach the following code to the button's
OnClick event handler:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 10; i <= 200; i++)
```

```
    Canvas->Pixels[i][10] = clRed;
}
```

Delphi Examples:

```
{
This example draws a red line (very slowly) when the user
presses a button. Attach the following code to the button's
OnClick event handler:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  W: Word;
begin
    for W := 10 to 200 do
      Canvas.Pixels[W, 10] := clRed;
end;
```

See Also

ClipRect (see page 2718)

Handle (see page 2722)

FillRect (see page 2689)

8.1.7.5.11 TCanvas.TextFlags Property

Specifies how text is written to the canvas.

Pascal

```
property TextFlags: Longint;
```

C++

```
__property Longint TextFlags;
```

Description

Use TextFlags to influence the way text is drawn when calling the TextOut (see page 2707) or TextRect (see page 2713) method. TextFlags is an integer made by combining any of the following constants:

Constant	Meaning
ETO_CLIPPED	Only the text that fits within a specified rectangle is displayed. This flag is automatically added when calling the TextRect (see page 2713) method. When writing text using TextOut (see page 2707) method, this flag has no effect because the drawing boundaries are determined by the TextExtent (see page 2705) property.
ETO_OPAQUE	The text is drawn with an opaque background color. Using this flag improves performance but does not allow images behind the text rectangle to show.
ETO_RTLREADING	Text strings are displayed from right to left. This is available only with Mideast versions of Windows, where text is written from right to left. Often this is used in combination with a CanvasOrientation (see page 2718) of coRightToLeft.
ETO_GLYPH_INDEX	The text is an array of character placement codes that are parsed directly by the Windows GDI. Glyph indexing only applies to TrueType fonts, but the flag can be used with other fonts to indicate that the GDI should process the text string directly without language processing. See the Windows GDI documentation for more details.
ETO_IGNORELANGUAGE	This is an undocumented Microsoft flag. Check updates of Microsoft documentation for when information is provided.
ETO_NUMERICSLOCAL	All numbers in the text are displayed using digits specific to the current locale.
ETO_NUMERICSLATIN	All numbers in the text are displayed using European digits.

ETO_PDY	This flag is only relevant when using the Windows API ExtTextOut. When this is set, the array pointed to by lpDx contains pairs of values. The first value of each pair is, as usual, the distance between origins of adjacent character cells, but the second value is the displacement along the vertical direction of the font.

See Also

CanvasOrientation (see page 2718)

TextExtent (see page 2705)

TextOut (see page 2707)

TextRect (see page 2713)

8.1.7.6 TCanvas Events

The events of the TCanvas class are listed here.

Public Events

	Name	Description
	OnChange (see page 2729)	Occurs when the image has just changed.
	OnChanging (see page 2730)	Occurs just before a change is made to the image.

8.1.7.6.1 TCanvas.OnChange Event

Occurs when the image has just changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the image on the canvas changes. When an application calls one of the drawing methods of TCanvas (see page 2673), the following steps occur.

1An OnChanging (see page 2730) event occurs.

2The TCanvas (see page 2673) method makes a change to the image.

3An OnChange event occurs.

OnChange events occur on changes to the image, not changes to the properties of the canvas. Applications can respond to changes in the Font (see page 2721), Brush (see page 2716), and Pen (see page 2725) properties by assigning OnChange event handlers to the corresponding TFont (see page 2730), TBrush (see page 2664), and TPen (see page 2801) objects. However, TCanvas (see page 2673) assigns its own OnChange event handlers for those objects. Before assigning an OnChange event handler to one of these objects, read the existing event handler. Use that value to call the TCanvas (see page 2673) event handler from the new event handler.

See Also

OnChanging (see page 2730)

OnChange (see page 2768)

8.1.7.6.2 TCanvas.OnChanging Event

Occurs just before a change is made to the image.

Pascal

```
property OnChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChanging;
```

Description

Write an OnChanging event handler to take specific action before the image on the canvas changes. When an application calls one of the drawing methods of TCanvas (see page 2673), the following steps occur.

1An OnChanging event occurs.

2The TCanvas (see page 2673) method makes a change to the image.

3An OnChange (see page 2729) event occurs.

See Also

OnChange (see page 2729)

8.1.8 TFont Class

TFont encapsulates a system font.

Class Hierarchy

Pascal

```
TFont = class(TGraphicsObject);
```

C++

```
class TFont : public TGraphicsObject;
```

File

Graphics

Description

TFont describes font characteristics used when displaying text. TFont defines a set of characters by specifying the height, font family (typeface), attributes (such as bold or italic) and so on.

TFont encapsulates a windows logical font.

See Also

TTextAttributes

Using the Main VCL

8.1.8.1 TFont Members

The following tables list the members exposed by TFont.

Protected Methods

	Name	Description
	Changed (see page 2734)	Generates an OnChange event when the properties of the graphics object change.
	GetCharset (see page 2735)	This is GetCharset, a member of class TFont.
	GetHandle (see page 2735)	This is GetHandle, a member of class TFont.
	GetHeight (see page 2735)	This is GetHeight, a member of class TFont.
	GetName (see page 2735)	This is GetName, a member of class TFont.
	GetOrientation (see page 2736)	This is GetOrientation, a member of class TFont.
	GetPitch (see page 2736)	This is GetPitch, a member of class TFont.
	GetSize (see page 2736)	This is GetSize, a member of class TFont.
	GetStyle (see page 2736)	This is GetStyle, a member of class TFont.
	Lock (see page 2766)	Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.
	SetCharset (see page 2737)	This is SetCharset, a member of class TFont.
	SetColor (see page 2737)	This is SetColor, a member of class TFont.
	SetHandle (see page 2737)	This is SetHandle, a member of class TFont.
	SetHeight (see page 2737)	This is SetHeight, a member of class TFont.
	SetName (see page 2737)	This is SetName, a member of class TFont.
	SetOrientation (see page 2738)	This is SetOrientation, a member of class TFont.
	SetPitch (see page 2738)	This is SetPitch, a member of class TFont.
	SetSize (see page 2738)	This is SetSize, a member of class TFont.
	SetStyle (see page 2738)	This is SetStyle, a member of class TFont.
	Unlock (see page 2767)	Releases the lock on execution threads set by the Lock (see page 2766) method.

Public Events

	Name	Description
	OnChange (see page 2768)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	Assign (see page 2733)	Copies the properties of another TFont (see page 2730) object.
	Create (see page 2732)	Creates and initializes an instance of TFont (see page 2730).
	Destroy (see page 2732)	Destroys an instance of TFont (see page 2730).
	HandleAllocated (see page 2766)	Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

Public Properties

	Name	Description
	FontAdapter (see page 2741)	Designates an interface for providing the font information to ActiveX controls.
	Handle (see page 2741)	Provides access to the handle of a system font.
	OwnerCriticalSection (see page 2767)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.
	PixelsPerInch (see page 2745)	Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Published Properties

	Name	Description
	Charset (see page 2739)	Specifies the character set of the font.
	Color (see page 2739)	Specifies the color of the text.

	Height (see page 2742)	Specifies the height of the font.
	Name (see page 2743)	Identifies the typeface of the font.
	Orientation (see page 2744)	Specifies text orientation.
	Pitch (see page 2744)	Specifies whether the characters in the font all have the same width.
	Size (see page 2745)	Specifies the height of the font in points.
	Style (see page 2746)	Determines whether the font is normal, italic, underlined, bold, and so on.

8.1.8.2 TFont.Create Constructor

Creates and initializes an instance of TFont (see page 2730).

Pascal

```
constructor Create;
```

C++

```
__fastcall TFont();
```

Description

Call Create to create a font object at runtime.

Create allocates memory for the font resource, and initializes the Color (see page 2739) property to clWindowText (see page 2966). Create also initializes the PixelsPerInch (see page 2745) property based on the screen resolution.

See Also

PixelsPerInch (see page 2745)

8.1.8.3 TFont.Destroy Destructor

Destroys an instance of TFont (see page 2730).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFont();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TFont (see page 2730) reference is not nil, and only then calls Destroy. Destroy frees the resource identified by the Handle (see page 2741) property.

See Also

Handle (see page 2741)

System::TObject::Free

8.1.8.4 TFont Methods

The methods of the TFont class are listed here.

Protected Methods

	Name	Description
	Changed (see page 2734)	Generates an OnChange event when the properties of the graphics object change.
	GetCharset (see page 2735)	This is GetCharset, a member of class TFont.
	GetHandle (see page 2735)	This is GetHandle, a member of class TFont.
	GetHeight (see page 2735)	This is GetHeight, a member of class TFont.
	GetName (see page 2735)	This is GetName, a member of class TFont.
	GetOrientation (see page 2736)	This is GetOrientation, a member of class TFont.
	GetPitch (see page 2736)	This is GetPitch, a member of class TFont.
	GetSize (see page 2736)	This is GetSize, a member of class TFont.
	GetStyle (see page 2736)	This is GetStyle, a member of class TFont.
	SetCharset (see page 2737)	This is SetCharset, a member of class TFont.
	SetColor (see page 2737)	This is SetColor, a member of class TFont.
	SetHandle (see page 2737)	This is SetHandle, a member of class TFont.
	SetHeight (see page 2737)	This is SetHeight, a member of class TFont.
	SetName (see page 2737)	This is SetName, a member of class TFont.
	SetOrientation (see page 2738)	This is SetOrientation, a member of class TFont.
	SetPitch (see page 2738)	This is SetPitch, a member of class TFont.
	SetSize (see page 2738)	This is SetSize, a member of class TFont.
	SetStyle (see page 2738)	This is SetStyle, a member of class TFont.

Public Methods

	Name	Description
	Assign (see page 2733)	Copies the properties of another TFont (see page 2730) object.

8.1.8.4.1 TFont.Assign Method

Copies the properties of another TFont (see page 2730) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to set the properties of the font to match the characteristics of the Source object. If Source is another TFont (see page 2730) object, Assign matches the font characteristics of the source. Assign does not copy the PixelsPerInch (see page 2745) property, so this method can be used to copy a screen font to a printer font, or vice versa.

If the Source parameter is not another TFont (see page 2730) object, Assign calls the inherited method so that the font can be copied from any object that can assign font properties in its AssignTo method.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
```

See Also

PixelsPerInch (see page 2745)

AssignTo

8.1.8.4.2 TFont.Changed Method

Generates an OnChange event when the properties of the graphics object change.

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

Descendants of TGraphicsObject call Changed after making changes to the properties encapsulated by the TGraphicsObject object.

See Also

OnChange (see page 2768)

8.1.8.4.3 TFont.GetCharset Method

Pascal

```
function GetCharset: TFontCharset;
```

C++

```
__fastcall TFontCharset GetCharset();
```

Description

This is GetCharset, a member of class TFont.

8.1.8.4.4 TFont.GetHandle Method

Pascal

```
function GetHandle: HFont;
```

C++

```
__fastcall HFont GetHandle();
```

Description

This is GetHandle, a member of class TFont.

8.1.8.4.5 TFont.GetHeight Method

Pascal

```
function GetHeight: Integer;
```

C++

```
__fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TFont.

8.1.8.4.6 TFont.GetName Method

Pascal

```
function GetName: TFontName;
```

C++

```
__fastcall TFontName GetName();
```

Description

This is GetName, a member of class TFont.

8.1.8.4.7 TFont.GetOrientation Method

Pascal

```
function GetOrientation: Integer;
```

C++

```
__fastcall int GetOrientation();
```

Description

This is GetOrientation, a member of class TFont.

8.1.8.4.8 TFont.GetPitch Method

Pascal

```
function GetPitch: TFontPitch;
```

C++

```
__fastcall TFontPitch GetPitch();
```

Description

This is GetPitch, a member of class TFont.

8.1.8.4.9 TFont.GetSize Method

Pascal

```
function GetSize: Integer;
```

C++

```
__fastcall int GetSize();
```

Description

This is GetSize, a member of class TFont.

8.1.8.4.10 TFont.GetStyle Method

Pascal

```
function GetStyle: TFontStyles;
```

C++

```
__fastcall TFontStyles GetStyle();
```

Description

This is GetStyle, a member of class TFont.

8.1.8.4.11 TFont.SetCharset Method

Pascal

```
procedure SetCharset(const Value: TFontCharset);
```

C++

```
__fastcall SetCharset(const TFontCharset Value);
```

Description

This is SetCharset, a member of class TFont.

8.1.8.4.12 TFont.SetColor Method

Pascal

```
procedure SetColor(const Value: TColor);
```

C++

```
__fastcall SetColor(const TColor Value);
```

Description

This is SetColor, a member of class TFont.

8.1.8.4.13 TFont.SetHandle Method

Pascal

```
procedure SetHandle(const Value: HFont);
```

C++

```
__fastcall SetHandle(const HFont Value);
```

Description

This is SetHandle, a member of class TFont.

8.1.8.4.14 TFont.SetHeight Method

Pascal

```
procedure SetHeight(const Value: Integer);
```

C++

```
__fastcall SetHeight(const int Value);
```

Description

This is SetHeight, a member of class TFont.

8.1.8.4.15 TFont.SetName Method

Pascal

```
procedure SetName(const Value: TFontName);
```

C++

```
__fastcall SetName(const TFontName Value);
```

Description

This is SetName, a member of class TFont.

8.1.8.4.16 TFont.SetOrientation Method

Pascal

```
procedure SetOrientation(const Value: Integer);
```

C++

```
__fastcall SetOrientation(const int Value);
```

Description

This is SetOrientation, a member of class TFont.

8.1.8.4.17 TFont.SetPitch Method

Pascal

```
procedure SetPitch(const Value: TFontPitch);
```

C++

```
__fastcall SetPitch(const TFontPitch Value);
```

Description

This is SetPitch, a member of class TFont.

8.1.8.4.18 TFont.SetSize Method

Pascal

```
procedure SetSize(const Value: Integer);
```

C++

```
__fastcall SetSize(const int Value);
```

Description

This is SetSize, a member of class TFont.

8.1.8.4.19 TFont.SetStyle Method

Pascal

```
procedure SetStyle(const Value: TFontStyles);
```

C++

```
__fastcall SetStyle(const TFontStyles Value);
```

Description

This is SetStyle, a member of class TFont.

8.1.8.5 TFont Properties

The properties of the TFont class are listed here.

Public Properties

	Name	Description
	FontAdapter (see page 2741)	Designates an interface for providing the font information to ActiveX controls.
	Handle (see page 2741)	Provides access to the handle of a system font.
	PixelsPerInch (see page 2745)	Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Published Properties

	Name	Description
	Charset (see page 2739)	Specifies the character set of the font.
	Color (see page 2739)	Specifies the color of the text.
	Height (see page 2742)	Specifies the height of the font.
	Name (see page 2743)	Identifies the typeface of the font.
	Orientation (see page 2744)	Specifies text orientation.
	Pitch (see page 2744)	Specifies whether the characters in the font all have the same width.
	Size (see page 2745)	Specifies the height of the font in points.
	Style (see page 2746)	Determines whether the font is normal, italic, underlined, bold, and so on.

8.1.8.5.1 TFont.Charset Property

Specifies the character set of the font.

Pascal

```
property Charset: TFontCharset;
```

C++

```
__property TFontCharset Charset;
```

Description

Set Charset to identify the character set of the font. Each typeface (specified by the Name (see page 2743) property) supports one or more character sets. Check the information supplied by the font vendor to determine what values of Charset are valid.

See Also

Name (see page 2743)

8.1.8.5.2 TFont.Color Property

Specifies the color of the text.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to specify the color of the text characters (as opposed to the background color). If Color is set to a dithered color, the text appears in the closest non-dithered value.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
```

```
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

Color

8.1.8.5.3 TFont.FontAdapter Property

Designates an interface for providing the font information to ActiveX controls.

Pascal

```
property FontAdapter: IChangeNotifier;
```

C++

```
__property IChangeNotifier FontAdapter;
```

Description

Do not use the FontAdapter property. FontAdapter is used internally by ActiveX controls to communicate font information through an OLE interface.

See Also

TFontAdapter

8.1.8.5.4 TFont.Handle Property

Provides access to the handle of a system font.

Pascal

```
property Handle: HFont;
```

C++

```
__property HFont Handle;
```

Description

The Handle property indicates the Windows GDI font object handle.

Use Handle as a parameter when calling Windows API functions that require a font handle. Set Handle to make the TFont (see page 2730) object represent the Windows font indicated by the font handle. Setting Handle releases the current Windows font resource.

Note: When possible, change a font by setting the other properties of TFont (see page 2730) rather than using the Handle property. Setting Handle does not reset the properties that describe the font. However, an OnChange (see page 2768) event does occur when Handle is set directly by an application.

C++ Examples:

```
/*
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    LOGFONT lf; // Windows native font structure
```

```
    Canvas->Brush->Style = bsClear; // set the brush style to transparent
    ZeroMemory(&lf, sizeof(LOGFONT));

    lf.lfHeight = 20;
    lf.lfEscapement = 10 * 45; // degrees to rotate
    lf.lfOrientation = 10 * 45;
    lf.lfCharSet = DEFAULT_CHARSET;
    strcpy(lf.lfFaceName, "Tahoma");

    Canvas->Font->Handle = CreateFontIndirect(&lf);
    Canvas->TextOut(10, 100, "Rotated text"); // output the rotated font
}
```

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
```

See Also

OnChange (see page 2768)

8.1.8.5.5 TFont.Height Property

Specifies the height of the font.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to specify the height of the font in pixels. If the value is negative, the internal leading that appears at the top of each line of text is not measured. If the value is positive, Height represents the height of the characters plus the internal leading.

To specify the size of the font in points, use the Size (see page 2745) property instead. Users usually specify font size in points, while applications may be concerned with the actual size of the font in pixels when displaying the font on the screen.

The value of Height can be obtained from the point size using this formula:

```
Font.Height = -Font.Size * Font.PixelsPerInch / 72
```

When the Height property has a positive value, the Size (see page 2745) property has a negative value. When the Size (see page 2745) property has a positive value, the Height property has a negative value.

See Also

Size (see page 2745)

8.1.8.5.6 TFont.Name Property

Identifies the typeface of the font.

Pascal

```
property Name: TFontName;
```

C++

```
__property TFontName Name;
```

Description

Use Name to specify the typeface of the font. If the font family described by Name includes multiple character sets, be sure to set the Charset (see page 2739) property as well.

Note: If the combination of font family (typeface) and attributes (such as bold or italic) specifies a font that is not available on the system, the system substitutes a different font.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

Charset (see page 2739)

Pitch (see page 2744)

Size (see page 2745)

8.1.8.5.7 TFont.Orientation Property

Specifies text orientation.

Pascal

```
property Orientation: Integer;
```

C++

```
__property int Orientation;
```

Description

Use Orientation to specify the orientation of text characters. It is the angle in tenths of degrees between the character's baseline and device x-axis. This is similar to the **lfOrientation** field of the Windows GDI **LOGFONT** structure that defines font attributes.

See Also

Color (see page 2739)

8.1.8.5.8 TFont.Pitch Property

Specifies whether the characters in the font all have the same width.

Pascal

```
property Pitch: TFontPitch;
```

C++

```
__property TFontPitch Pitch;
```

Description

Each typeface has an associated pitch. The characters in fixed-pitch fonts are all the same width. The characters in variable-pitch fonts differ in width. Fixed pitch fonts are generally used for representing source code, because they allow the indentation to line up properly. Variable pitch fonts are generally used to give text a more natural appearance

Here are the possible values for Pitch:

Value	Meaning
fpDefault	The font pitch is set to the default value, which depends on the font specified in the Name (see page 2743) property.
fpFixed	The font pitch is set to Fixed. All characters in the font have the same width.
fpVariable	The font pitch is set to Variable. The characters in the font have varying widths.

Setting the Pitch of a fixed-pitch font to fpVariable or a variable-pitch font to fpFixed will cause window system to find what it considers the best match to all of the font characteristics that have been specified. This might have no effect on the appearance of a font, or might cause the window system to substitute a different font with the appropriate pitch. For example, setting the pitch of Serif (a variable-pitch font) to fpFixed causes Courier to be displayed.

See Also

Name (see page 2743)

8.1.8.5.9 TFont.PixelsPerInch Property

Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

PixelsPerInch is used to ensure that when a font is copied from the form's canvas to the printer, the font is the same size in points. The PixelsPerInch property affects printer fonts only and should not be modified.

To modify the size of a font, use the Size (see page 2745) or Height (see page 2742) properties.

See Also

Height (see page 2742)

Size (see page 2745)

8.1.8.5.10 TFont.Size Property

Specifies the height of the font in points.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size to specify the point size of the font. If the value is negative, the internal leading that appears at the top of each line of text is included. If the value is positive, Size represents the height of the characters but not the internal leading.

To determine the size of the font in pixels, use the Height (see page 2742) property instead. The value of Size can be obtained from the height in pixels using this formula:

```
Font.Size = -Font.Height * 72 / Font.PixelsPerInch
```

When the Size property has a positive value, the Height (see page 2742) property has a negative value. When the Height (see page 2742) property has a positive value, the Size property has a negative value.

See Also

Height (see page 2742)

8.1.8.5.11 TFont.Style Property

Determines whether the font is normal, italic, underlined, bold, and so on.

Pascal

```
property Style: TFontStyles;
```

C++

```
__property TFontStyles Style;
```

Description

Use Style to add special characteristics to characters that use the font. Style is a set containing zero or more values from the following:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

C++ Examples:

```
/*
This example demonstrates how to set and clear font styles.
This example requires a button and a label that has text
with some styles set.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   Label1->Font->Style = TFontStyles() << fsBold << fsUnderline;
}
/*
This example demonstrates how to clear any Styles that were previously set.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
   Label1->Font->Style = TFontStyles();
}
```

Delphi Examples:

```
{
This example demonstrates how to set and clear font styles.
This example requires a button and a label that has text
with some styles set.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   Label1.Font.Style := [fsBold, fsUnderline];
end;
{
This code demonstrates how to clear any Styles that were
previously set.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
   Label1.Font.Style := [];
end;
```

See Also

Color (see page 2739)

8.1.9 TFontRecall Class

TFontRecall saves and restores the properties of a TFont (see page 2730) object.

Class Hierarchy

Pascal

```
TFontRecall = class(TRecall);
```

C++

```
class TFontRecall : public TRecall;
```

File

Graphics

Description

Use TFontRecall to store the current state of a TFont (see page 2730) object. You are then free to change the current state of the font object. To restore the font object to its original state, all that is necessary is to free the TFontRecall object and the TFont (see page 2730) object is automatically restored to the saved property values.

You can update the TFontRecall instance to reflect the current properties of the TFont (see page 2730) object by calling the Store method. You can prevent the TFontRecall destructor from updating the TFont (see page 2730) object by calling the Forget method.

See Also

TRecall

TFont (see page 2730)

8.1.9.1 TFontRecall Members

The following tables list the members exposed by TFontRecall.

Public Methods

	Name	Description
	Create (see page 2747)	Instantiates a TFontRecall (see page 2747) object.

8.1.9.2 TFontRecall.Create Constructor

Instantiates a TFontRecall (see page 2747) object.

Pascal

```
constructor Create(AFont: TFont);
```

C++

```
__fastcall TFontRecall(TFont AFont);
```

Description

Call Create to instantiate a font recall object at runtime. Create saves the current properties of the TFont (see page 2730) object specified by AFont, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of AFont and save them, replacing the values saved by this constructor. When you destroy the TFontRecall (see page 2747) instance, the most recently-saved properties are assigned to AFont, restoring it to a previous state.

See Also

Destroy

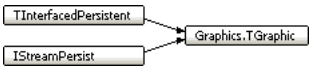
Reference

Store

8.1.10 TGraphic Class

TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.

Class Hierarchy



Pascal

```
TGraphic = class(TInterfacedPersistent, IStreamPersist);
```

C++

```
class TGraphic : public TInterfacedPersistent, public IStreamPersist;
```

File

Graphics

Description

TGraphic is an abstract class that cannot be instantiated. Descendant graphics objects override many of the methods of TGraphic to address the needs of their particular file format and graphical characteristics. TGraphic also introduces methods that work with TPicture (see page 2812) objects and the Clipboard. Properties of TGraphic provide information about the state and size of the graphic image.

When the type of graphic is known, store the graphic in its specific type object. Otherwise, use a TPicture (see page 2812) object that can hold any type of TGraphic.

See Also

TClipboard

TBitmap (see page 2628)

TPicture (see page 2812)

TIcon (see page 2768)

Tmetafile

TJPEGImage

TOleGraphic

TImage (see page 1048)

Graphic Controls

8.1.10.1 TGraphic Members

The following tables list the members exposed by TGraphic.

Protected Methods

	Name	Description
	Changed (see page 2751)	This is Changed, a member of class TGraphic.
	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 2752)	This is Draw, a member of class TGraphic.
	DrawTransparent (see page 2752)	This is DrawTransparent, a member of class TGraphic.
	Equals (see page 2752)	This is the overview for the Equals method overload.
	GetEmpty (see page 2753)	This is GetEmpty, a member of class TGraphic.
	GetHeight (see page 2753)	This is GetHeight, a member of class TGraphic.
	GetPalette (see page 2754)	This is GetPalette, a member of class TGraphic.
	GetSupportsPartialTransparency (see page 2754)	This is GetSupportsPartialTransparency, a member of class TGraphic.
	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 2754)	This is GetWidth, a member of class TGraphic.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.
	ReadData (see page 2757)	This is ReadData, a member of class TGraphic.
	SetHeight (see page 2758)	This is SetHeight, a member of class TGraphic.
	SetPalette (see page 2759)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 2759)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 2759)	This is SetWidth, a member of class TGraphic.
	WriteData (see page 2760)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	Create (see page 2750)	Constructs an object and initializes its data before the object is first used.
	LoadFromClipboardFormat (see page 2754)	Introduces an abstract method that loads the image from a variable in Clipboard format.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromStream (see page 2756)	Introduces an abstract method that loads the image from a stream.
	SaveToClipboardFormat (see page 2757)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
	SaveToFile (see page 2757)	Saves a graphics image to a file.
	SaveToStream (see page 2758)	Introduces an abstract method that saves the graphic image to a stream.
	SetSize (see page 2759)	Specifies the size of the graphic.

Public Properties

	Name	Description
R	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.
R	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

8.1.10.2 TGraphic.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TGraphic();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy

System::TObject::Free

8.1.10.3 TGraphic Methods

The methods of the TGraphic class are listed here.

Protected Methods

	Name	Description
V	Changed (see page 2751)	This is Changed, a member of class TGraphic.
V	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	Draw (see page 2752)	This is Draw, a member of class TGraphic.
V	DrawTransparent (see page 2752)	This is DrawTransparent, a member of class TGraphic.
V	Equals (see page 2752)	This is the overview for the Equals method overload.
A	GetEmpty (see page 2753)	This is GetEmpty, a member of class TGraphic.
A	GetHeight (see page 2753)	This is GetHeight, a member of class TGraphic.
V	GetPalette (see page 2754)	This is GetPalette, a member of class TGraphic.
V	GetSupportsPartialTransparency (see page 2754)	This is GetSupportsPartialTransparency, a member of class TGraphic.
V	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.
A	GetWidth (see page 2754)	This is GetWidth, a member of class TGraphic.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.

	ReadData (see page 2757)	This is ReadData, a member of class TGraphic.
	SetHeight (see page 2758)	This is SetHeight, a member of class TGraphic.
	SetPalette (see page 2759)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 2759)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 2759)	This is SetWidth, a member of class TGraphic.
	WriteData (see page 2760)	This is WriteData, a member of class TGraphic.

Public Methods

	Name	Description
	LoadFromClipboardFormat (see page 2754)	Introduces an abstract method that loads the image from a variable in Clipboard format.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromStream (see page 2756)	Introduces an abstract method that loads the image from a stream.
	SaveToClipboardFormat (see page 2757)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
	SaveToFile (see page 2757)	Saves a graphics image to a file.
	SaveToStream (see page 2758)	Introduces an abstract method that saves the graphic image to a stream.
	SetSize (see page 2759)	Specifies the size of the graphic.

8.1.10.3.1 TGraphic.Changed Method

Pascal

```
procedure Changed(Sender: TObject); virtual;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TGraphic.

8.1.10.3.2 TGraphic.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties

DefineBinaryProperty

DefineProperty

8.1.10.3.3 TGraphic.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); virtual; abstract;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect) = 0;
```

Description

This is Draw, a member of class TGraphic.

8.1.10.3.4 TGraphic.DrawTransparent Method

Pascal

```
procedure DrawTransparent(ACanvas: TCanvas; const Rect: TRect; Opacity: Byte); virtual;
```

C++

```
virtual __fastcall DrawTransparent(TCanvas ACanvas, const TRect Rect, Byte Opacity);
```

Description

This is DrawTransparent, a member of class TGraphic.

8.1.10.3.5 Equals Method

This is the overview for the Equals method overload.

Overload List

	Name	Description
	TGraphic.Equals (TGraphic) (see page 2752)	This is Equals, a member of class TGraphic.
	TGraphic.Equals (TObject) (see page 2753)	Checks if the current instance and the parameter are equal.

8.1.10.3.5.1 TGraphic.Equals Method (TGraphic)

Pascal

```
function Equals(Graphic: TGraphic): Boolean; virtual; overload; reintroduce;
```

C++

```
virtual __fastcall Boolean Equals(TGraphic Graphic);
```

Description

This is Equals, a member of class TGraphic.

8.1.10.3.5.2 TGraphic.Equals Method (TObject)

Checks if the current instance and the parameter are equal.

Pascal

```
function Equals(Obj: TObject): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean Equals(TObject * Obj);
```

Description

The function has one parameter, **Obj** of type TObject.

By default, the System::TObject::Equals method shows if the addresses corresponding to the object and the parameter are identical. Returns a boolean value that represents the equality between the two addresses.

Note: System::TObject::Equals is supposed to be overridden in user derived classes, in order to provide consumer objects with an equality determining function.

See Also

System::TObject::ClassType

System::TObject::ClassName

System::TObject::ClassParent

System::TObject::FieldAddress

System::TObject::MethodAddress

System::TObject::InstanceSize

System::TObject::MethodName

System::TObject::InheritsFrom

8.1.10.3.6 TGraphic.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetEmpty() = 0;
```

Description

This is GetEmpty, a member of class TGraphic.

8.1.10.3.7 TGraphic.GetHeight Method

Pascal

```
function GetHeight: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetHeight() = 0;
```

Description

This is GetHeight, a member of class TGraphic.

8.1.10.3.8 TGraphic.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; virtual;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TGraphic.

8.1.10.3.9 TGraphic.GetSupportsPartialTransparency Method

Pascal

```
function GetSupportsPartialTransparency: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetSupportsPartialTransparency();
```

Description

This is GetSupportsPartialTransparency, a member of class TGraphic.

8.1.10.3.10 TGraphic.GetTransparent Method

Pascal

```
function GetTransparent: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetTransparent();
```

Description

This is GetTransparent, a member of class TGraphic.

8.1.10.3.11 TGraphic.GetWidth Method

Pascal

```
function GetWidth: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetWidth() = 0;
```

Description

This is GetWidth, a member of class TGraphic.

8.1.10.3.12 TGraphic.LoadFromClipboardFormat Method

Introduces an abstract method that loads the image from a variable in Clipboard format.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); virtual;
abstract;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette) = 0;
```

Description

Each descendant graphic object defines a LoadFromClipboardFormat method that replaces the current graphic image with the graphic indicated by AData, which it loads from the Clipboard.

The format for the new graphic object must be registered with the Clipboard in the RegisterClipboardFormat method, or an exception is raised.

See Also

TClipboard

RegisterClipboardFormat (see page 2819)

8.1.10.3.13 TGraphic.LoadFromFile Method

Loads a graphic image stored in a file.

Pascal

```
procedure LoadFromFile(const Filename: string); virtual;
```

C++

```
virtual __fastcall LoadFromFile(const UnicodeString Filename);
```

Description

LoadFromFile reads the file specified in FileName and loads the data into the graphics object.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> Bitmap1(new Graphics::TBitmap());
  std::auto_ptr<Graphics::TBitmap> Bitmap2(new Graphics::TBitmap());
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1.get());     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20, 20, Bitmap2.get());  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80, 80, Bitmap2.get());
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

LoadFromStream (see page 2756)

8.1.10.3.14 TGraphic.LoadFromStream Method

Introduces an abstract method that loads the image from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream) = 0;
```

Description

Each descendant graphic object defines a LoadFromStream method that loads a graphics object from Stream.

See Also

LoadFromFile (see page 2755)

TStream

8.1.10.3.15 TGraphic.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const UnicodeString Msg);
```

Description

This is Progress, a member of class TGraphic.

8.1.10.3.16 TGraphic.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TGraphic.

8.1.10.3.17 TGraphic.SaveToClipboardFormat Method

Introduces an abstract method that assigns the graphic to a variable in clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); virtual; abstract;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette) = 0;
```

Description

Each descendant graphic object defines a SaveToClipboardFormat method that saves that particular graphic image type to a Clipboard format. An application must have registered the format with the TGraphic (see page 2748) object using the RegisterClipboardFormat method.

See Also

TClipboard

RegisterClipboardFormat (see page 2819)

8.1.10.3.18 TGraphic.SaveToFile Method

Saves a graphics image to a file.

Pascal

```
procedure SaveToFile(const Filename: string); virtual;
```

C++

```
virtual __fastcall SaveToFile(const UnicodeString Filename);
```

Description

SaveToFile writes the graphic to a file, specified by Filename.

C++ Examples:

```
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
```

```
*/

#include <Jpeg.hpp>
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
  std::auto_ptr<TJPEGImage> jp(new TJPEGImage());
  jp->Assign(Image1->Picture->Bitmap);
  jp->SaveToFile("..\\factory.jpg");
  Image2->Stretch = True;
  Image2->Picture->LoadFromFile("..\\factory.jpg");
}
```

Delphi Examples:

```
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

8.1.10.3.19 TGraphic.SaveToStream Method

Introduces an abstract method that saves the graphic image to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream) = 0;
```

Description

Each descendant graphic object defines a SaveToStream method that saves the object to a Stream.

8.1.10.3.20 TGraphic.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetHeight(int Value) = 0;
```

Description

This is SetHeight, a member of class TGraphic.

8.1.10.3.21 TGraphic.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPALETTE); virtual;
```

C++

```
virtual __fastcall SetPalette(HPALETTE Value);
```

Description

This is SetPalette, a member of class TGraphic.

8.1.10.3.22 TGraphic.SetSize Method

Specifies the size of the graphic.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); virtual;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the graphic. This results in better performance than setting the height and width separately.

8.1.10.3.23 TGraphic.SetTransparent Method

Pascal

```
procedure SetTransparent(Value: Boolean); virtual;
```

C++

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

Description

This is SetTransparent, a member of class TGraphic.

8.1.10.3.24 TGraphic.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetWidth(int Value) = 0;
```

Description

This is SetWidth, a member of class TGraphic.

8.1.10.3.25 TGraphic.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TGraphic.

8.1.10.4 TGraphic Properties

The properties of the TGraphic class are listed here.

Public Properties

	Name	Description
R	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.
R	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

8.1.10.4.1 TGraphic.Empty Property

Indicates whether the graphics object contains a graphic.

Pascal

```
property Empty: Boolean;
```

C++

```
__property Boolean Empty;
```

Description

Use Empty to determine whether the graphic is bound to an image. Each descendant graphic object defines its own Get method to access the Empty property.

8.1.10.4.2 TGraphic.Height Property

Specifies the vertical size of the graphic in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to determine the height of the graphic image. Each descendant graphic object defines its own Get and Set methods

to access the Height property.

8.1.10.4.3 TGraphic.Modified Property

Indicates whether the graphics object has been changed or edited.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

If Modified is true, the graphic object has changed. If Modified is false, the graphics object is in the same state as when the object was loaded.

The Modified property indicates only if bitmap objects have been modified. Modified is not true if the graphics object contains an icon or metafile graphic, even if they have been modified.

If the graphics object was modified, save the changes to a file with the SaveToFile (see page 2757) method. The next time the application runs, it can load the graphic from the file with the LoadFromFile (see page 2755) method.

8.1.10.4.4 TGraphic.Palette Property

Indicates the color palette of the graphical image.

Pascal

```
property Palette: HPALETTE;
```

C++

```
__property HPALETTE Palette;
```

Description

Use Palette to get the color palette of a graphical image. If the graphic does not need or does not use a palette, the Palette property is zero.

8.1.10.4.5 TGraphic.PaletteModified Property

Indicates whether the palette has changed.

Pascal

```
property PaletteModified: Boolean;
```

C++

```
__property Boolean PaletteModified;
```

Description

Use PaletteModified to determine if the palette used for a graphical image has changed. PaletteModified is used in the OnChange (see page 2763) event. PaletteModified remains true until whoever is responsible for realizing this new palette (for example, TImage) sets it to false.

8.1.10.4.6 TGraphic.SupportsPartialTransparency Property

Pascal

```
property SupportsPartialTransparency: Boolean;
```

C++

```
__property Boolean SupportsPartialTransparency;
```

Description

This is SupportsPartialTransparency, a member of class TGraphic.

8.1.10.4.7 TGraphic.Transparent Property

Indicates whether the image covers its rectangular area.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Use Transparent to specify that the graphic be drawn transparently. Some descendants of TGraphic (see page 2748) such as TIcon (see page 2768) and TMetafile (see page 2780) are always transparent, so setting the property for those objects does not change their behavior. However, the TBitmap (see page 2628) graphic's drawing is affected by this property. The TImage component sets this property to be the same as its Transparent property to achieve transparent painting.

When the Transparent property is set to True, you can either specify a color as the transparent color or you can use the default color, which is the pixel in the lower left. The specified color is not displayed in the graphic, which lets the background show through. This can be useful in layering and for non-rectangular graphics.

C++ Examples:

```
/*
An examples of TBitmap Color, TGraphic Transparent, TBitmap
TransparentMode, TBitmap TransparentColor and TCanvas Draw.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<Graphics::TBitmap> bitmap(new Graphics::TBitmap);
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap.get());
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap.get());
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
}
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

8.1.10.4.8 TGraphic.Width Property

Determines the maximum width of the graphics object in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Each descendant graphic object defines its own Get and Set methods to access the Width property.

8.1.10.5 TGraphic Events

The events of the TGraphic class are listed here.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

8.1.10.5.1 TGraphic.OnChange Event

Occurs whenever a graphical image changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write a handler to perform an action then the graphical image changes.

8.1.10.5.2 TGraphic.OnProgress Event

Occurs when a graphical image is in the process of changing.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

For certain descendants of TGraphic (see page 2748), OnProgress occurs during slow processes such as loading, storing, or transforming image data. OnProgress allows applications to provide feedback to the user about the progress of the slow process.

Component writers can generate OnProgress events for new descendants of TGraphic (see page 2748) by calling the protected Progress (see page 2756) method. These events are propagated to the TPicture (see page 2812) and TImage objects.

See Also

OnProgress (see page 1098)

OnProgress (see page 2830)

8.1.11 TGraphicsObject Class

TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 2664), TFont (see page 2730), and TPen (see page 2801).

Class Hierarchy

Pascal

```
TGraphicsObject = class(TPersistent);
```

C++

```
class TGraphicsObject : public TPersistent;
```

File

Graphics

Description

Create descendants of TGraphicsObject when drawing an image to a canvas (TCanvas (see page 2673)). Each TGraphicsObject object describes a system graphics object that is selected into the device context of the canvas.

TGraphicsObject introduces a mechanism for responding to changes in the properties of the graphics object.

See Also

TBrush (see page 2664)

TCanvas (see page 2673)

TFont (see page 2730)

TPen (see page 2801)

8.1.11.1 TGraphicsObject Members

The following tables list the members exposed by TGraphicsObject.

Protected Methods

	Name	Description
	Changed (see page 2765)	Generates an OnChange (see page 2768) event when the properties of the graphics object change.
	Lock (see page 2766)	Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.
	Unlock (see page 2767)	Releases the lock on execution threads set by the Lock (see page 2766) method.

Public Events

	Name	Description
	OnChange (see page 2768)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	HandleAllocated (see page 2766)	Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

Public Properties

	Name	Description
	OwnerCriticalSection (see page 2767)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

8.1.11.2 TGraphicsObject Methods

The methods of the TGraphicsObject class are listed here.

Protected Methods

	Name	Description
	Changed (see page 2765)	Generates an OnChange (see page 2768) event when the properties of the graphics object change.
	Lock (see page 2766)	Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.
	Unlock (see page 2767)	Releases the lock on execution threads set by the Lock (see page 2766) method.

Public Methods

	Name	Description
	HandleAllocated (see page 2766)	Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

8.1.11.2.1 TGraphicsObject.Changed Method

Generates an OnChange (see page 2768) event when the properties of the graphics object change.

Pascal

```
procedure Changed; dynamic;
```

C++

```
__fastcall Changed();
```

Description

Descendants of TGraphicsObject (see page 2764) call Changed after making changes to the properties encapsulated by the TGraphicsObject (see page 2764) object.

See Also

OnChange (see page 2768)

8.1.11.2.2 TGraphicsObject.HandleAllocated Method

Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle property is set for the graphics object. HandleAllocated indicates whether the underlying GDI object has been acquired. Use HandleAllocated rather than reading the Handle property. When an application reads the Handle property, TGraphicsObject (see page 2764) descendants automatically create the underlying GDI object and return its handle if the property had not been previously set.

See Also

Handle (see page 2807)

Handle (see page 2741)

Handle (see page 2670)

8.1.11.2.3 TGraphicsObject.Lock Method

Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock in a multithreaded application before using the graphics object in a section of code that would not otherwise be thread-safe. Lock prevents other cooperating threads in the application from using the associated canvas until the Unlock (see page 2767) method is called. Calls to Lock can be nested so that the canvas is not unlocked until the last lock is released.

Because Lock prevents other threads from executing, it can adversely affect performance. Do not call Lock unless there is a danger that another thread might interfere with the drawing.

Note: Lock will do nothing unless the drawing surface sets the OwnerCriticalSection (see page 2767) property. TCanvas (see page 2673) objects automatically set the OwnerCriticalSection (see page 2767) property when they use graphics objects.

See Also

OwnerCriticalSection (see page 2767)

TCanvas (see page 2673)

TThread

Unlock (see page 2767)

8.1.11.2.4 TGraphicsObject.Unlock Method

Releases the lock on execution threads set by the Lock (see page 2766) method.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a thread-sensitive section of code that was started with a call to the Lock (see page 2766) method. When every call to Lock (see page 2766) is balanced by a call to Unlock, other threads are allowed to lock the associated canvas.

See Also

Lock (see page 2766)

TThread

TCanvas (see page 2673)

8.1.11.3 TGraphicsObject Properties

The properties of the TGraphicsObject class are listed here.

Public Properties

	Name	Description
	OwnerCriticalSection (see page 2767)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

8.1.11.3.1 TGraphicsObject.OwnerCriticalSection Property

Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Pascal

```
property OwnerCriticalSection: PRTLCriticalSection;
```

C++

```
__property PRTLCriticalSection OwnerCriticalSection;
```

Description

Use OwnerCriticalSection to obtain information about the thread locking state of the canvas that draws with this graphics object. The canvas sets OwnerCriticalSection when it creates the graphics object. Do not change this property while the canvas is using

the graphics object.

If the graphics object is created on behalf of a drawing surface that is not a canvas, set OwnerCriticalSection to an initialized TRTLCriticalSection that is maintained by the drawing surface.

To use OwnerCriticalSection to block other execution threads, use the Lock (see page 2766) method.

See Also

Lock (see page 2766)

Unlock (see page 2767)

Lock (see page 2694)

8.1.11.4 TGraphicsObject Events

The events of the TGraphicsObject class are listed here.

Public Events

	Name	Description
	OnChange (see page 2768)	Occurs immediately after the graphics object changes.

8.1.11.4.1 TGraphicsObject.OnChange Event

Occurs immediately after the graphics object changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the graphics object changes. The properties of the graphics object reflect the new settings when OnChange occurs.

See Also

Changed (see page 2765)

8.1.12 TIcon Class

TIcon is an encapsulation of a Windows icon.

Class Hierarchy

Pascal

```
TIcon = class(TGraphic);
```

C++

```
class TIcon : public TGraphic;
```

File

Graphics

Description

Use TIcon to represent one of the icons in an image list. Assign (see page 2772) an icon to objects that have an Icon property, such as TForm or TPicture (see page 2812).

Icon objects represent the value loaded from a Windows icon file (.ICO file). Draw (see page 2773) an icon on a canvas, using the Draw (see page 2773) method of the TCanvas (see page 2673) object. Icons do not stretch, so do not use StretchDraw (TCanvas (see page 2673)) with an icon.

Note: TIcon can be used to display the value of a multi-resolution (>16 color) icon. However, it can not support the creation of multi-resolution icons, nor the resizing of the image after it is loaded from a file or stream.

See Also

TCanvas (see page 2673)

Icon

TImageList

TPicture (see page 2812)

8.1.12.1 TIcon Members

The following tables list the members exposed by TIcon.

Protected Methods

	Name	Description
	Changed (see page 2751)	This is Changed, a member of class TGraphic.
	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 2773)	This is Draw, a member of class TIcon.
	DrawTransparent (see page 2752)	This is DrawTransparent, a member of class TGraphic.
	Equals (see page 2752)	This is the overview for the Equals method overload.
	GetEmpty (see page 2773)	This is GetEmpty, a member of class TIcon.
	GetHeight (see page 2773)	This is GetHeight, a member of class TIcon.
	GetPalette (see page 2754)	This is GetPalette, a member of class TGraphic.
	GetSupportsPartialTransparency (see page 2754)	This is GetSupportsPartialTransparency, a member of class TGraphic.
	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 2773)	This is GetWidth, a member of class TIcon.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.
	ReadData (see page 2757)	This is ReadData, a member of class TGraphic.
	SetHeight (see page 2776)	This is SetHeight, a member of class TIcon.
	SetPalette (see page 2759)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 2777)	This is SetTransparent, a member of class TIcon.
	SetWidth (see page 2777)	This is SetWidth, a member of class TIcon.
	WriteData (see page 2760)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	Assign (see page 2772)	Copies an icon image from another TIcon (see page 2768) object.
	Create (see page 2770)	Creates an instance of TIcon (see page 2768).
	Destroy (see page 2772)	Destroys an instance of TIcon (see page 2768).
	HandleAllocated (see page 2774)	Indicates whether the TIcon (see page 2768) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2774)	Prevents applications from loading icons from the clipboard.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromResourceID (see page 2774)	This is LoadFromResourceID, a member of class TIcon.
	LoadFromResourceName (see page 2775)	Loads an icon resource into the icon object.
	LoadFromStream (see page 2775)	Loads the icon from a stream.
	ReleaseHandle (see page 2775)	Releases the Windows GDI object represented by the icon.
	SaveToClipboardFormat (see page 2776)	Prevents applications from saving icons to the clipboard.
	SaveToFile (see page 2757)	Saves a graphics image to a file.
	SaveToStream (see page 2776)	Saves the icon to a stream.
	SetSize (see page 2777)	Specifies the size of the icon.

Public Properties

	Name	Description
	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Handle (see page 2777)	Provides access to the Windows GDI icon handle.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.
	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

8.1.12.2 TIcon.Create Constructor

Creates an instance of TIcon (see page 2768).

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TIcon();
```

Description

Call Create to create an empty icon object. Once an image has been read into the icon from a file or stream, or by assigning another icon object, the icon can be drawn on a canvas or added to an image list. Do not create an icon object for setting the icon property of a form or picture. These objects create a TIcon (see page 2768) object in their own constructors. Instead, use the LoadFromFile (see page 2755), LoadFromStream (see page 2775), or LoadFromClipboardFormat (see page 2774) method on the TIcon (see page 2768) object that you can obtain by reading the Icon property.

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/
```

```
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    std::auto_ptr<TIcon> pi(new Graphics::TIcon);
    AnsiString as;
    pi->LoadFromFile(OpenDialog1->FileName);
    as = ChangeFileExt(OpenDialog1->FileName,".BMP");
    Image1->Width = pi->Width;
    Image1->Height = pi->Height;
    Image1->Canvas->Draw(0, 0, pi.get());
    Image1->Picture->SaveToFile(as);
    ShowMessage(OpenDialog1->FileName + " Saved to " + as);
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
```

See Also

Empty

LoadFromClipboardFormat (see page 2754)

LoadFromFile (see page 2755)

LoadFromStream (see page 2775)

8.1.12.3 TIcon.Destroy Destructor

Destroys an instance of TIcon (see page 2768).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIcon();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIcon (see page 2768) reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

ReleaseHandle (see page 2775)

8.1.12.4 TIcon Methods

The methods of the TIcon class are listed here.

Protected Methods

	Name	Description
	Draw (see page 2773)	This is Draw, a member of class TIcon.
	GetEmpty (see page 2773)	This is GetEmpty, a member of class TIcon.
	GetHeight (see page 2773)	This is GetHeight, a member of class TIcon.
	GetWidth (see page 2773)	This is GetWidth, a member of class TIcon.
	SetHeight (see page 2776)	This is SetHeight, a member of class TIcon.
	SetTransparent (see page 2777)	This is SetTransparent, a member of class TIcon.
	SetWidth (see page 2777)	This is SetWidth, a member of class TIcon.

Public Methods

	Name	Description
	Assign (see page 2772)	Copies an icon image from another TIcon (see page 2768) object.
	HandleAllocated (see page 2774)	Indicates whether the TIcon (see page 2768) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2774)	Prevents applications from loading icons from the clipboard.
	LoadFromResourceID (see page 2774)	This is LoadFromResourceID, a member of class TIcon.
	LoadFromResourceName (see page 2775)	Loads an icon resource into the icon object.
	LoadFromStream (see page 2775)	Loads the icon from a stream.
	ReleaseHandle (see page 2775)	Releases the Windows GDI object represented by the icon.
	SaveToClipboardFormat (see page 2776)	Prevents applications from saving icons to the clipboard.
	SaveToStream (see page 2776)	Saves the icon to a stream.
	SetSize (see page 2777)	Specifies the size of the icon.

8.1.12.4.1 TIcon.Assign Method

Copies an icon image from another TIcon (see page 2768) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy another icon object. Assign copies the icon image from the Source parameter if it is another TIcon (see page 2768) object. Otherwise, Assign calls the inherited method, which copies the icon image from any source object that specifies how to copy to a TIcon (see page 2768) in its AssignTo method.

See Also

AssignTo

8.1.12.4.2 TIcon.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TIcon.

8.1.12.4.3 TIcon.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TIcon.

8.1.12.4.4 TIcon.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TIcon.

8.1.12.4.5 TIcon.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TIcon.

8.1.12.4.6 TIcon.HandleAllocated Method

Indicates whether the TIcon (see page 2768) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 2777) property is set to an HICON value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 2777) property. When an application reads the Handle (see page 2777) property, it automatically creates an icon and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 2777)

8.1.12.4.7 TIcon.LoadFromClipboardFormat Method

Prevents applications from loading icons from the clipboard.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is overridden in TIcon (see page 2768) because the clipboard format is not supported.

See Also

RegisterClipboardFormat (see page 2819)

8.1.12.4.8 TIcon.LoadFromResourceID Method

Pascal

```
procedure LoadFromResourceID(Instance: THandle; ResID: Integer);
```

C++

```
__fastcall LoadFromResourceID(THandle Instance, int ResID);
```

Description

This is LoadFromResourceID, a member of class TIcon.

8.1.12.4.9 TIcon.LoadFromResourceName Method

Loads an icon resource into the icon object.

Pascal

```
procedure LoadFromResourceName(Instance: THandle; const ResName: String);
```

C++

```
__fastcall LoadFromResourceName(THandle Instance, const UnicodeString ResName);
```

Description

LoadFromResourceName loads the specified icon resource from a module's executable file.

Instance is the handle of the module that contains the resource.

ResName is the name of the resource to load.

See Also

LoadFromMimeSource

LoadFromStream (see page 2775)

8.1.12.4.10 TIcon.LoadFromStream Method

Loads the icon from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to read the icon image from a stream. Set the Stream parameter to a stream object that provides access to the memory image of the icon. To load the icon from a .ico file, use the LoadFromFile (see page 2755) method. To load the icon from the Clipboard, use the LoadFromClipboardFormat (see page 2774) method.

See Also

LoadFromClipboardFormat (see page 2754)

LoadFromFile (see page 2755)

SaveToStream (see page 2776)

8.1.12.4.11 TIcon.ReleaseHandle Method

Releases the Windows GDI object represented by the icon.

Pascal

```
function ReleaseHandle: HICON;
```

C++

```
__fastcall HICON ReleaseHandle();
```

Description

Call ReleaseHandle to release the resources used to represent the icon. ReleaseHandle sets the Handle (see page 2777)

property to nil (Delphi) or NULL (C++).

ReleaseHandle informs the TIcon (see page 2768) instance that it is no longer responsible for destroying the icon handle.

See Also

Handle (see page 2777)

8.1.12.4.12 TIcon.SaveToClipboardFormat Method

Prevents applications from saving icons to the clipboard.

Pascal

```
procedure SaveToClipboardFormat(var Format: Word; var Data: THandle; var APalette: HPALETTE);
override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word Format, THandle Data, HPALETTE APalette);
```

Description

SaveToClipboardFormat is overridden in TIcon (see page 2768) because the format is not supported for the Clipboard.

See Also

LoadFromClipboardFormat (see page 2774)

8.1.12.4.13 TIcon.SaveToStream Method

Saves the icon to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to write the icon image to a stream. Specify the Stream parameter as the stream object that receives the memory image of the icon. To write the icon to a .ico file, use the SaveToFile (see page 2757) method. To write the icon to the Clipboard, use the SaveToClipboardFormat (see page 2776) method.

See Also

LoadFromStream (see page 2775)

SaveToClipboardFormat (see page 2757)

SaveToFile (see page 2757)

8.1.12.4.14 TIcon.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TIcon.

8.1.12.4.15 TIcon.SetSize Method

Specifies the size of the icon.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the icon. This results in better performance than setting the height and width separately.

8.1.12.4.16 TIcon.SetTransparent Method

Pascal

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

C++

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

Description

This is SetTransparent, a member of class TIcon.

8.1.12.4.17 TIcon.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TIcon.

8.1.12.5 TIcon Properties

The properties of the TIcon class are listed here.

Public Properties

	Name	Description
	Handle (see page 2777)	Provides access to the Windows GDI icon handle.

8.1.12.5.1 TIcon.Handle Property

Provides access to the Windows GDI icon handle.

Pascal

```
property Handle: HICON;
```

C++

```
__property HICON Handle;
```

Description

Use Handle to specify the icon when using a Windows API function that requires the handle of an icon object. The Handle property is nil (Delphi) or NULL (C++) if the icon has not been loaded.

Call the ReleaseHandle (see page 2775) method before changing the icon image by setting the Handle property. The icon image can also be loaded by assigning another icon object or using the LoadFromClipboardFormat (see page 2774), LoadFromFile (see page 2755), or LoadFromStream (see page 2775) method.

See Also

Assign (see page 2772)

Empty

LoadFromClipboardFormat (see page 2754)

LoadFromFile (see page 2755)

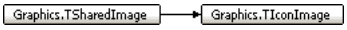
LoadFromStream (see page 2775)

ReleaseHandle (see page 2775)

8.1.13 TIconImage Class

TIconImage is the internal representation of the bitmap image for a TIcon (see page 2768) object.

Class Hierarchy



Pascal

```
TIconImage = class(TSharedImage);
```

C++

```
class TIconImage : public TSharedImage;
```

File

Graphics

Description

TIconImage is used for internal implementation only. It represents the internal image of the bitmap encapsulated by the TIcon (see page 2768) object. All of the data and methods introduced in TIconImage are private. They contain information about the specific icon resource, including pointers to the standard HICON structures defined by Windows. TIconImage is referenced in the constructor of the TIcon (see page 2768) object when the icon image is created.

See Also

TIcon (see page 2768)

8.1.13.1 TIconImage Members

The following tables list the members exposed by TIconImage.

Protected Methods

	Name	Description
	FreeHandle (see page 2779)	This is FreeHandle, a member of class TIconImage.
	Reference (see page 2832)	This is Reference, a member of class TSharedImage.
	Release (see page 2832)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
	RefCount (see page 2832)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
	Destroy (see page 2779)	Destroys the TIconImage (see page 2778) instance and frees its memory.

8.1.13.2 TIconImage.Destroy Destructor

Destroys the TIconImage (see page 2778) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIconImage();
```

Description

You do not need to destroy the icon image. The TIconImage (see page 2778) object is automatically destroyed by the icon object.

See Also

System::TObject::Free

8.1.13.3 TIconImage Methods

The methods of the TIconImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 2779)	This is FreeHandle, a member of class TIconImage.

8.1.13.3.1 TIconImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TIconImage.

8.1.14 TMetafile Class

TMetafile is an encapsulation of the Win32 Enhanced (see page 2793) metafile.

Class Hierarchy

Pascal

```
TMetafile = class(TGraphic);
```

C++

```
class TMetafile : public TGraphic;
```

File

Graphics

Description

TMetafile contains a metafile graphic (EMF file format).

Properties of TMetafile indicate the size and characteristics of the metafile.

To draw a metafile on a canvas, call the Draw (see page 2785) or StretchDraw methods of a TCanvas (see page 2673) object, passing a TMetafile as a parameter. The Enhanced (see page 2793) property determines how the metafile will be stored on disk. If Enhanced (see page 2793) is true, the metafile is stored as an .EMF (Win32 Enhanced (see page 2793) Metafile) file. If Enhanced (see page 2793) is false, the metafile is stored as a .WMF (Windows 3.1 Metafile, with Aldus header).

See Also

TMetafileImage (see page 2799)

Using the Main VCL

8.1.14.1 TMetafile Members

The following tables list the members exposed by TMetafile.

Protected Methods

	Name	Description
	Changed (see page 2751)	This is Changed, a member of class TGraphic.
	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 2785)	This is Draw, a member of class TMetafile.
	DrawTransparent (see page 2752)	This is DrawTransparent, a member of class TGraphic.
	Equals (see page 2752)	This is the overview for the Equals method overload.
	GetEmpty (see page 2786)	This is GetEmpty, a member of class TMetafile.
	GetHeight (see page 2786)	This is GetHeight, a member of class TMetafile.

	GetPalette (see page 2786)	This is GetPalette, a member of class TMetafile.
	GetSupportsPartialTransparency (see page 2754)	This is GetSupportsPartialTransparency, a member of class TGraphic.
	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 2786)	This is GetWidth, a member of class TMetafile.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.
	ReadData (see page 2787)	This is ReadData, a member of class TMetafile.
	ReadEMFStream (see page 2788)	This is ReadEMFStream, a member of class TMetafile.
	ReadWMFStream (see page 2788)	This is ReadWMFStream, a member of class TMetafile.
	SetHeight (see page 2790)	This is SetHeight, a member of class TMetafile.
	SetPalette (see page 2759)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 2790)	This is SetTransparent, a member of class TMetafile.
	SetWidth (see page 2790)	This is SetWidth, a member of class TMetafile.
	TestEMF (see page 2791)	This is TestEMF, a member of class TMetafile.
	WriteData (see page 2791)	This is WriteData, a member of class TMetafile.
	WriteEMFStream (see page 2791)	This is WriteEMFStream, a member of class TMetafile.
	WriteWMFStream (see page 2791)	This is WriteWMFStream, a member of class TMetafile.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	Assign (see page 2785)	Copies an object to the metafile.
	Clear (see page 2785)	Deletes the metafile image.
	Create (see page 2782)	Instantiates a metafile object.
	Destroy (see page 2783)	Destroys an instance of a metafile object.
	HandleAllocated (see page 2786)	Indicates whether the TMetafile (see page 2780) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2787)	Loads a metafile from the Clipboard.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromStream (see page 2787)	Loads the metafile from a stream.
	ReleaseHandle (see page 2788)	Releases the Windows GDI object represented by the metafile.
	SaveToClipboardFormat (see page 2788)	Saves a metafile to a Clipboard format.
	SaveToFile (see page 2789)	Writes the metafile to disk.
	SaveToStream (see page 2789)	Writes the metafile to a stream.
	SetSize (see page 2790)	Specifies the size of the metafile.

Public Properties

	Name	Description
	CreatedBy (see page 2792)	Specifies the name of the author or application used to create the metafile.
	Description (see page 2792)	Provides an optional text description that is embedded in the metafile.
	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Enhanced (see page 2793)	Determines how the metafile will be stored on disk.
	Handle (see page 2793)	Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	Inch (see page 2793)	Returns the units per inch that are used for the metafile's coordinate mapping.
	MMHeight (see page 2793)	Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	MMWidth (see page 2794)	Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.

	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

8.1.14.2 TMetafile.Create Constructor

Instantiates a metafile object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TMetafile();
```

Description

Call Create to instantiate a metafile at runtime.

Create allocates memory for a metafile object, and calls the inherited Create. Then sets the Enhanced (see page 2793) and Transparent (see page 2762) properties to true.

To create a metafile image from scratch, draw the image in a metafile canvas. When the metafile canvas is destroyed, it transfers the image into the metafile object provided to the metafile canvas constructor. After the image is drawn on the canvas and the canvas is destroyed, the image is 'playable' in the metafile object.

C++ Examples:

```
/*
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
*/
#include <memory>       //for STL auto_ptr class

TMetafile *MyMetafile;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static std::auto_ptr<TMetafile> _MyMetafileCleaner(MyMetafile = new TMetafile());
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Brush->Color = clRed;
  canvas->Ellipse(0, 0, 100, 200);
  // ...
  delete canvas;
  Form1->Canvas->Draw(0, -50, MyMetafile); //1 red circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); //1 red circle
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyMetafile == NULL) return;   // click Button1 first!
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Draw(-50, 0, MyMetafile);
  canvas->Brush->Color = clBlue;
  canvas->Ellipse(100, 100, 300, 200);
  // ...
  delete canvas; // TMetaFile canvas paints in the destructor.
  Form1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
}
```

Delphi Examples:

```
{
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
}

var
  MyMetafile: TMetafile;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyMetafile := TMetafile.Create;
  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Brush.Color := clRed;
    Ellipse(0, 0, 100, 200);
    //  ...
  finally
//    Free;
  end;
  Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle }
  PaintBox1.Canvas.Draw(0, -50, MyMetafile); {1 red circle }
end;

{
To add to an existing metafile image, create a metafile
canvas and play the source metafile into the metafile canvas:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyMetafile = nil) then exit;  { click Button1 first! }

  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Draw(-50, 0, MyMetafile);
    Brush.Color := clBlue;
    Ellipse(100, 100, 300, 200);
// ...
  finally
//    Free;
  end;
  Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
  PaintBox1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
end;
```

See Also

Enhanced (see page 2793)

System::TObject::Create

TMetafileCanvas (see page 2794)

TMetafileImage (see page 2799)

Transparent (see page 2762)

8.1.14.3 TMetafile.Destroy Destructor

Destroys an instance of a metafile object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafile();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the TMetafile (see page 2780) reference is not nil before it calls Destroy.

Destroy releases the image, which frees the image and the handle when the reference count on the metafile image is zero. Then Destroy calls the inherited Destroy.

See Also

System::TObject::Free

TMetafileImage (see page 2799)

System::TObject::Destroy

8.1.14.4 TMetafile Methods

The methods of the TMetafile class are listed here.

Protected Methods

	Name	Description
	Draw (see page 2785)	This is Draw, a member of class TMetafile.
	GetEmpty (see page 2786)	This is GetEmpty, a member of class TMetafile.
	GetHeight (see page 2786)	This is GetHeight, a member of class TMetafile.
	GetPalette (see page 2786)	This is GetPalette, a member of class TMetafile.
	GetWidth (see page 2786)	This is GetWidth, a member of class TMetafile.
	ReadData (see page 2787)	This is ReadData, a member of class TMetafile.
	ReadEMFStream (see page 2788)	This is ReadEMFStream, a member of class TMetafile.
	ReadWMFStream (see page 2788)	This is ReadWMFStream, a member of class TMetafile.
	SetHeight (see page 2790)	This is SetHeight, a member of class TMetafile.
	SetTransparent (see page 2790)	This is SetTransparent, a member of class TMetafile.
	SetWidth (see page 2790)	This is SetWidth, a member of class TMetafile.
	TestEMF (see page 2791)	This is TestEMF, a member of class TMetafile.
	WriteData (see page 2791)	This is WriteData, a member of class TMetafile.
	WriteEMFStream (see page 2791)	This is WriteEMFStream, a member of class TMetafile.
	WriteWMFStream (see page 2791)	This is WriteWMFStream, a member of class TMetafile.

Public Methods

	Name	Description
	Assign (see page 2785)	Copies an object to the metafile.
	Clear (see page 2785)	Deletes the metafile image.
	HandleAllocated (see page 2786)	Indicates whether the TMetafile (see page 2780) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2787)	Loads a metafile from the Clipboard.
	LoadFromStream (see page 2787)	Loads the metafile from a stream.
	ReleaseHandle (see page 2788)	Releases the Windows GDI object represented by the metafile.
	SaveToClipboardFormat (see page 2788)	Saves a metafile to a Clipboard format.
	SaveToFile (see page 2789)	Writes the metafile to disk.
	SaveToStream (see page 2789)	Writes the metafile to a stream.
	SetSize (see page 2790)	Specifies the size of the metafile.

8.1.14.4.1 TMetafile.Assign Method

Copies an object to the metafile.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the Enhanced (see page 2793) and Palette (see page 2761) properties of an object to the metafile. If the metafile is nil (Delphi) or NULL (C++), Assign creates one. Otherwise, Assign calls the inherited method. For assignment, metafiles are polymorphic with other TGraphic (see page 2748) objects, and with TPicture (see page 2812).

See Also

Enhanced (see page 2793)

Palette

Assign

Assign (see page 2815)

Graphic (see page 2827)

8.1.14.4.2 TMetafile.Clear Method

Deletes the metafile image.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to release the old metafile image, create a new image, and increment a reference count on it.

See Also

TMetafileImage (see page 2799)

8.1.14.4.3 TMetafile.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TMetafile.

8.1.14.4.4 TMetafile.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TMetafile.

8.1.14.4.5 TMetafile.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TMetafile.

8.1.14.4.6 TMetafile.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TMetafile.

8.1.14.4.7 TMetafile.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TMetafile.

8.1.14.4.8 TMetafile.HandleAllocated Method

Indicates whether the TMetafile (see page 2780) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 2793) property is set to an HMETAFILE (see page 2876) value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 2793) property. When an application reads the Handle (see page 2793) property, it automatically creates a metafile and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 2793)

8.1.14.4.9 TMetafile.LoadFromClipboardFormat Method

Loads a metafile from the Clipboard.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat replaces the current image with the data in the clipboard. LoadFromClipboardFormat ignores the AData and APalette parameters.

See Also

RegisterClipboardFormat (see page 2819)

8.1.14.4.10 TMetafile.LoadFromStream Method

Loads the metafile from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream by specifying the stream from which the metafile is loaded as the value of Stream. LoadFromStream reads a metafile from a stream and replaces the current image with it.

See Also

SaveToStream (see page 2789)

8.1.14.4.11 TMetafile.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); override;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TMetafile.

8.1.14.4.12 TMetafile.ReadEMFStream Method

Pascal

```
procedure ReadEMFStream(Stream: TStream);
```

C++

```
__fastcall ReadEMFStream(TStream Stream);
```

Description

This is ReadEMFStream, a member of class TMetafile.

8.1.14.4.13 TMetafile.ReadWMFStream Method

Pascal

```
procedure ReadWMFStream(Stream: TStream; Length: Longint);
```

C++

```
__fastcall ReadWMFStream(TStream Stream, Longint Length);
```

Description

This is ReadWMFStream, a member of class TMetafile.

8.1.14.4.14 TMetafile.ReleaseHandle Method

Releases the Windows GDI object represented by the metafile.

Pascal

```
function ReleaseHandle: HENHMETAFILE;
```

C++

```
__fastcall HENHMETAFILE ReleaseHandle();
```

Description

Call ReleaseHandle to release the resources used to represent the metafile. ReleaseHandle sets the Handle (see page 2793) property to nil. NULL.

Call ReleaseHandle before setting the Handle (see page 2793) property, so that the resources associated with the metafile are not lost.

See Also

Handle (see page 2793)

8.1.14.4.15 TMetafile.SaveToClipboardFormat Method

Saves a metafile to a Clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the metafile to a Clipboard format. The resulting values can then be copied to the Windows clipboard using the clipboard's SetAsHandle method.

The metafile's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the metafile can be saved, an application must have registered the format with the metafile object using the RegisterClipboardFormat method.

To save a metafile to a file, call SaveToFile (see page 2789).

See Also

LoadFromClipboardFormat (see page 2787)

SaveToFile (see page 2789)

SetAsHandle

8.1.14.4.16 TMetafile.SaveToFile Method

Writes the metafile to disk.

Pascal

```
procedure SaveToFile(const Filename: String); override;
```

C++

```
virtual __fastcall SaveToFile(const UnicodeString Filename);
```

Description

Use SaveToFile to save the metafile to a file provided by the Filename parameter. To load a metafile from a file, call the LoadFromFile (see page 2755) method.

See Also

SaveToStream (see page 2789)

LoadFromFile (see page 2755)

8.1.14.4.17 TMetafile.SaveToStream Method

Writes the metafile to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save the metafile to the stream specified by the value of the Stream parameter.

See Also

LoadFromStream (see page 2787)

SaveToFile (see page 2789)

8.1.14.4.18 TMetafile.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TMetafile.

8.1.14.4.19 TMetafile.SetSize Method

Specifies the size of the metafile.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the metafile. This results in better performance than setting the height and width separately.

8.1.14.4.20 TMetafile.SetTransparent Method

Pascal

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

C++

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

Description

This is SetTransparent, a member of class TMetafile.

8.1.14.4.21 TMetafile.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TMetafile.

8.1.14.4.22 TMetafile.TestEMF Method

Pascal

```
function TestEMF(Stream: TStream): Boolean;
```

C++

```
__fastcall Boolean TestEMF(TStream Stream);
```

Description

This is TestEMF, a member of class TMetafile.

8.1.14.4.23 TMetafile.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); override;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TMetafile.

8.1.14.4.24 TMetafile.WriteEMFStream Method

Pascal

```
procedure WriteEMFStream(Stream: TStream);
```

C++

```
__fastcall WriteEMFStream(TStream Stream);
```

Description

This is WriteEMFStream, a member of class TMetafile.

8.1.14.4.25 TMetafile.WriteWMFStream Method

Pascal

```
procedure WriteWMFStream(Stream: TStream);
```

C++

```
__fastcall WriteWMFStream(TStream Stream);
```

Description

This is WriteWMFStream, a member of class TMetafile.

8.1.14.5 TMetafile Properties

The properties of the TMetafile class are listed here.

Public Properties

	Name	Description
	CreatedBy (see page 2792)	Specifies the name of the author or application used to create the metafile.
	Description (see page 2792)	Provides an optional text description that is embedded in the metafile.
	Enhanced (see page 2793)	Determines how the metafile will be stored on disk.
	Handle (see page 2793)	Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.
	Inch (see page 2793)	Returns the units per inch that are used for the metafile's coordinate mapping.
	MMHeight (see page 2793)	Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	MMWidth (see page 2794)	Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

8.1.14.5.1 TMetafile.CreatedBy Property

Specifies the name of the author or application used to create the metafile.

Pascal

```
property CreatedBy: String;
```

C++

```
__property UnicodeString CreatedBy;
```

Description

Use CreatedBy to set the string to an optional name of the author or application used to create the metafile. To set the CreatedBy string of a new metafile, call the TMetaFileCanvas CreateWithComment constructor (Delphi) or the TMetafileCanvas (see page 2794) constructor that takes a comment (C++).

See Also

TMetafileCanvas (see page 2794)

CreateWithComment (see page 2798)

Create_c

8.1.14.5.2 TMetafile.Description Property

Provides an optional text description that is embedded in the metafile.

Pascal

```
property Description: String;
```

C++

```
__property UnicodeString Description;
```

Description

Set the Description string of a new metafile by calling the TMetaFileCanvas CreateWithComment constructor (Delphi) or the TMetafileCanvas (see page 2794) constructor that takes a comment (C++)..

See Also

TMetafileCanvas (see page 2794)

CreateWithComment (see page 2798)

Create_c

8.1.14.5.3 TMetafile.Enhanced Property

Determines how the metafile will be stored on disk.

Pascal

```
property Enhanced: Boolean;
```

C++

```
__property Boolean Enhanced;
```

Description

Use Enhanced to determine how the metafile is stored on disk. If Enhanced is true, the metafile is stored as an .EMF (Win32 Enhanced Metafile). If Enhanced is false, the metafile is stored as a .WMF (Windows 3.1 Metafile, with Aldus header).

The in-memory format is always EMF. WMF has very limited capabilities; storing as WMF will lose information that would be retained by EMF. This property is set to match the metafile type when loaded from a stream or file. If loaded as WMF, then save as WMF.

8.1.14.5.4 TMetafile.Handle Property

Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.

Pascal

```
property Handle: HENHMETAFILE;
```

C++

```
__property HENHMETAFILE Handle;
```

Description

Use Handle when calling a Windows API function that requires the handle of a metafile object. Pass the handle from the Handle property of the metafile object to the function.

See Also

DestroyMetaPict (see page 6055)

8.1.14.5.5 TMetafile.Inch Property

Returns the units per inch that are used for the metafile's coordinate mapping.

Pascal

```
property Inch: Word;
```

C++

```
__property Word Inch;
```

Description

Use Inch to find the units per inch assumed by a WMF metafile, and to alter scale when writing as WMF, but otherwise this property is obsolete. Enhanced (see page 2793) metafiles maintain complete scale information internally. For example, if the metafile was created in a Twips coordinate system (using MM_TWIPS mapping), the value of Inch is 1440.

8.1.14.5.6 TMetafile.MMHeight Property

Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

Pascal

```
property MMHeight: Integer;
```

C++

```
__property int MMHeight;
```

Description

MMHeight is used for a more accurate reading of the vertical size of the graphic. The Height (see page 2760) property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the dimensions in millimeters with MMHeight.

See Also

MMWidth (see page 2794)

8.1.14.5.7 TMetafile.MMWidth Property

Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

Pascal

```
property MMWidth: Integer;
```

C++

```
__property int MMWidth;
```

Description

MMWidth is used for a more accurate reading of the horizontal size of the graphic. The Width (see page 2763) property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the dimensions in millimeters with MMWidth.

The MMWidth property is always in screen device pixel units.

See Also

MMHeight (see page 2793)

8.1.15 TMetafileCanvas Class

TMetafileCanvas allows applications to create metafile images from scratch.

Class Hierarchy

Pascal

```
TMetafileCanvas = class(TCanvas);
```

C++

```
class TMetafileCanvas : public TCanvas;
```

File

Graphics

Description

Use TMetafileCanvas as a surface on which to draw a metafile image. When the canvas is destroyed, it transfers the image into the metafile object provided to the canvas constructor. After the image is drawn on the canvas and the canvas is destroyed, the image is 'playable' in the metafile object.

See Also

TMetafile (see page 2780)

8.1.15.1 TMetafileCanvas Members

The following tables list the members exposed by TMetafileCanvas.

Protected Methods

	Name	Description
	Changed (see page 2681)	This is Changed, a member of class TCanvas.
	Changing (see page 2682)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 2685)	This is CreateHandle, a member of class TCanvas.
	RequiredState (see page 2702)	This is RequiredState, a member of class TCanvas.

Public Events

	Name	Description
	OnChange (see page 2729)	Occurs when the image has just changed.
	OnChanging (see page 2730)	Occurs just before a change is made to the image.

Public Methods

	Name	Description
	Arc (see page 2679)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	BrushCopy (see page 2680)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 2682)	Draws a closed figure represented by the intersection of a line and an ellipse.
	CopyRect (see page 2683)	Copies part of an image from another canvas into the canvas.
	Create (see page 2796)	Creates an instance of TMetafileCanvas (see page 2794) and acquires a handle to a metafile device context.
	CreateWithComment (see page 2798)	Creates an instance of TMetafileCanvas (see page 2794) and acquires a handle to a metafile device context with a comment string.
	Destroy (see page 2799)	Destroys an instance of the TMetafileCanvas (see page 2794) after transferring the image to the TMetafile (see page 2780) object that was passed to the constructor.
	Draw (see page 2685)	This is the overview for the Draw method overload.
	DrawFocusRect (see page 2687)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 2688)	This is the overview for the Ellipse method overload.
	FillRect (see page 2689)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 2690)	Fills an area of the canvas using the current brush.
	FrameRect (see page 2692)	Draws a rectangle using the Brush (see page 2716) of the canvas to draw the border.
	HandleAllocated (see page 2693)	Indicates whether the TCanvas (see page 2673) object has acquired a handle to a device context.
	LineTo (see page 2693)	Draws a line on the canvas from PenPos (see page 2727) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 2694)	Prevents other threads from drawing on the canvas.
	MoveTo (see page 2695)	Changes the current drawing position to the point (X,Y).
	Pie (see page 2695)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.

	PolyBezier (see page 2696)	Draws a set of Bezier curves.
	PolyBezierTo (see page 2697)	Draws a set of Bezier curves and updates the value of PenPos (see page 2727).
	Polygon (see page 2697)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 2698)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 2700)	This is the overview for the Rectangle method overload.
	Refresh (see page 2701)	Deselects the Pen (see page 2725), Brush (see page 2716), and Font (see page 2721) from the device context.
	RoundRect (see page 2702)	This is the overview for the RoundRect method overload.
	StretchDraw (see page 2704)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 2705)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 2706)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 2707)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 2727) to the end of the string.
	TextRect (see page 2713)	This is the overview for the TextRect method overload.
	TextWidth (see page 2714)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 2715)	Locks the canvas if it is currently unlocked.
	Unlock (see page 2716)	Decreases the LockCount (see page 2725) property, allowing other threads to execute when LockCount (see page 2725) reaches 0.

Public Properties

	Name	Description
R	CanvasOrientation (see page 2718)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 2718)	Read-only property that specifies the boundaries of the clipping rectangle.
	Handle (see page 2722)	Specifies the handle for this canvas.
R	LockCount (see page 2725)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 2727)	Specifies the current drawing position of the Pen (see page 2725).
	Pixels (see page 2727)	Specifies the color of the pixels within the current ClipRect (see page 2718).
	TextFlags (see page 2728)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 2716)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 2719)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 2721)	Specifies the font to use when writing text on the image.
	Pen (see page 2725)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

8.1.15.2 TMetafileCanvas.Create Constructor

Creates an instance of TMetafileCanvas (see page 2794) and acquires a handle to a metafile device context.

Pascal

```
constructor Create(AMetafile: TMetafile; ReferenceDevice: HDC);
```

C++

```
__fastcall TMetafileCanvas(TMetafile AMetafile, HDC ReferenceDevice);
```

Description

Call Create to create an instantiate a TMetafileCanvas (see page 2794) object. Pass in the associated metafile whose image is about to be drawn as the AMetafile parameter. Pass in a handle to a device context on which to base the metafile's device context as the ReferenceDevice parameter. In most cases, this is the device context for a window on which the image is normally drawn.

Create sets the size of the TMetafile (see page 2780) object from ReferenceDevice if it does not already have the MMHeight and MMWidth properties set. Create then creates a metafile device context, and sets the Handle (see page 2722) property to its handle. All subsequent drawing methods will draw to the metafile device context.

When the TMetafileCanvas (see page 2794) is destroyed, The CreatedBy and Description strings will be used to set the properties of the same name in the TMetafile (see page 2780) object.

C++ Examples:

```
/*
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
*/
#include <memory>       //for STL auto_ptr class

TMetafile *MyMetafile;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static std::auto_ptr<TMetafile> _MyMetafileCleaner(MyMetafile = new TMetafile());
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Brush->Color = clRed;
  canvas->Ellipse(0, 0, 100, 200);
  // ...
  delete canvas;
  Form1->Canvas->Draw(0, -50, MyMetafile); //1 red circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); //1 red circle
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyMetafile == NULL) return;   // click Button1 first!
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Draw(-50, 0, MyMetafile);
  canvas->Brush->Color = clBlue;
  canvas->Ellipse(100, 100, 300, 200);
  // ...
  delete canvas; // TMetaFile canvas paints in the destructor.
  Form1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
}
```

Delphi Examples:

```
{
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
}

var
  MyMetafile: TMetafile;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyMetafile := TMetafile.Create;
  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Brush.Color := clRed;
    Ellipse(0, 0, 100, 200);
    //  ...
  finally
//    Free;
```

```
    end;
    Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle }
    PaintBox1.Canvas.Draw(0, -50, MyMetafile); {1 red circle }
  end;

  {
  To add to an existing metafile image, create a metafile
  canvas and play the source metafile into the metafile canvas:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (MyMetafile = nil) then exit;  { click Button1 first! }

    with TMetafileCanvas.Create(MyMetafile, 0) do
    try
      Draw(-50, 0, MyMetafile);
      Brush.Color := clBlue;
      Ellipse(100, 100, 300, 200);
  // ...
    finally
  //    Free;
    end;
    Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
    PaintBox1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
  end;
```

See Also

CreateWithComment (see page 2798)

8.1.15.3 TMetafileCanvas.CreateWithComment Constructor

Creates an instance of TMetafileCanvas (see page 2794) and acquires a handle to a metafile device context with a comment string.

Pascal

```
constructor CreateWithComment(AMetafile: TMetafile; ReferenceDevice: HDC; const CreatedBy:
String; const Description: String);
```

C++

```
__fastcall TMetafileCanvas(TMetafile AMetafile, HDC ReferenceDevice, const UnicodeString
CreatedBy, const UnicodeString Description);
```

Description

Call CreateWithComment to create an instantiate a TMetafileCanvas (see page 2794) object for a metafile that will receive the CreatedBy and Description strings. CreateWithComment sets the size of the TMetafile (see page 2780) object from the ReferenceDevice if it does not already have the MMHeight and MMWidth properties set. Create (see page 2796) then creates a metafile device context using the CreatedBy and Description strings as a description, and sets the Handle (see page 2722) property to its handle. All subsequent drawing methods will draw to the metafile device context.

When the TMetafileCanvas (see page 2794) is destroyed, the CreatedBy and Description strings are used to set the properties of the same name in the TMetafile (see page 2780) object.

See Also

Description (see page 2792)

8.1.15.4 TMetafileCanvas.Destroy Destructor

Destroys an instance of the TMetafileCanvas (see page 2794) after transferring the image to the TMetafile (see page 2780) object that was passed to the constructor.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafileCanvas();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMetafileCanvas (see page 2794) reference is not nil, and only then calls Destroy.

Destroy sets the Handle (see page 2722) property of the TMetafile (see page 2780) object that was passed to the constructor before freeing the memory associated with the object.

See Also

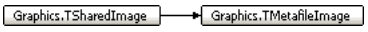
System::TObject::Free

Handle (see page 2793)

8.1.16 TMetafileImage Class

TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 2780) object.

Class Hierarchy



Pascal

```
TMetafileImage = class(TSharedImage);
```

C++

```
class TMetafileImage : public TSharedImage;
```

File

Graphics

Description

TMetafileImage is used for internal implementation only. It represents the internal image of the metafile encapsulated by the TMetafile (see page 2780) object. All of the data and methods introduced in TMetafileImage are private. They contain information about the specific metafile image, including pointers to the standard HENHMETAFILE (see page 2875) (enhanced metafile handle) and HPALETTE (handle to palette) structures defined by Windows.

See Also

TMetafile (see page 2780)

TMetafileCanvas (see page 2794)

8.1.16.1 TMetafileImage Members

The following tables list the members exposed by TMetafileImage.

Protected Methods

	Name	Description
	FreeHandle (see page 2800)	This is FreeHandle, a member of class TMetafileImage.
	Reference (see page 2832)	This is Reference, a member of class TSharedImage.
	Release (see page 2832)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
	RefCount (see page 2832)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
	Destroy (see page 2800)	Disposes of the instance of a metafile image.

8.1.16.2 TMetafileImage.Destroy Destructor

Disposes of the instance of a metafile image.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafileImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMetafileImage (see page 2799) reference is not nil before it calls Destroy.

Destroy frees the metafile handle and palette before calling the inherited Destroy.

See Also

System::TObject::Destroy

System::TObject::Free

8.1.16.3 TMetafileImage Methods

The methods of the TMetafileImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 2800)	This is FreeHandle, a member of class TMetafileImage.

8.1.16.3.1 TMetafileImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TMetafileImage.

8.1.17 TPen Class

TPen is used to draw lines or outline shapes on a canvas.

Class Hierarchy

Pascal

```
TPen = class(TGraphicsObject);
```

C++

```
class TPen : public TGraphicsObject;
```

File

Graphics

Description

Use TPen to describe the attributes of a Windows pen when drawing an image to a canvas (TCanvas (see page 2673)). TPen encapsulates the pen properties that are selected into the device context of the canvas.

See Also

TPen

TCanvas (see page 2673)

Using the Main VCL

8.1.17.1 TPen Members

The following tables list the members exposed by TPen.

Protected Methods

	Name	Description
	Changed (see page 2765)	Generates an OnChange (see page 2768) event when the properties of the graphics object change.
	GetColor (see page 2804)	This is GetColor, a member of class TPen.
	GetHandle (see page 2804)	This is GetHandle, a member of class TPen.
	GetStyle (see page 2804)	This is GetStyle, a member of class TPen.
	GetWidth (see page 2804)	This is GetWidth, a member of class TPen.
	Lock (see page 2766)	Blocks other execution threads from using the associated canvas until the Unlock (see page 2767) method is called.
	SetColor (see page 2805)	This is SetColor, a member of class TPen.
	SetHandle (see page 2805)	This is SetHandle, a member of class TPen.
	SetMode (see page 2805)	This is SetMode, a member of class TPen.
	SetStyle (see page 2805)	This is SetStyle, a member of class TPen.

	SetWidth (see page 2805)	This is SetWidth, a member of class TPen.
	Unlock (see page 2767)	Releases the lock on execution threads set by the Lock (see page 2766) method.

Public Events

	Name	Description
	OnChange (see page 2768)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	Assign (see page 2803)	Assigns the properties of the pen from another TPen (see page 2801) object.
	Create (see page 2802)	Creates an instance of TPen (see page 2801).
	Destroy (see page 2802)	Destroys an instance of TPen (see page 2801).
	HandleAllocated (see page 2766)	Indicates whether the TGraphicsObject (see page 2764) object has acquired a handle to the underlying GDI object.

Public Properties

	Name	Description
	Handle (see page 2807)	Specifies the Windows pen object handle.
	OwnerCriticalSection (see page 2767)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Published Properties

	Name	Description
	Color (see page 2806)	Determines the color used to draw lines on the canvas.
	Mode (see page 2808)	Determines how the pen draws lines on the canvas.
	Style (see page 2808)	Determines the style in which the pen draws lines.
	Width (see page 2809)	Specifies the width of the pen in pixels.

8.1.17.2 TPen.Create Constructor

Creates an instance of TPen (see page 2801).

Pascal

```
constructor Create;
```

C++

```
__fastcall TPen();
```

Description

Call Create to create a TPen (see page 2801) object for assigning the pen property of a canvas. The pen for the canvas is created automatically in the constructor for the TCanvas (see page 2673) object.

Create allocates memory for the TPen (see page 2801) object and initializes the Mode (see page 2808) property to pmCopy.

See Also

Mode (see page 2808)

Pen (see page 2725)

8.1.17.3 TPen.Destroy Destructor

Destroys an instance of TPen (see page 2801).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPen();
```

Description

Do not call Destroy directly from an application. Instead, call Free. Free verifies that the TPen (see page 2801) reference is not nil, and only then calls Destroy.Destroy frees the resource identified by the Handle (see page 2807) property.

See Also

System::TObject::Free

8.1.17.4 TPen Methods

The methods of the TPen class are listed here.

Protected Methods

	Name	Description
	GetColor (see page 2804)	This is GetColor, a member of class TPen.
	GetHandle (see page 2804)	This is GetHandle, a member of class TPen.
	GetStyle (see page 2804)	This is GetStyle, a member of class TPen.
	GetWidth (see page 2804)	This is GetWidth, a member of class TPen.
	SetColor (see page 2805)	This is SetColor, a member of class TPen.
	SetHandle (see page 2805)	This is SetHandle, a member of class TPen.
	SetMode (see page 2805)	This is SetMode, a member of class TPen.
	SetStyle (see page 2805)	This is SetStyle, a member of class TPen.
	SetWidth (see page 2805)	This is SetWidth, a member of class TPen.

Public Methods

	Name	Description
	Assign (see page 2803)	Assigns the properties of the pen from another TPen (see page 2801) object.

8.1.17.4.1 TPen.Assign Method

Assigns the properties of the pen from another TPen (see page 2801) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

If the Source parameter identifies another TPen (see page 2801) object, Assign copies the Color (see page 2806), Width (see page 2809), Style (see page 2808), and Mode (see page 2808) properties of the source pen. Otherwise, the inherited method copies the properties from any object that can write to a TPen (see page 2801) object in its AssignTo method.

See Also

Color (see page 2806)

Mode (see page 2808)

Style (see page 2808)

AssignTo

Width (see page 2809)

8.1.17.4.2 TPen.GetColor Method

Pascal

```
function GetColor: TColor;
```

C++

```
__fastcall TColor GetColor();
```

Description

This is GetColor, a member of class TPen.

8.1.17.4.3 TPen.GetHandle Method

Pascal

```
function GetHandle: HPen;
```

C++

```
__fastcall HPen GetHandle();
```

Description

This is GetHandle, a member of class TPen.

8.1.17.4.4 TPen.GetStyle Method

Pascal

```
function GetStyle: TPenStyle;
```

C++

```
__fastcall TPenStyle GetStyle();
```

Description

This is GetStyle, a member of class TPen.

8.1.17.4.5 TPen.GetWidth Method

Pascal

```
function GetWidth: Integer;
```

C++

```
__fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TPen.

8.1.17.4.6 TPen.SetColor Method

Pascal

```
procedure SetColor(Value: TColor);
```

C++

```
__fastcall SetColor(TColor Value);
```

Description

This is SetColor, a member of class TPen.

8.1.17.4.7 TPen.SetHandle Method

Pascal

```
procedure SetHandle(Value: HPen);
```

C++

```
__fastcall SetHandle(HPen Value);
```

Description

This is SetHandle, a member of class TPen.

8.1.17.4.8 TPen.SetMode Method

Pascal

```
procedure SetMode(Value: TPenMode);
```

C++

```
__fastcall SetMode(TPenMode Value);
```

Description

This is SetMode, a member of class TPen.

8.1.17.4.9 TPen.SetStyle Method

Pascal

```
procedure SetStyle(Value: TPenStyle);
```

C++

```
__fastcall SetStyle(TPenStyle Value);
```

Description

This is SetStyle, a member of class TPen.

8.1.17.4.10 TPen.SetWidth Method

Pascal

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

C++

```
__fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TPen.

8.1.17.5 TPen Properties

The properties of the TPen class are listed here.

Public Properties

	Name	Description
	Handle (see page 2807)	Specifies the Windows pen object handle.

Published Properties

	Name	Description
	Color (see page 2806)	Determines the color used to draw lines on the canvas.
	Mode (see page 2808)	Determines how the pen draws lines on the canvas.
	Style (see page 2808)	Determines the style in which the pen draws lines.
	Width (see page 2809)	Specifies the width of the pen in pixels.

8.1.17.5.1 TPen.Color Property

Determines the color used to draw lines on the canvas.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Set Color to change the color used to draw lines or outline shapes. The way the color is used by the pen depends on the Mode (see page 2808) and Style (see page 2808) properties.

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  switch (random(5))
  {
```

```
      case 0: Canvas->Pen->Style = psSolid; break;
      case 1: Canvas->Pen->Style = psDash; break;
      case 2: Canvas->Pen->Style = psDot; break;
      case 3: Canvas->Pen->Style = psDashDot; break;
      case 4: Canvas->Pen->Style = psDashDotDot; break;
    }
    Canvas->Rectangle(x, y, x + random(400), y + random(400));
  }
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

Mode (see page 2808)

Style (see page 2808)

8.1.17.5.2 TPen.Handle Property

Specifies the Windows pen object handle.

Pascal

```
property Handle: HPen;
```

C++

```
__property HPen Handle;
```

Description

The Handle property provides access to the Windows GDI object handle. TPen (see page 2801) encapsulates the Windows pen object (HPEN). Use Handle when making a Windows API function call that requires the handle of a pen object. Handle is the Windows handle currently selected into the device context of the canvas.

See Also

Handle (see page 2722)

8.1.17.5.3 TPen.Mode Property

Determines how the pen draws lines on the canvas.

Pascal

```
property Mode: TPenMode;
```

C++

```
__property TPenMode Mode;
```

Description

Set Mode to determine how the color of the pen interacts with the color on the canvas.

See Also

Color (see page 2806)

TCanvas (see page 2673)

8.1.17.5.4 TPen.Style Property

Determines the style in which the pen draws lines.

Pascal

```
property Style: TPenStyle;
```

C++

```
__property TPenStyle Style;
```

Description

Use Style to draw a dotted or dashed line, or to omit the line that appears as a frame around shapes.

Note: Only the psInsideFrame style will produce a dithered color to match a Color (see page 2806) property that is not in the color table. All others choose the closest color from the Windows color table.

Note: Dotted or dashed pen styles are not available when the Width (see page 2809) property is not 1.

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
```

```
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  switch (random(5))
  {
    case 0: Canvas->Pen->Style = psSolid; break;
    case 1: Canvas->Pen->Style = psDash; break;
    case 2: Canvas->Pen->Style = psDot; break;
    case 3: Canvas->Pen->Style = psDashDot; break;
    case 4: Canvas->Pen->Style = psDashDotDot; break;
  }
  Canvas->Rectangle(x, y, x + random(400), y + random(400));
}
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

Color (see page 2806)

Width (see page 2809)

8.1.17.5.5 TPen.Width Property

Specifies the width of the pen in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use Width to give the line greater weight. If you attempt to set Width to a value less than 0, the new value is ignored.

Note: The value of Width influences which values of Style (see page 2808) are valid.

C++ Examples:

```
/*
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  Canvas->Pen->Width = random(7);
  int Dx = random(400);
  int Dy = random(400);
  Canvas->RoundRect(x, y, x + Dx, y + Dy, Dx/2, Dy/2);
}
```

Delphi Examples:

```
{
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
}
var
  X, Y, DX, DY: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  Canvas.Pen.Width := Random(7);
  DX := Random(400);
  DY := Random(400);
  Canvas.RoundRect(X, Y, X + DX, Y + DY, DX div 2, DY div 2);
end;
```

See Also

Style (see page 2808)

8.1.18 TPenRecall Class

TPenRecall saves and restores the properties of a TPen (see page 2801) object.

Class Hierarchy

Pascal

```
TPenRecall = class(TRecall);
```

C++

```
class TPenRecall : public TRecall;
```

File

Graphics

Description

Use TPenRecall to store the current state of a TPen (see page 2801) object. You are then free to change the current state of the pen object. To restore the pen object to its original state, all that is necessary is to destroy the TPenRecall object and the pen object is automatically restored to the saved property values.

You can update the TPenRecall instance to reflect the current properties of the TPen (see page 2801) object by calling the Store method. You can prevent the TPenRecall destructor from updating the TPen (see page 2801) object by calling the Forget method.

See Also

TRecall

TPen (see page 2801)

Using Pens

8.1.18.1 TPenRecall Members

The following tables list the members exposed by TPenRecall.

Public Methods

	Name	Description
	Create (see page 2811)	Instantiates a TPenRecall (see page 2811) object.

8.1.18.2 TPenRecall.Create Constructor

Instantiates a TPenRecall (see page 2811) object.

Pascal

```
constructor Create(APen: TPen);
```

C++

```
__fastcall TPenRecall(TPen APen);
```

Description

Call Create to instantiate a pen recall object at runtime. Create saves the current properties of the TPen (see page 2801) object specified by APen, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of APen and save them, replacing the values saved by this constructor. When you destroy the TPenRecall (see page 2811) instance, the most recently-saved properties are assigned to APen, restoring it to a previous state.

See Also

Destroy

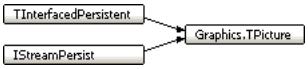
Reference

Store

8.1.19 TPicture Class

TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.

Class Hierarchy



Pascal

```
TPicture = class(TInterfacedPersistent, IStreamPersist);
```

C++

```
class TPicture : public TInterfacedPersistent, public IStreamPersist;
```

File

Graphics

Description

TPicture is a TGraphic (see page 2748) container, used to hold a graphic, the type of which is specified in the Graphic (see page 2827) property. It is used in place of a TGraphic (see page 2748) if the graphic can be of any TGraphic (see page 2748) class. LoadFromFile (see page 2818) and SaveToFile (see page 2823) are polymorphic. For example, if the TPicture is holding an Icon (see page 2828), it is valid to LoadFromFile (see page 2818) a bitmap file, where the class TIcon (see page 2768) can only read .ICO files.

If the TPicture contains a bitmap graphic, the Bitmap (see page 2825) property specifies the graphic. If the TPicture contains an icon graphic, the Icon (see page 2828) property specifies the graphic. If the TPicture contains a metafile graphic, the Metafile (see page 2828) property specifies the graphic.

The properties of TPicture indicate the type of graphic that the picture object contains, and its size. The methods of TPicture are used to load, save, and manipulate graphics.

To load or save a picture to the Clipboard, use the Assign (see page 2815) method of a TClipboard object.

To draw a picture on a canvas, call the Draw or StretchDraw methods of a TCanvas (see page 2673) object, passing the Graphic (see page 2827) property of a TPicture as a parameter.

See Also

TCanvas (see page 2673)

TClipboard

8.1.19.1 TPicture Members

The following tables list the members exposed by TPicture.

Protected Methods

	Name	Description
	AssignTo (see page 2816)	Copies the properties of an object to a destination object.
	Changed (see page 2817)	This is Changed, a member of class TPicture.
	DefineProperties (see page 2817)	Provides an interface for a method that reads and writes otherwise unpublished data.
	LoadFromStream (see page 2819)	Reads the picture from a Stream.
	Progress (see page 2819)	This is Progress, a member of class TPicture.
	SaveToStream (see page 2824)	Saves the picture to a stream.

Public Events

	Name	Description
	OnChange (see page 2830)	Occurs when the specific graphics item encapsulated by the picture object changes.
	OnProgress (see page 2830)	Occurs periodically during slow operations that affect the graphic.

Public Methods

	Name	Description
	Assign (see page 2815)	Copies one object to another by copying the contents of that object to the other.
	Create (see page 2814)	Creates a TPicture (see page 2812) object.
	Destroy (see page 2814)	Destroys an instance of a picture object.
	LoadFromClipboardFormat (see page 2818)	Reads the picture from the handle provided in the given Clipboard format.
	LoadFromFile (see page 2818)	Reads the file specified in Filename and loads the data into the TPicture (see page 2812) object.
	RegisterClipboardFormat (see page 2819)	Registers a new TGraphic (see page 2748) class for use in the LoadFromClipboardFormat (see page 2818) method.
	RegisterFileFormat (see page 2821)	Registers a new TGraphic (see page 2748) class for use in LoadFromFile (see page 2818).
	RegisterFileFormatRes (see page 2822)	Registers a new TGraphic (see page 2748) class for use in the LoadFromFile (see page 2818) method.
	SaveToClipboardFormat (see page 2822)	Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).
	SaveToFile (see page 2823)	Writes the picture to disk.
	SupportsClipboardFormat (see page 2824)	Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 2818) method.
	UnregisterGraphicClass (see page 2825)	This is UnregisterGraphicClass, a member of class TPicture.

Public Properties

	Name	Description
	Bitmap (see page 2825)	Specifies the contents of the picture object as a bitmap graphic (.BMP file format).
	Graphic (see page 2827)	Specifies the graphic that the picture contains.
	Height (see page 2828)	Specifies the vertical size (in pixels) of the graphic.
	Icon (see page 2828)	Specifies the contents of the TPicture (see page 2812) object as an icon graphic (.ICO file format).
	Metafile (see page 2828)	Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).
	PictureAdapter (see page 2829)	Represents an OLE interface for the picture.
	Width (see page 2829)	Specifies the horizontal size (in pixels) of the picture.

8.1.19.2 TPicture.Create Constructor

Creates a TPicture (see page 2812) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TPicture();
```

Description

Call Create to create an instance of TPicture (see page 2812) at runtime. Create allocates the memory for the picture object and initializes its internal formats. After creating the TPicture (see page 2812) object, use the LoadFromClipboardFormat (see page 2818) or LoadFromFile (see page 2818) method to assign a graphical image to the picture, so that it is available to the application.

See Also

LoadFromClipboardFormat (see page 2818)

LoadFromFile (see page 2818)

8.1.19.3 TPicture.Destroy Destructor

Destroys an instance of a picture object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPicture();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPicture (see page 2812) reference is not nil before it calls Destroy.

Destroy frees the graphic image it contains before calling the inherited Destroy.

See Also

System::TObject::Free

System::TObject::Destroy

8.1.19.4 TPicture Methods

The methods of the TPicture class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 2816)	Copies the properties of an object to a destination object.
	Changed (see page 2817)	This is Changed, a member of class TPicture.
	DefineProperties (see page 2817)	Provides an interface for a method that reads and writes otherwise unpublished data.

	LoadFromStream (see page 2819)	Reads the picture from a Stream.
	Progress (see page 2819)	This is Progress, a member of class TPicture.
	SaveToStream (see page 2824)	Saves the picture to a stream.

Public Methods

	Name	Description
	Assign (see page 2815)	Copies one object to another by copying the contents of that object to the other.
	LoadFromClipboardFormat (see page 2818)	Reads the picture from the handle provided in the given Clipboard format.
	LoadFromFile (see page 2818)	Reads the file specified in Filename and loads the data into the TPicture (see page 2812) object.
	RegisterClipboardFormat (see page 2819)	Registers a new TGraphic (see page 2748) class for use in the LoadFromClipboardFormat (see page 2818) method.
	RegisterFileFormat (see page 2821)	Registers a new TGraphic (see page 2748) class for use in LoadFromFile (see page 2818).
	RegisterFileFormatRes (see page 2822)	Registers a new TGraphic (see page 2748) class for use in the LoadFromFile (see page 2818) method.
	SaveToClipboardFormat (see page 2822)	Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).
	SaveToFile (see page 2823)	Writes the picture to disk.
	SupportsClipboardFormat (see page 2824)	Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 2818) method.
	UnregisterGraphicClass (see page 2825)	This is UnregisterGraphicClass, a member of class TPicture.

8.1.19.4.1 TPicture.Assign Method

Copies one object to another by copying the contents of that object to the other.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

When Source is a object type that is valid for the Graphic (see page 2827) property, Assign makes that graphic the value of the Graphic (see page 2827) property.

The actions performed by Assign depend on the actual types of the TPicture (see page 2812) Graphic (see page 2827) property and Source. For example, if the Graphic (see page 2827) property and Source are bitmaps (TBitmap (see page 2628)), the bitmap contained in Source is copied into the Graphic (see page 2827) property. Similar conversions are valid, for example, for TIcon (see page 2768) or TMetafile (see page 2780).

If the Source parameter is not a valid object for the Graphic (see page 2827) property, Assign calls the inherited method so that the picture can be copied from any object with which it is compatible.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
```

See Also

Graphic (see page 2827)

Assign

8.1.19.4.2 TPicture.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

8.1.19.4.3 TPicture.Changed Method

Pascal

```
procedure Changed(Sender: TObject); dynamic;
```

C++

```
__fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TPicture.

8.1.19.4.4 TPicture.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties

DefineBinaryProperty

DefineProperty

8.1.19.4.5 TPicture.LoadFromClipboardFormat Method

Reads the picture from the handle provided in the given Clipboard format.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE);
```

C++

```
__fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use LoadFromClipboardFormat to read in a graphic from the Clipboard. If the format is not supported, an EInvalidGraphic (see page 2627) exception is raised.

See Also

EInvalidGraphic (see page 2627)

RegisterClipboardFormat (see page 2819)

TClipboard

8.1.19.4.6 TPicture.LoadFromFile Method

Reads the file specified in Filename and loads the data into the TPicture (see page 2812) object.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

Use LoadFromFile to read a picture from disk. The TGraphic (see page 2748) class created is determined by the file extension of the file. If the file extension is not recognized an EInvalidGraphic (see page 2627) exception is raised.

See Also

EInvalidGraphic (see page 2627)

RegisterFileFormat (see page 2821)

TGraphic (see page 2748)

8.1.19.4.7 TPicture.LoadFromStream Method

Reads the picture from a Stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to read a picture from a TStream object. The TGraphic (see page 2748) class created is determined by the type of data contained in the stream.

The data in the stream should be previously written by a call to the SaveToStream (see page 2824) method of another TPicture (see page 2812) or of a TGraphic (see page 2748) descendant.

See Also

EInvalidGraphic (see page 2627)

LoadFromMimeSource

LoadFromFile (see page 2818)

SaveToStream (see page 2824)

SaveToStream (see page 2758)

8.1.19.4.8 TPicture.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const UnicodeString Msg);
```

Description

This is Progress, a member of class TPicture.

8.1.19.4.9 TPicture.RegisterClipboardFormat Method

Registers a new TGraphic (see page 2748) class for use in the LoadFromClipboardFormat (see page 2818) method.

Pascal

```
class procedure RegisterClipboardFormat(AFormat: Word; AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterClipboardFormat(Word AFormat, TGraphicClass AGraphicClass);
```

Description

Use RegisterClipboardFormat register a new graphic format with TPicture (see page 2812) so that it can be used with the LoadFromClipboardFormat (see page 2818) method.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5, 5, Bitmap.get());
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  std::auto_ptr<Graphics::TBitmap> Bitmap(new Graphics::TBitmap());
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

LoadFromClipboardFormat (see page 2818)

TClipboard

UnRegisterGraphicClass (see page 2825)

8.1.19.4.10 TPicture.RegisterFileFormat Method

Registers a new TGraphic (see page 2748) class for use in LoadFromFile (see page 2818).

Pascal

```
class procedure RegisterFileFormat(const AExtension: string; const ADescription: string;
AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterFileFormat(const UnicodeString AExtension, const UnicodeString
ADescription, TGraphicClass AGraphicClass);
```

Description

Use RegisterFileFormat to register a graphic file format with TPicture (see page 2812) so that it can be used with a Open or Save dialog box.

The AExtension parameter specifies the three-character system file extension to associate with the graphic class (for example, "bmp" is associated with TBitmap (see page 2628)). The ADescription parameter specifies the description of the graphic to appear in the drop down list of the dialog box (for example, "Bitmaps" is the description of TBitmap (see page 2628)). The AGraphicClass parameter registers the new graphic class to associate with the file format.

See Also

LoadFromFile (see page 2818)

UnRegisterGraphicClass (see page 2825)

8.1.19.4.11 TPicture.RegisterFileFormatRes Method

Registers a new TGraphic (see page 2748) class for use in the LoadFromFile (see page 2818) method.

Pascal

```
class procedure RegisterFileFormatRes(const AExtension: String; ADescriptionResID: Integer;
AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterFileFormatRes(const UnicodeString AExtension, int ADescriptionResID,
TGraphicClass AGraphicClass);
```

Description

Use RegisterFileFormatRes by specifying a string resource.

The AExtension parameter specifies the three-character system file extension to associate with the graphic class (for example, "bmp" is associated with TBitmap (see page 2628)).

The ADescriptionResID parameter specifies the resource ID for a description of the graphic, which then appears in the drop down list of the dialog box (for example, "Bitmaps" is the description of TBitmap (see page 2628)).

The AGraphicClass parameter registers the new graphic class to associate with the file format.

See Also

LoadFromFile (see page 2818)

UnRegisterGraphicClass (see page 2825)

8.1.19.4.12 TPicture.SaveToClipboardFormat Method

Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette: HPALETTE);
```

C++

```
__fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the picture to a Clipboard format. The resulting values can then be copied to the Windows clipboard using the clipboard's SetAsHandle method.

The picture's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the picture can be saved, an application must have registered the format using the RegisterClipboardFormat (see page 2819) method.

See Also

Formats

RegisterClipboardFormat (see page 2819)

8.1.19.4.13 TPicture.SaveToFile Method

Writes the picture to disk.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

Use SaveToFile to save a TPicture (see page 2812) object to the file specified in Filename.

The saved picture can be loaded to another TPicture (see page 2812) or to an appropriate TGraphic (see page 2748) descendant by calling its LoadFromFile (see page 2818) method.

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    std::auto_ptr<TIcon> pi(new Graphics::TIcon);
    AnsiString as;
    pi->LoadFromFile(OpenDialog1->FileName);
    as = ChangeFileExt(OpenDialog1->FileName,".BMP");
    Image1->Width = pi->Width;
    Image1->Height = pi->Height;
    Image1->Canvas->Draw(0, 0, pi.get());
    Image1->Picture->SaveToFile(as);
    ShowMessage(OpenDialog1->FileName + " Saved to " + as);
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
```

```
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
```

See Also

LoadFromFile (see page 2818)

LoadFromFile (see page 2755)

8.1.19.4.14 TPicture.SaveToStream Method

Saves the picture to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save a picture to the TStream object specified by the Stream parameter.

The saved picture can be loaded to another TPicture (see page 2812) or to an appropriate TGraphic (see page 2748) descendant by calling its LoadFromStream (see page 2819) method.

See Also

LoadFromStream (see page 2819)

LoadFromStream (see page 2756)

8.1.19.4.15 TPicture.SupportsClipboardFormat Method

Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 2818) method.

Pascal

```
class function SupportsClipboardFormat(AFormat: Word): Boolean;
```

C++

```
__fastcall Boolean SupportsClipboardFormat(Word AFormat);
```

Description

If the LoadFromClipboardFormat (see page 2818) method supports the Clipboard format specified as the value of AFormat, SupportsClipboardFormat returns true. If the format is not supported, the method returns false.

See Also

LoadFromClipboardFormat (see page 2818)

8.1.19.4.16 TPicture.UnregisterGraphicClass Method

Pascal

```
class procedure UnregisterGraphicClass(AClass: TGraphicClass);
```

C++

```
__fastcall UnregisterGraphicClass(TGraphicClass AClass);
```

Description

This is UnregisterGraphicClass, a member of class TPicture.

8.1.19.5 TPicture Properties

The properties of the TPicture class are listed here.

Public Properties

	Name	Description
	Bitmap (see page 2825)	Specifies the contents of the picture object as a bitmap graphic (.BMP file format).
	Graphic (see page 2827)	Specifies the graphic that the picture contains.
R	Height (see page 2828)	Specifies the vertical size (in pixels) of the graphic.
	Icon (see page 2828)	Specifies the contents of the TPicture (see page 2812) object as an icon graphic (.ICO file format).
	Metafile (see page 2828)	Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).
	PictureAdapter (see page 2829)	Represents an OLE interface for the picture.
R	Width (see page 2829)	Specifies the horizontal size (in pixels) of the picture.

8.1.19.5.1 TPicture.Bitmap Property

Specifies the contents of the picture object as a bitmap graphic (.BMP file format).

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Use Bitmap to reference the picture object when it contains a bitmap. If Bitmap is referenced when the picture contains a Metafile (see page 2828) or Icon (see page 2828) graphic, the graphic won't be converted. Instead, the original contents of the picture are discarded and Bitmap returns a new, blank bitmap.

Note: When assigning the Bitmap property, TPicture (see page 2812) assigns the properties of a another TBitmap (see page 2628) object. It does not take ownership of the specified value.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
```

```
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

Graphic (see page 2827)

8.1.19.5.2 TPicture.Graphic Property

Specifies the graphic that the picture contains.

Pascal

```
property Graphic: TGraphic;
```

C++

```
__property TGraphic Graphic;
```

Description

Use Graphic to specify what graphic the TPicture (see page 2812) object contains. The graphic can be a bitmap, icon, metafile, or user-defined graphic class.

Note: When assigning the Graphic property, TPicture (see page 2812) assigns the properties of a another TGraphic (see page 2748) object. It does not take ownership of the specified object.

C++ Examples:

```
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

See Also

Bitmap (see page 2825)

Icon (see page 2828)

Metafile (see page 2828)

8.1.19.5.3 TPicture.Height Property

Specifies the vertical size (in pixels) of the graphic.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to find the height of the graphic image contained in the picture object. Height is the native, unstretched, height of the picture.

See Also

Width (see page 2829)

8.1.19.5.4 TPicture.Icon Property

Specifies the contents of the TPicture (see page 2812) object as an icon graphic (.ICO file format).

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

If Icon is referenced when the TPicture (see page 2812) contains a Bitmap (see page 2825) or Metafile (see page 2828) graphic, the graphic won't be converted. Instead, the original contents of the TPicture (see page 2812) are discarded and Icon returns a new, blank icon.

Note: When assigning the Icon property, TPicture (see page 2812) assigns the properties of a another Icon object. It does not take ownership of the specified value.

See Also

Graphic (see page 2827)

8.1.19.5.5 TPicture.Metafile Property

Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).

Pascal

```
property Metafile: TMetafile;
```

C++

```
__property TMetafile Metafile;
```

Description

If Metafile is referenced when the TPicture (see page 2812) contains a Bitmap (see page 2825) or Icon (see page 2828)

graphic, the graphic won't be converted. Instead, the original contents of the TPicture (see page 2812) are discarded and Metafile returns a new, blank metafile.

Note: When assigning the Metafile property, TPicture (see page 2812) assigns the properties of a another TMetafile (see page 2780) object. It does not take ownership of the specified value.

See Also

Graphic (see page 2827)

8.1.19.5.6 TPicture.PictureAdapter Property

Represents an OLE interface for the picture.

Pascal

```
property PictureAdapter: IChangeNotifier;
```

C++

```
__property IChangeNotifier PictureAdapter;
```

Description

PictureAdapter is for internal use only.

See Also

TPictureAdapter

8.1.19.5.7 TPicture.Width Property

Specifies the horizontal size (in pixels) of the picture.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use Width to find the width of a graphic image. Width contains the native, unstretched, width of the picture.

See Also

Height (see page 2828)

8.1.19.6 TPicture Events

The events of the TPicture class are listed here.

Public Events

	Name	Description
	OnChange (see page 2830)	Occurs when the specific graphics item encapsulated by the picture object changes.
	OnProgress (see page 2830)	Occurs periodically during slow operations that affect the graphic.

8.1.19.6.1 TPicture.OnChange Event

Occurs when the specific graphics item encapsulated by the picture object changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write an event handler to process an action when the graphic is modified.

8.1.19.6.2 TPicture.OnProgress Event

Occurs periodically during slow operations that affect the graphic.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

OnProgress is generated by the particular graphic that the picture object contains. Whether OnProgress occurs depends upon the type of graphic in the Graphic (see page 2827) property. Some graphics generate this event, others do not. Jpeg images, for example, generate an OnProgress event.

Write an OnProgress event handler to provide the user with feedback during slow operations such as loading large compressed images. TImage, for example, hooks the OnProgress event of TPicture (see page 2812).

Note: The PercentDone parameter on the event handler is only an approximation. With some image formats, the value of PercentDone may actually decrease from the value in previous events, as the graphic object discovers there is more work to do.

See Also

Graphic (see page 2827)

OnProgress (see page 1098)

8.1.20 TSharedImage Class

TSharedImage is the base class for classes that contain images used for graphics objects.

Class Hierarchy

Pascal

```
TSharedImage = class;
```

C++

```
class TSharedImage;
```

File

Graphics

Description

TSharedImage is used as a base class for classes that hold the internal representation of graphical images such as bitmaps and icons. TSharedImage is an abstract class. Its descendant classes are used internally by graphics objects, and are automatically created by their associated graphics objects.

See Also

TGraphic (see page 2748)

8.1.20.1 TSharedImage Members

The following tables list the members exposed by TSharedImage.

Protected Methods

	Name	Description
	FreeHandle (see page 2832)	This is FreeHandle, a member of class TSharedImage.
	Reference (see page 2832)	This is Reference, a member of class TSharedImage.
	Release (see page 2832)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
	RefCount (see page 2832)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
	Destroy (see page 2831)	This is Destroy, a member of class TSharedImage.

8.1.20.2 TSharedImage.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSharedImage();
```

Description

This is Destroy, a member of class TSharedImage.

8.1.20.3 TSharedImage Methods

The methods of the TSharedImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 2832)	This is FreeHandle, a member of class TSharedImage.
	Reference (see page 2832)	This is Reference, a member of class TSharedImage.
	Release (see page 2832)	This is Release, a member of class TSharedImage.

8.1.20.3.1 TSharedImage.FreeHandle Method

Pascal

```
procedure FreeHandle; virtual; abstract;
```

C++

```
virtual __fastcall FreeHandle() = 0;
```

Description

This is FreeHandle, a member of class TSharedImage.

8.1.20.3.2 TSharedImage.Reference Method

Pascal

```
procedure Reference;
```

C++

```
__fastcall Reference();
```

Description

This is Reference, a member of class TSharedImage.

8.1.20.3.3 TSharedImage.Release Method

Pascal

```
procedure Release;
```

C++

```
__fastcall Release();
```

Description

This is Release, a member of class TSharedImage.

8.1.20.4 TSharedImage Properties

The properties of the TSharedImage class are listed here.

Protected Properties

	Name	Description
	RefCount (see page 2832)	This is RefCount, a member of class TSharedImage.

8.1.20.4.1 TSharedImage.RefCount Property

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

This is RefCount, a member of class TSharedImage.

8.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IChangeNotifier (see page 2834)	IChangeNotifier is the interface used for responding to status changes.

8.2.1 IChangeNotifier Interface

IChangeNotifier is the interface used for responding to status changes.

Class Hierarchy

Graphics.IChangeNotifier

Pascal

```
IChangeNotifier = interface;
```

C++

```
__interface IChangeNotifier;
```

File

Graphics

Description

IChangeNotifier introduces one method, Changed (see page 2835) that is invoked when the status of the implementing class changes.

See Also

Changed (see page 2835)

8.2.1.1 IChangeNotifier Members

The following tables list the members exposed by IChangeNotifier.

Public Methods

	Name	Description
	Changed (see page 2835)	Called when a change occurs in the implementing class.

8.2.1.2 IChangeNotifier Methods

The methods of the IChangeNotifier class are listed here.

Public Methods

	Name	Description
	Changed (see page 2835)	Called when a change occurs in the implementing class.

8.2.1.2.1 IChangeNotifier.Changed Method

Called when a change occurs in the implementing class.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is principally used internally by TFont (see page 2730) class TFontAdapter property and TPicture (see page 2812) class TPictureAdapter property.

See Also

TFontAdapter

TPictureAdapter

8.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AllocPatternBitmap (see page 2836)	Creates a TBitmap (see page 2628) filled with a pattern.
	BytesPerScanline (see page 2837)	This is function Graphics.BytesPerScanline.
	CharsetToIdent (see page 2837)	Obtains the name of a character set.
	ColorToIdent (see page 2838)	Provides the symbolic name of a TColor (see page 2878) value.
	ColorToRGB (see page 2838)	Converts a TColor (see page 2878) value into an RGB representation of the color.
	ColorToString (see page 2840)	Returns a string that represents the name of a TColor (see page 2878) value.
	CopyPalette (see page 2841)	Creates a new palette object that matches an existing palette.
	CreateGrayMappedBmp (see page 2841)	Remaps the standard gray colors in a bitmap with the system grays.
	CreateGrayMappedRes (see page 2842)	Remaps the standard gray colors in a bitmap resource with the system grays.
	CreateMappedBmp (see page 2842)	Changes the color table in a bitmap.
	CreateMappedRes (see page 2843)	Changes the color table in a bitmap resource.
	FreeMemoryContexts (see page 2844)	This is function Graphics.FreeMemoryContexts.
	GetCharsetValues (see page 2844)	Passes the name of every predefined character set string to a callback function.
	GetColorValues (see page 2848)	Passes the name of every predefined color constant to a callback function.
	GetDefFontCharSet (see page 2848)	Returns the character set of the default system font.
	GetDIB (see page 2849)	Returns the info header and image for a device independent bitmap.
	GetDIBSizes (see page 2849)	Returns the size of the info header and of the image for a device independent bitmap.
	GraphicExtension (see page 2850)	Returns the default file-name extension of a graphics object.
	GraphicFileMask (see page 2852)	Returns a file mask that specifies the valid extensions for a graphic class.
	GraphicFilter (see page 2854)	Returns a file filter compatible with the Filter property of an Open or Save dialog.
	IdentToCharset (see page 2857)	Translates the name of a character set constant to the corresponding character set.
	IdentToColor (see page 2857)	Translates the name of a color constant to the corresponding TColor (see page 2878) value.
	PaletteChanged (see page 2858)	This is function Graphics.PaletteChanged.
	StringToColor (see page 2858)	Translates a string representation of a color to the corresponding TColor (see page 2878) value.
	TransparentStretchBlt (see page 2859)	This is function Graphics.TransparentStretchBlt.

8.3.1 Graphics.AllocPatternBitmap Function

Creates a TBitmap (see page 2628) filled with a pattern.

Pascal

```
function AllocPatternBitmap(BkColor: TColor; FgColor: TColor): TBitmap;
```

C++

```
TBitmap AllocPatternBitmap(TColor BkColor, TColor FgColor);
```

File

Graphics

Description

AllocPatternBitmap returns a reference to an 8 by 8 pixel TBitmap (see page 2628) that is filled with a pattern. Pixels alternate between **BkColor** and **FgColor** colors horizontally and vertically in a quilt pattern.

See Also

TBitmap (see page 2628)

TColor (see page 2878)

8.3.2 Graphics.BytesPerScanline Function

Pascal

```
function BytesPerScanline(PixelsPerScanline: Longint; BitsPerPixel: Longint; Alignment:
Longint): Longint;
```

C++

```
Longint BytesPerScanline(Longint PixelsPerScanline, Longint BitsPerPixel, Longint Alignment);
```

File

Graphics

Description

This is function Graphics.BytesPerScanline.

8.3.3 Graphics.CharsetToIdent Function

Obtains the name of a character set.

Pascal

```
function CharsetToIdent(Charset: Longint; var Ident: string): Boolean;
```

C++

```
Boolean CharsetToIdent(Longint Charset, UnicodeString Ident);
```

File

Graphics

Description

Call CharsetToIdent to obtain the symbolic name for the character set of a font. CharsetToIdent returns true if the Ident parameter is successfully given an identifying name for the character set.

Note: The return value of CharsetToIdent does not indicate whether the Charset parameter is valid for a given font. Many fonts support only a limited number of character sets. Some fonts may support less common character sets that can't be mapped to identifiers by CharsetToIdent.

See Also

TFont (see page 2730)

8.3.4 Graphics.ColorToIdent Function

Provides the symbolic name of a TColor (see page 2878) value.

Pascal

```
function ColorToIdent(Color: Longint; var Ident: string): Boolean;
```

C++

```
Boolean ColorToIdent(Longint Color, UnicodeString Ident);
```

File

Graphics

Description

Call ColorToIdent to obtain the name of the symbolic constant that represents a TColor (see page 2878) value, such as clBlack (see page 2899) or clWindow (see page 2965). The Ident parameter is set to the symbolic name. If there is no symbolic constant defined for the color, ColorToIdent returns false.

See Also

ColorToString (see page 2840)

TColor (see page 2878)

IdentToColor (see page 2857)

8.3.5 Graphics.ColorToRGB Function

Converts a TColor (see page 2878) value into an RGB representation of the color.

Pascal

```
function ColorToRGB(Color: TColor): Longint;
```

C++

```
Longint ColorToRGB(TColor Color);
```

File

Graphics

Description

Call ColorToRGB to obtain an RGB representation of a color for using with Windows API calls. ColorToRGB strips away the information that is stored in the highest order bits about which palette to use for colors that are not always available.

C++ Examples:

```
/*
The following code converts the color of the current form,
Form1, to a Windows RGB value:
*/
int NumPaletteEntries = 20;
TPaletteEntry FPaletteEntries[20];

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  GetPaletteEntries(GetStockObject(DEFAULT_PALETTE), 0, NumPaletteEntries,
    FPaletteEntries);
  Form1->Color = clMoneyGreen;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Result = 0;
  int I = 0;
  TColor RealColor = Graphics::ColorToRGB(Form1->Color);
  while (I < NumPaletteEntries)
  {
//  with FPaletteEntries[I] do
    TColor paletteColor =
    RGB(
      FPaletteEntries[I].peRed,
      FPaletteEntries[I].peGreen,
      FPaletteEntries[I].peBlue);
    if (RealColor == paletteColor)
    {
      Label1->Caption = IntToStr(I);
      RedEdit->Text = IntToStr(FPaletteEntries[I].peRed);
      GreenEdit->Text = IntToStr(FPaletteEntries[I].peGreen);
      BlueEdit->Text = IntToStr(FPaletteEntries[I].peBlue);
      break;
    }
    I++;
  };
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Color =
    (StrToInt(RedEdit->Text) << 16) +
    (StrToInt(GreenEdit->Text) << 8) +
    (StrToInt(BlueEdit->Text));
}
```

Delphi Examples:

```
{
The following code converts the color of the current form,
Form1, to a Windows RGB value:
}
const
  NumPaletteEntries = 20;
var
  FPaletteEntries: array[0..NumPaletteEntries - 1] of TPaletteEntry;

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Result: Integer;
  RealColor, paletteColor: TColor;
begin
  Result := 0;
  I := 0;
  RealColor := Graphics.ColorToRGB(Form1.Color);
  while I < NumPaletteEntries do
  begin
    with FPaletteEntries[I] do
    begin
      paletteColor:= TColor(RGB(peRed, peGreen, peBlue));
      if RealColor = paletteColor then
      begin
        Label1.Caption := IntToStr(I);
```

```
          RedEdit.Text:= IntToStr(peRed);
          GreenEdit.Text:= IntToStr(peGreen);
          BlueEdit.Text:= IntToStr(peBlue);
          Exit;
        end;
      end;
      Inc(I)
    end;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form1.Color:=
      StrToInt(RedEdit.Text) shl 16 +
      StrToInt(GreenEdit.Text) shl 8 +
      StrToInt(BlueEdit.Text);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    GetPaletteEntries(GetStockObject(DEFAULT_PALETTE), 0, NumPaletteEntries,
      FPaletteEntries);
    Form1.Color:= clMoneyGreen;
  end;
```

See Also

ColorToIdent (see page 2838)

8.3.6 Graphics.ColorToString Function

Returns a string that represents the name of a TColor (see page 2878) value.

Pascal

```
function ColorToString(Color: TColor): string;
```

C++

```
UnicodeString ColorToString(TColor Color);
```

File

Graphics

Description

Call ColorToString to obtain a string that represents a TColor (see page 2878) value. If there is a symbolic constant defined for the color (such as clBlack (see page 2899) or clHighlightText (see page 2907)), ColorToString returns the name of the constant. Otherwise, ColorToString returns the hex value of the color, formatted as a string such as "$02FF8800" (Delphi) or "0x02FF8800" (C++).

See Also

ColorToIdent (see page 2838)

StringToColor (see page 2858)

8.3.7 Graphics.CopyPalette Function

Creates a new palette object that matches an existing palette.

Pascal

```
function CopyPalette(Palette: HPALETTE): HPALETTE;
```

C++

```
HPALETTE CopyPalette(HPALETTE Palette);
```

File

Graphics

Description

Call CopyPalette to create a new Windows palette object that exactly matches an existing palette. If the Palette parameter does not refer to a valid Windows palette, or if a problem occurs while copying the palette, CopyPalette returns 0.

8.3.8 Graphics.CreateGrayMappedBmp Function

Remaps the standard gray colors in a bitmap with the system grays.

Pascal

```
function CreateGrayMappedBmp(Handle: HBITMAP): HBITMAP;
```

C++

```
HBITMAP CreateGrayMappedBmp(HBITMAP Handle);
```

File

Graphics

Description

Call CreateGrayMappedBmp to create a bitmap that uses the system gray colors (clBtnHightlight, clBtnFace (see page 2900), clBtnShadow (see page 2901) and clBtnText (see page 2901)) from a bitmap that uses standard gray scale colors (clWhite (see page 2965), clSilver (see page 2916), clGray (see page 2905), and clBlack (see page 2899)). This allows an application to dynamically create images that match the user's control panel settings, given a base image.

CreateGrayMappedBmp creates a new bitmap. It does not alter the original bitmap, which has the handle specified by the Handle parameter.

Note: CreateGrayMappedBmp does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns the original bitmap.

See Also

TBitmap (see page 2628)

CreateMappedBmp (see page 2842)

CreateGrayMappedRes (see page 2842)

Graphics

8.3.9 Graphics.CreateGrayMappedRes Function

Remaps the standard gray colors in a bitmap resource with the system grays.

Pascal

```
function CreateGrayMappedRes(Instance: THandle; ResName: PChar): HBITMAP;
```

C++

```
HBITMAP CreateGrayMappedRes(THandle Instance, const char * ResName);
```

File

Graphics

Description

Call CreateGrayMappedRes to create a bitmap that uses the system gray colors (clBtnHightlight, clBtnFace (see page 2900), clBtnShadow (see page 2901) and clBtnText (see page 2901)) from a resourced bitmap that uses standard gray scale colors (clWhite (see page 2965), clSilver (see page 2916), clGray (see page 2905), and clBlack (see page 2899)). This allows an application to dynamically create images that match the user's control panel settings, by creating a base image as a bitmap resource.

Use the Instance and ResName parameters to identify the bitmap resource whose color table should be changed

Note: CreateMappedRes (see page 2843) does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns an instance of the original bitmap without color modifications.

See Also

TBitmap (see page 2628)

CreateGrayMappedBmp (see page 2841)

FindResourceHInstance

Hinstance

CreateMappedRes (see page 2843)

Graphics

8.3.10 Graphics.CreateMappedBmp Function

Changes the color table in a bitmap.

Pascal

```
function CreateMappedBmp(Handle: HBITMAP; const OldColors: array of TColor; const NewColors:
array of TColor): HBITMAP;
```

C++

```
HBITMAP CreateMappedBmp(HBITMAP Handle, const array of TColor OldColors, const array of TColor
NewColors);
```

File

Graphics

Description

Call CreateMappedBmp to make selective changes to the color map of a bitmap. This function replaces the colors specified by the OldColors parameter with the colors specified by the NewColors parameter and returns the handle for a new device dependent bitmap (DDB) that uses the new color table. Handle is the handle to the bitmap whose color table should be changed. In C++, specify the size of the color arrays in OldColors_Size and NewColors_Size. This value should be the index of the last entry in the corresponding array. The OldColors and NewColors arrays should contain the same number of elements.

CreateMappedBmp creates a new bitmap. It does not alter the original bitmap, which has the handle specified by the Handle parameter.

Note: CreateMappedBmp does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns the original bitmap.

See Also

TBitmap (see page 2628)

CreateMappedRes (see page 2843)

CreateGrayMappedBmp (see page 2841)

8.3.11 Graphics.CreateMappedRes Function

Changes the color table in a bitmap resource.

Pascal

```
function CreateMappedRes(Instance: THandle; ResName: PChar; const OldColors: array of TColor;
const NewColors: array of TColor): HBITMAP;
```

C++

```
HBITMAP CreateMappedRes(THandle Instance, const char * ResName, const array of TColor
OldColors, const array of TColor NewColors);
```

File

Graphics

Description

Call CreateMappedRes to make selective changes to the color map of a resourced bitmap. This function replaces the colors specified by the OldColors parameter with the colors specified by the NewColors parameter and returns the handle for a new device dependent bitmap (DDB) that uses the new color table. Use the Instance and ResName parameters to identify the bitmap resource whose color table should be changed. In C++, indicate the number of elements in the color arrays by the OldColors_Size and NewColors_Size parameters. These values should be the index of the last element in the corresponding color array, not the number of entries. The OldColors and NewColors arrays should contain the same number of elements.

Note: CreateMappedRes does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns an instance of the original bitmap without color modifications.

See Also

TBitmap (see page 2628)

CreateMappedBmp (see page 2842)

FindResourceHInstance

Hinstance

CreateGrayMappedRes (see page 2842)

8.3.12 Graphics.FreeMemoryContexts Function

Pascal

```
procedure FreeMemoryContexts;
```

C++

```
FreeMemoryContexts();
```

File

Graphics

Description

This is function Graphics.FreeMemoryContexts.

8.3.13 Graphics.GetCharsetValues Function

Passes the name of every predefined character set string to a callback function.

Pascal

```
procedure GetCharsetValues(Proc: TGetStrProc);
```

C++

```
GetCharsetValues(TGetStrProc Proc);
```

File

Graphics

Description

Call GetCharsetValues to execute a callback for every character set name that has a corresponding constant defined in the application. The Proc parameter is the callback function that is called for every character set constant name.

Note: To convert the strings that are passed to the callback into the TFontCharset (see page 2881) type, use the IdentToCharset (see page 2857) function.

C++ Examples:

```
/*
The following uses the GetCharsetValues procedure to fill a
string list with the names of all the available character
```

```
set constants. This string list can then be used to allow
the user to select a character set.  Here is the callback
that adds the character sets to the string list:
*/
TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(const UnicodeString S)
{
  CharSetList->Add(S);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  CharSetList = new TStringList();
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ListBox1->Items = CharSetList;
  delete CharSetList;
}
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
```

```
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}

TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(String S)
{
  CharSetList->Add(S);
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TStringList> _CharSetListCleaner(CharSetList = new TStringList);
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following uses the GetCharsetValues procedure to fill a
string list with the names of all the available character
set constants. This string list can then be used to allow
the user to select a character set.  Here is the callback
that adds the character sets to the string list:
}
var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;
{
Here is the code that creates the string list, fills it, and sorts the character set names.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  CharSetList.Sort;
  ListBox1.Items := CharSetList;
  CharSetList.Destroy;
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
```

```
  default auto completion.
  }
  procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
  var
    Found: boolean;
    i, ct, SelSt: Integer;
    TmpStr: string;
  begin
    { first, process the keystroke to obtain the current string }
    { This code requires all items in list to be uppercase}
    if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
    ct := (Sender as TComboBox).Items.Count;
    with (Sender as TComboBox) do
    begin
      SelSt := SelStart;
      if (Key = Chr(vk_Back)) and (SelLength <> 0) then
       TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
      else if Key = Chr(vk_Back) then {SelLength = 0}
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

TFont (see page 2730)

IdentToCharset (see page 2857)

TFontCharset (see page 2881)

8.3.14 Graphics.GetColorValues Function

Passes the name of every predefined color constant to a callback function.

Pascal

```
procedure GetColorValues(Proc: TGetStrProc);
```

C++

```
GetColorValues(TGetStrProc Proc);
```

File

Graphics

Description

Call GetColorValues to execute a callback for every TColor (see page 2878) constant defined in the application. The Proc parameter is the callback function that is called for every color constant name.

Note: To convert the strings that are passed to the callback into the TColor (see page 2878) type, use the IdentToColor (see page 2857) function.

See Also

TColor (see page 2878)

IdentToColor (see page 2857)

8.3.15 Graphics.GetDefFontCharSet Function

Returns the character set of the default system font.

Pascal

```
function GetDefFontCharSet: TFontCharSet;
```

C++

```
TFontCharSet GetDefFontCharSet();
```

File

Graphics

Description

Call GetDefFontCharSet to determine the character set that is used when a font is not explicitly assigned to a control. This character set can be used to initialize a font that is specified by font name only, to best match the characters for the locale of the machine on which the application is running.

See Also

TFont (see page 2730)

8.3.16 Graphics.GetDIB Function

Returns the info header and image for a device independent bitmap.

Pascal

```
function GetDIB(Bitmap: HBITMAP; Palette: HPALETTE; var BitmapInfo; var Bits): Boolean;
```

C++

```
Boolean GetDIB(HBITMAP Bitmap, HPALETTE Palette,  BitmapInfo,  Bits);
```

File

Graphics

Description

Call GetDIB to convert a device dependent bitmap into a device independent bitmap (DIB). Pass the handle to a device dependent bitmap and a handle to its palette as the Bitmap and Palette parameters. GetDIB fills in the BitmapInfo and Bits parameters with the info header and image of the corresponding DIB. GetDIB returns true if the bitmap was successfully converted into a DIB, false otherwise.

Before calling GetDIB, the memory for the info header and image must be allocated. Call GetDIBSizes (see page 2849) to determine the amount of memory that must be allocated for the info header and image.

See Also

GetDIBSizes (see page 2849)

8.3.17 Graphics.GetDIBSizes Function

Returns the size of the info header and of the image for a device independent bitmap.

Pascal

```
procedure GetDIBSizes(Bitmap: HBITMAP; var InfoHeaderSize: DWORD; var ImageSize: DWORD);
```

C++

```
GetDIBSizes(HBITMAP Bitmap, DWORD InfoHeaderSize, DWORD ImageSize);
```

File

Graphics

Description

Call GetDIBSizes to determine the amount of memory needed to convert a device dependent bitmap into a device independent bitmap (DIB). Pass the handle to a device dependent bitmap as the Bitmap parameter. GetDIBSizes returns the sizes of the info header and image of the corresponding DIB as the InfoHeaderSize parameter and the ImageSize parameter, respectively.

Call GetDIBSizes to determine the amount of memory that must be allocated before calling GetDIB (see page 2849) to convert a device dependent bitmap into a device independent bitmap.

See Also

GetDIB (see page 2849)

8.3.18 Graphics.GraphicExtension Function

Returns the default file-name extension of a graphics object.

Pascal

```
function GraphicExtension(GraphicClass: TGraphicClass): string;
```

C++

```
UnicodeString GraphicExtension(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicExtension to obtain a default extension value for the DefaultExt property of an Open, Open Picture, Save Picture or Save dialog. GraphicExtension returns the default extension for the type of graphics object specified by the GraphicClass parameter. In Delphi, the TGraphicClass (see page 2887) type is the class type for descendants TGraphic (see page 2748).

These are the file extensions returned for each built-in graphics class:

Graphic class	File extension returned
TBitmap (see page 2628)	bmp
TIcon (see page 2768)	ico
TMetafile (see page 2780)	emf

If custom graphics classes register a file extension, GraphicExtension will return that extension when passed the custom graphics class as an argument.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
#include <memory>       //for STL auto_ptr class

TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TClassFinder> myClassFinder(new TClassFinder(MetaClass, False));
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, (fmOpenRead |
fmShareDenyWrite)));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, (fmCreate |
fmShareDenyRead)));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
  { Usually, this goes in the initialization section where it is only executed once }
  RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
  try
    myClassFinder := TClassFinder.Create(TIcon, False);
    mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
```

```
    Edit1.Text := GraphicExtension(mygraphclass);
  except
    on EAccessViolation do
      MessageDlg('Invalid class name.', mtError, [mbOK], 0)
  end;
end;
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

See Also

GraphicFileMask (see page 2852)

GraphicFilter (see page 2854)

TPicture (see page 2812)

TOpenDialog

TOpenPictureDialog

TSavePictureDialog

8.3.19 Graphics.GraphicFileMask Function

Returns a file mask that specifies the valid extensions for a graphic class.

Pascal

```
function GraphicFileMask(GraphicClass: TGraphicClass): string;
```

C++

```
UnicodeString GraphicFileMask(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicFileMask to obtain a mask to determine the file extensions associated with a graphic class. The GraphicClass parameter can specify one of these values: TBitmap (see page 2628), TGraphic (see page 2748), TIcon (see page 2768), TMetafile (see page 2780), or a user-defined descendant of TGraphic (see page 2748). These are the strings that are returned for each built-in graphics class:

Graphic class	Filter string returned
TBitmap (see page 2628)	*.bmp
TIcon (see page 2768)	*.ico
TMetafile (see page 2780)	*.emf;*.wmf
TGraphic (see page 2748)	*.bmp;*.ico;*.emf;*.wmf
TGraphic (see page 2748)	;*.bmp;*.png;*.xpm*.ico

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

See Also

GraphicExtension (see page 2850)

GraphicFilter (see page 2854)

TFilterComboBox

TPicture (see page 2812)

8.3.20 Graphics.GraphicFilter Function

Returns a file filter compatible with the Filter property of an Open or Save dialog.

Pascal

```
function GraphicFilter(GraphicClass: TGraphicClass): string;
```

C++

```
UnicodeString GraphicFilter(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicFilter to obtain a value for the Filter property of an Open, Open Picture, Save Picture or Save dialog box. The GraphicClass parameter can specify one of these values: TBitmap (see page 2628), TGraphic (see page 2748), TIcon (see page 2768), TMetafile (see page 2780), or a user-defined descendant of TGraphic (see page 2748).

These are the strings that are returned for each built-in graphics class when the AllFilter parameter is true. (When AllFilter is false, the section labelled "All" is omitted):

Graphic class	Filter string returned
TBitmap (see page 2628)	Bitmaps (*.bmp;*.png;*.xpm)|*.bmp;*.png;*.xpm
TIcon (see page 2768)	Icons (*.ico)|*.ico
TGraphic (see page 2748)	All (*.bmp;*.png;*.xpm*.ico)|*.bmp;*.png;*.xpm*.ico|Bitmaps (*.bmp;*.png;*.xpm)|*.bmp;*.png;*.xpm|Icons (*.ico)|*.ico
TBitmap (see page 2628)	Bitmaps (*.bmp)|*.bmp
TIcon (see page 2768)	Icons (*.ico)|*.ico
TMetafile (see page 2780)	All (*.emf;*.wmf)|*.emf;*.wmf|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf
TGraphic (see page 2748)	All (*.bmp;*.ico;*.emf;*.wmf)|*.bmp;*.ico;*.emf; *.wmf|Bitmaps (*.bmp)|*.bmp|Icons (*.ico)|*.ico|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf

Graphic class	Filter string returned
TBitmap (see page 2628)	Bitmaps (*.bmp)|*.bmp
TIcon (see page 2768)	Icons (*.ico)|*.ico
TMetafile (see page 2780)	All (*.emf;*.wmf)|*.emf;*.wmf|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf
TGraphic (see page 2748)	All (*.bmp;*.ico;*.emf;*.wmf)|*.bmp;*.ico;*.emf; *.wmf|Bitmaps (*.bmp)|*.bmp|Icons (*.ico)|*.ico|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
```

```
  {
    NewFileName = SavePictureDialog1->FileName;
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, (fmOpenRead |
fmShareDenyWrite)));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, (fmCreate |
fmShareDenyRead)));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

See Also

GraphicExtension (see page 2850)

GraphicFileMask (see page 2852)

TOpenDialog

TOpenPictureDialog

TPicture (see page 2812)

TSavePictureDialog

8.3.21 Graphics.IdentToCharset Function

Translates the name of a character set constant to the corresponding character set.

Pascal

```
function IdentToCharset(const Ident: string; var Charset: Longint): Boolean;
```

C++

```
Boolean IdentToCharset(const UnicodeString Ident, Longint Charset);
```

File

Graphics

Description

Call IdentToCharset to reverse the translation performed by CharsetToIdent (see page 2837). This method is useful for converting the strings generated by GetCharsetValues (see page 2844) into useable character sets. IdentToCharset returns true if the Ident parameter is successfully converted into a character set that is returned by the Charset parameter. IdentToCharset returns false if the Ident parameter is not the name of one of the TFontCharset (see page 2881) constants.

See Also

TFont (see page 2730)

GetCharsetValues (see page 2844)

CharSetToIdent

TFontCharset (see page 2881)

8.3.22 Graphics.IdentToColor Function

Translates the name of a color constant to the corresponding TColor (see page 2878) value.

Pascal

```
function IdentToColor(const Ident: string; var Color: Longint): Boolean;
```

C++

```
Boolean IdentToColor(const UnicodeString Ident, Longint Color);
```

File

Graphics

Description

Call IdentToColor to reverse the translation performed by ColorToIdent (see page 2838). This method is useful for converting the strings generated by GetColorValues (see page 2848) into useable TColor (see page 2878) values. IdentToColor returns true if the Ident parameter is successfully converted into a color that is returned by the Color parameter. IdentToColor returns false if the Ident parameter is not the name of one of the TColor (see page 2878) constants.

Note: If the string identifier for a color may represent either a hex value or a color constant, use the StringToColor (see page 2858) function instead.

See Also

TColor (see page 2878)

GetColorValues (see page 2848)

ColorToIdent (see page 2838)

StringToColor (see page 2858)

8.3.23 Graphics.PaletteChanged Function

Pascal

```
procedure PaletteChanged;
```

C++

```
PaletteChanged();
```

File

Graphics

Description

This is function Graphics.PaletteChanged.

8.3.24 Graphics.StringToColor Function

Translates a string representation of a color to the corresponding TColor (see page 2878) value.

Pascal

```
function StringToColor(const S: string): TColor;
```

C++

```
TColor StringToColor(const UnicodeString S);
```

File

Graphics

Description

Call StringToColor to reverse the translation performed by ColorToString (see page 2840). This method is useful for converting the strings entered by the user into useable TColor (see page 2878) values.

The S parameter can be either the name of a built-in color constant, such as "clBtnFace (see page 2900)" (Windows) or "clBtnText (see page 2901)" (cross-platform), or the string representation of a valid TColor (see page 2878) value such as "$02FF8800" (Delphi) or "0x02FF8800" (C++). StringToColor returns the TColor (see page 2878) value that corresponds to the S parameter.

Note: If the string identifier can only be the name of a color constant, use IdentToColor (see page 2857) instead.

See Also

ColorToString (see page 2840)

IdentToColor (see page 2857)

8.3.25 Graphics.TransparentStretchBlt Function

Pascal

```
function TransparentStretchBlt(DstDC: HDC; DstX: Integer; DstY: Integer; DstW: Integer; DstH:
Integer; SrcDC: HDC; SrcX: Integer; SrcY: Integer; SrcW: Integer; SrcH: Integer; MaskDC: HDC;
MaskX: Integer; MaskY: Integer): Boolean;
```

C++

```
Boolean TransparentStretchBlt(HDC DstDC, int DstX, int DstY, int DstW, int DstH, HDC SrcDC,
int SrcX, int SrcY, int SrcW, int SrcH, HDC MaskDC, int MaskX, int MaskY);
```

File

Graphics

Description

This is function Graphics.TransparentStretchBlt.

8.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAlphaFormat (see page 2860)	This is record Graphics.TAlphaFormat.
	TBitmapHandleType (see page 2861)	TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.
	TBrushStyle (see page 2862)	TBrushStyle specifies the pattern on a brush.
	TCanvasOrientation (see page 2863)	TCanvasOrientation specifies the pixel coordinate system that a canvas uses.
	TCanvasStates (see page 2863)	This is record Graphics.TCanvasStates.
	TFillMode (see page 2864)	This is record Graphics.TFillMode.
	TFillStyle (see page 2864)	TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.
	TFontPitch (see page 2866)	TFontPitch indicates whether the characters in a font have the same width.
	TFontStyle (see page 2866)	TFontStyle and TFontStyles (see page 2883) specify style characteristics of a font.
	TPenMode (see page 2868)	TPenMode indicates how the color of a pen interacts with the color of a canvas.
	TPenStyle (see page 2870)	TPenStyle indicates the type of line a pen draws.
	TPixelFormat (see page 2871)	TPixelFormat specifies the format of a bitmap.
	TProgressStage (see page 2872)	TProgressStage indicates the stage of a lengthy operation.
	TTextFormats (see page 2873)	This is record Graphics.TTextFormats.
	TTransparentMode (see page 2874)	TTransparentMode indicates how the transparent color of a bitmap is determined.

Records

	Name	Description
	TBrushData (see page 2861)	This is record Graphics.TBrushData.
	TCursorOrIcon (see page 2864)	This is record Graphics.TCursorOrIcon.
	TFontData (see page 2865)	This is record Graphics.TFontData.
	TIconRec (see page 2867)	This is record Graphics.TIconRec.
	TPenData (see page 2868)	This is record Graphics.TPenData.
	TResData (see page 2872)	This is record Graphics.TResData.
	TResource (see page 2873)	This is record Graphics.TResource.

8.4.1 Graphics.TAlphaFormat Enumeration

Pascal

```
TAlphaFormat = (
  afIgnored,
  afDefined,
  afPremultiplied
);
```

C++

```
enum TAlphaFormat {
  afIgnored,
  afDefined,
  afPremultiplied
};
```

File

Graphics

Description

This is record Graphics.TAlphaFormat.

8.4.2 Graphics.TBitmapHandleType Enumeration

TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.

Pascal

```
TBitmapHandleType = (
  bmDIB,
  bmDDB
);
```

C++

```
enum TBitmapHandleType {
  bmDIB,
  bmDDB
};
```

File

Graphics

Description

TBitmapHandleType can be any of the following values:

Value	Description
bmDIB	device-independent bitmap (DIB)
bmDDB	device-dependent bitmap (DDB)

8.4.3 Graphics.TBrushData Record

Pascal

```
TBrushData = record
  Handle: HBrush;
  Color: TColor;
  Bitmap: TBitmap;
  Style: TBrushStyle;
  OwnsBitmap: Boolean;
end;
```

C++

```
struct TBrushData {
  HBrush Handle;
  TColor Color;
  TBitmap Bitmap;
  TBrushStyle Style;
  Boolean OwnsBitmap;
```

```
};
```

File

Graphics

Description

This is record Graphics.TBrushData.

8.4.4 Graphics.TBrushStyle Enumeration

TBrushStyle specifies the pattern on a brush.

Pascal

```
TBrushStyle = (
  bsSolid,
  bsClear,
  bsHorizontal,
  bsVertical,
  bsFDiagonal,
  bsBDiagonal,
  bsCross,
  bsDiagCross
);
```

C++

```
enum TBrushStyle {
  bsSolid,
  bsClear,
  bsHorizontal,
  bsVertical,
  bsFDiagonal,
  bsBDiagonal,
  bsCross,
  bsDiagCross
};
```

File

Graphics

Description

The possible values of TBrushStyle are

ValuePatternValuePattern

Value	Pattern
bsSolid	bsCross
bsClear	bsDiagCross
bsBDiagonal	bsHorizontal
bsFDiagonal	bsVertical

See Also

TBrush (see page 2664)

8.4.5 Graphics.TCanvasOrientation Enumeration

TCanvasOrientation specifies the pixel coordinate system that a canvas uses.

Pascal

```
TCanvasOrientation = (
  coLeftToRight,
  coRightToLeft
);
```

C++

```
enum TCanvasOrientation {
  coLeftToRight,
  coRightToLeft
};
```

File

Graphics

Description

TCanvasOrientation determines where the (0,0) point falls on a canvas, and whether pixel coordinates are interpreted in a right-to-left or left-to-right manner.

The following table describes the possible values:

Value	Meaning
coLeftToRight	The origin (0,0) appears in the top-left corner. Pixel coordinates get larger as you move left or down.
coRightToLeft	The origin (0,0) appears in the top-right corner. Pixel coordinates get larger as you move right or down.

8.4.6 Graphics.TCanvasStates Enumeration

Pascal

```
TCanvasStates = (
  csHandleValid,
  csFontValid,
  csPenValid,
  csBrushValid
);
```

C++

```
enum TCanvasStates {
  csHandleValid,
  csFontValid,
  csPenValid,
  csBrushValid
};
```

File

Graphics

Description

This is record Graphics.TCanvasStates.

8.4.7 Graphics.TCursorOrIcon Record

Pascal

```
TCursorOrIcon = packed record
  Reserved: Word;
  wType: Word;
  Count: Word;
end;
```

C++

```
struct TCursorOrIcon {
  Word Reserved;
  Word wType;
  Word Count;
};
```

File

Graphics

Description

This is record Graphics.TCursorOrIcon.

8.4.8 Graphics.TFillMode Enumeration

Pascal

```
TFillMode = (
  fmAlternate,
  fmWinding
);
```

C++

```
enum TFillMode {
  fmAlternate,
  fmWinding
};
```

File

Graphics

Description

This is record Graphics.TFillMode.

8.4.9 Graphics.TFillStyle Enumeration

TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.

Pascal

```
TFillStyle = (
  fsSurface,
  fsBorder
);
```

C++

```
enum TFillStyle {
  fsSurface,
  fsBorder
};
```

File

Graphics

Description

TFillStyle values determine what type of color changes define the boundaries of a (possibly) non-rectangular region of a canvas, as indicated in the following table.

Value	Meaning
fsSurface	The region is defined as all area that has a specified color. When another color is encountered, it is taken as the boundary of the region.
fsBorder	The region is defined as all area that does not have a specified color. When the specified color is encountered, it is taken as the boundary of the region.

8.4.10 Graphics.TFontData Record

Pascal

```
TFontData = record
  Handle: HFont;
  Height: Integer;
  Orientation: Integer;
  Pitch: TFontPitch;
  Style: TFontStylesBase;
  Charset: TFontCharset;
  Name: TFontDataName;
end;
```

C++

```
struct TFontData {
  HFont Handle;
  int Height;
  int Orientation;
  TFontPitch Pitch;
  TFontStylesBase Style;
  TFontCharset Charset;
  TFontDataName Name;
};
```

File

Graphics

Description

This is record Graphics.TFontData.

8.4.11 Graphics.TFontPitch Enumeration

TFontPitch indicates whether the characters in a font have the same width.

Pascal

```
TFontPitch = (
  fpDefault,
  fpVariable,
  fpFixed
);
```

C++

```
enum TFontPitch {
  fpDefault,
  fpVariable,
  fpFixed
};
```

File

Graphics

Description

The following table lists the possible values of the TFontPitch type:

Value	Meaning
fpDefault	The font pitch is set to the default value, which depends on the font specified in the font object's Name property.
fpFixed	The font pitch is set to fixed. All characters in the font have the same width.
fpVariable	The font pitch is set to variable. The characters in the font have different widths.

8.4.12 Graphics.TFontStyle Enumeration

TFontStyle and TFontStyles (see page 2883) specify style characteristics of a font.

Pascal

```
TFontStyle = (
  fsBold,
  fsItalic,
  fsUnderline,
  fsStrikeOut
);
```

C++

```
enum TFontStyle {
  fsBold,
  fsItalic,
  fsUnderline,
  fsStrikeOut
};
```

File

Graphics

Description

TFontStyles (see page 2883) is a set of TFontStyle values. The following table lists the possible values of the TFontStyle type:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

Delphi Examples:

```
{
This example demonstrates how to set and clear font styles.
This example requires a button and a label that has text
with some styles set.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   Label1.Font.Style := [fsBold, fsUnderline];
end;
{
This code demonstrates how to clear any Styles that were
previously set.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
   Label1.Font.Style := [];
end;
```

8.4.13 Graphics.TIconRec Record

Pascal

```
TIconRec = packed record
  Width: Byte;
  Height: Byte;
  Colors: Word;
  Reserved1: Word;
  Reserved2: Word;
  DIBSize: Longint;
  DIBOffset: Longint;
end;
```

C++

```
struct TIconRec {
  Byte Width;
  Byte Height;
  Word Colors;
  Word Reserved1;
  Word Reserved2;
  Longint DIBSize;
  Longint DIBOffset;
};
```

File

Graphics

Description

This is record Graphics.TIconRec.

8.4.14 Graphics.TPenData Record

Pascal

```
TPenData = record
  Handle: HPen;
  Color: TColor;
  Width: Integer;
  Style: TPenStyle;
end;
```

C++

```
struct TPenData {
  HPen Handle;
  TColor Color;
  int Width;
  TPenStyle Style;
};
```

File

Graphics

Description

This is record Graphics.TPenData.

8.4.15 Graphics.TPenMode Enumeration

TPenMode indicates how the color of a pen interacts with the color of a canvas.

Pascal

```
TPenMode = (
  pmBlack,
  pmWhite,
  pmNop,
  pmNot,
  pmCopy,
  pmNotCopy,
  pmMergePenNot,
  pmMaskPenNot,
  pmMergeNotPen,
  pmMaskNotPen,
  pmMerge,
  pmNotMerge,
  pmMask,
  pmNotMask,
  pmXor,
  pmNotXor
);
```

C++

```
enum TPenMode {
  pmBlack,
  pmWhite,
  pmNop,
  pmNot,
  pmCopy,
  pmNotCopy,
  pmMergePenNot,
  pmMaskPenNot,
  pmMergeNotPen,
  pmMaskNotPen,
  pmMerge,
  pmNotMerge,
  pmMask,
  pmNotMask,
  pmXor,
  pmNotXor
};
```

File

Graphics

Description

TPenMode indicates how a pen color interacts with the color of the canvas it is writing on. The following table lists the possible values:

Mode	Pixel color
pmBlack	Always black
pmWhite	Always white
pmNop	Unchanged
pmNot	Inverse of canvas background color
pmCopy	Pen color specified in Color property
pmNotCopy	Inverse of pen color
pmMergePenNot	Combination of pen color and inverse of canvas background
pmMaskPenNot	Combination of colors common to both pen and inverse of canvas background.
pmMergeNotPen	Combination of canvas background color and inverse of pen color
pmMaskNotPen	Combination of colors common to both canvas background and inverse of pen
pmMerge	Combination of pen color and canvas background color
pmNotMerge	Inverse of pmMerge: combination of pen color and canvas background color
pmMask	Combination of colors common to both pen and canvas background
pmNotMask	Inverse of pmMask: combination of colors common to both pen and canvas background
pmXor	Combination of colors in either pen or canvas background, but not both
pmNotXor	Inverse of pmXor: combination of colors in either pen or canvas background, but not both

8.4.16 Graphics.TPenStyle Enumeration

TPenStyle indicates the type of line a pen draws.

Pascal

```
TPenStyle = (
  psSolid,
  psDash,
  psDot,
  psDashDot,
  psDashDotDot,
  psClear,
  psInsideFrame,
  psUserStyle,
  psAlternate
);
```

C++

```
enum TPenStyle {
  psSolid,
  psDash,
  psDot,
  psDashDot,
  psDashDotDot,
  psClear,
  psInsideFrame,
  psUserStyle,
  psAlternate
};
```

File

Graphics

Description

TPenStyle can have one of the following values:

Value	Meaning
psSolid	A solid line.
psDash	A line made up of a series of dashes.
psDot	A line made up of a series of dots.
psDashDot	A line made up of alternating dashes and dots.
psDashDotDot	A line made up of a series of dash-dot-dot combinations.
psClear	No line is drawn (used to omit the line around shapes that draw an outline using the current pen).
psInsideFrame	A solid line, but one that may use a dithered color if Width is greater than 1.
psAlternate	The pen sets every other pixel. (This style is applicable only for cosmetic pens.) This style is only valid for pens created with the `ExtCreatePen` API function. (See MS Windows SDK docs.) This applies to both VCL and VCL.NET.
psUserStyle	The pen uses a styling array supplied by the user. Thus style is only valid for pens created with the `ExtCreatePen` API function. (See MS Windows SDK docs.) This applies to both VCL and VCL.NET.

8.4.17 Graphics.TPixelFormat Enumeration

TPixelFormat specifies the format of a bitmap.

Pascal

```
TPixelFormat = (
  pfDevice,
  pf1bit,
  pf4bit,
  pf8bit,
  pf15bit,
  pf16bit,
  pf24bit,
  pf32bit,
  pfCustom
);
```

C++

```
enum TPixelFormat {
  pfDevice,
  pf1bit,
  pf4bit,
  pf8bit,
  pf15bit,
  pf16bit,
  pf24bit,
  pf32bit,
  pfCustom
};
```

File

Graphics

Description

TPixelFormat can have one of the following values:

Value	Meaning
pfDevice	The bitmap is stored as a device-dependent bitmap.
pf1bit	The bitmap is a device-independent bitmap with one bit per pixel (black and white palette)
pf4bit	The bitmap is a device-independent bitmap that uses a 16-color palette.
pf8bit	The bitmap is a device-independent bitmap that uses a 256color palette.
pf15bit	The bitmap is a device-independent true-color bitmap that uses 15 bits per pixel (RGB compression).
pf16bit	The bitmap is a device-independent true-color bitmap that uses 16 bits per pixel (bitfield compression).
pf24bit	The bitmap is a device-independent true-color bitmap that uses 24 bits per pixel.
pf32bit	The bitmap is a device-independent true-color bitmap that uses 32 bits per pixel (RGB compression).
pfCustom	The bitmap uses some other format. TBitmap (see page 2628) does not support pfCustom.

8.4.18 Graphics.TProgressStage Enumeration

TProgressStage indicates the stage of a lengthy operation.

Pascal

```
TProgressStage = (
  psStarting,
  psRunning,
  psEnding
);
```

C++

```
enum TProgressStage {
  psStarting,
  psRunning,
  psEnding
};
```

File

Graphics

Description

The following table lists the possible values:

Value	Meaning
psStarting	The operation is about to begin.
psRunning	The operation is underway and has not yet completed.
psEnding	The operation has just completed.

8.4.19 Graphics.TResData Record

Pascal

```
TResData = record
  Handle: THandle;
end;
```

C++

```
struct TResData {
  THandle Handle;
};
```

File

Graphics

Description

This is record Graphics.TResData.

8.4.20 Graphics.TResource Record

Pascal

```
TResource = record
  Next: PResource;
  RefCount: Integer;
  Handle: THandle;
  HashCode: Word;
  case Integer of
    0: (Data: TResData;);
    1: (Font: TFontData;);
    2: (Pen: TPenData;);
    3: (Brush: TBrushData;);
end;
```

C++

```
struct TResource {
  PResource Next;
  int RefCount;
  THandle Handle;
  Word HashCode;
  TResData Data;
  TFontData Font;
  TPenData Pen;
  TBrushData Brush;
};
```

File

Graphics

Description

This is record Graphics.TResource.

8.4.21 Graphics.TTextFormats Enumeration

Pascal

```
TTextFormats = (
  tfBottom,
  tfCalcRect,
  tfCenter,
  tfEditControl,
  tfEndEllipsis,
  tfPathEllipsis,
  tfExpandTabs,
  tfExternalLeading,
  tfLeft,
  tfModifyString,
  tfNoClip,
  tfNoPrefix,
  tfRight,
  tfRtlReading,
  tfSingleLine,
  tfTop,
  tfVerticalCenter,
  tfWordBreak
```

```
);
```

C++

```
enum TTextFormats {
  tfBottom,
  tfCalcRect,
  tfCenter,
  tfEditControl,
  tfEndEllipsis,
  tfPathEllipsis,
  tfExpandTabs,
  tfExternalLeading,
  tfLeft,
  tfModifyString,
  tfNoClip,
  tfNoPrefix,
  tfRight,
  tfRtlReading,
  tfSingleLine,
  tfTop,
  tfVerticalCenter,
  tfWordBreak
};
```

File

Graphics

Description

This is record Graphics.TTextFormats.

8.4.22 Graphics.TTransparentMode Enumeration

TTransparentMode indicates how the transparent color of a bitmap is determined.

Pascal

```
TTransparentMode = (
  tmAuto,
  tmFixed
);
```

C++

```
enum TTransparentMode {
  tmAuto,
  tmFixed
};
```

File

Graphics

Description

The following table lists the possible values:

Value	Meaning
tmAuto	The transparent color is determined by the bottom-leftmost pixel of the bitmap image.
tmFixed	The transparent color is explicitly assigned and stored in the bitmap object.

8.5 Types

The following table lists types in this documentation.

Types

Name	Description
HENHMETAFILE (see page 2875)	Meta file handle type.
HMETAFILE (see page 2876)	This is type Graphics.HMETAFILE.
PColor (see page 2876)	PColor is a pointer to a TColor (see page 2878) type.
PCursorOrIcon (see page 2876)	This is type Graphics.PCursorOrIcon.
PIconRec (see page 2877)	This is type Graphics.PIconRec.
PResource (see page 2877)	This is type Graphics.PResource.
TCanvasState (see page 2877)	This is type Graphics.TCanvasState.
TColor (see page 2878)	TColor is used to specify the color of a Windows-only control.
TCopyMode (see page 2880)	TCopyMode specifies how a graphical image is copied onto a canvas.
TFontCharset (see page 2881)	TFontCharset represents the character set of a font.
TFontDataName (see page 2882)	This is type Graphics.TFontDataName.
TFontName (see page 2882)	TFontName is a string that holds the typeface name of a font.
TFontStyles (see page 2883)	TFontStyle (see page 2866) and TFontStyles specify style characteristics of a font.
TFontStylesBase (see page 2887)	This is type Graphics.TFontStylesBase.
TGraphicClass (see page 2887)	TGraphicClass defines the metaclass for TGraphic (see page 2748).
TProgressEvent (see page 2888)	TProgressEvent is the type of event handlers that provide feedback during lengthy operations.
TTextFormat (see page 2889)	TTextFormat indicates how a text viewer control displays its contents.

8.5.1 Graphics.HENHMETAFILE Type

Meta file handle type.

Pascal

```
HENHMETAFILE = THandle;
```

C++

```
THandle HENHMETAFILE;
```

File

Graphics

Description

ReleaseHandle is called to release the resources used to represent the metafile. HENHMETAFILE is the data type passed to that method.

See Also

ReleaseHandle

8.5.2 Graphics.HMETAFILE Type

Pascal

```
HMETAFILE = THandle;
```

C++

```
THandle HMETAFILE;
```

File

Graphics

Description

This is type Graphics.HMETAFILE.

8.5.3 Graphics.PColor Type

PColor is a pointer to a TColor (see page 2878) type.

Pascal

```
PColor = ^TColor;
```

C++

```
^TColor PColor;
```

File

Graphics

Description

The TColor (see page 2878) type represents a color as an integer.

Note: Static arrays are declared with an immutable size.

See Also

TColor (see page 2878)

8.5.4 Graphics.PCursorOrIcon Type

Pascal

```
PCursorOrIcon = ^TCursorOrIcon;
```

C++

```
^TCursorOrIcon PCursorOrIcon;
```

File

Graphics

Description

This is type Graphics.PCursorOrIcon.

8.5.5 Graphics.PIconRec Type

Pascal

```
PIconRec = ^TIconRec;
```

C++

```
^TIconRec PIconRec;
```

File

Graphics

Description

This is type Graphics.PIconRec.

8.5.6 Graphics.PResource Type

Pascal

```
PResource = ^TResource;
```

C++

```
^TResource PResource;
```

File

Graphics

Description

This is type Graphics.PResource.

8.5.7 Graphics.TCanvasState Type

Pascal

```
TCanvasState = set of TCanvasStates;
```

C++

```
TCanvasStates TCanvasState;
```

File

Graphics

Description

This is type Graphics.TCanvasState.

8.5.8 Graphics.TColor Type

TColor is used to specify the color of a Windows-only control.

Pascal

```
TColor = -$7FFFFFFF-1..$7FFFFFFF;
```

C++

```
-$7FFFFFFF-1..$7FFFFFFF TColor;
```

File

Graphics

Description

TColor is used to specify the color of a Windows-only control. It is used by the Color property of many components and by a number of other properties that specify color values.

The Graphics (see page 2619) unit contains definitions of useful constants for TColor. These constants map either directly to the closest matching color in the system palette (for example, clBlue (see page 2899) maps to blue) or to the corresponding system screen element color defined in the Color section of the Windows Control panel (for example, clBtnFace (see page 2900) maps to the system color for button faces).

If you specify TColor as a specific 4-byte hexadecimal number instead of using the constants defined in the Graphics (see page 2619) unit, the low three bytes represent RGB color intensities for blue, green, and red, respectively. The value $00FF0000 (Delphi) or 0x00FF0000 (C++) represents full-intensity, pure blue, $0000FF00 (Delphi) or 0x0000FF00 (C++) is pure green, and $000000FF (Delphi) or 0x000000FF (C++) is pure red. $00000000 (Delphi) or 0x00000000 (C++) is black and $00FFFFFF (Delphi) or 0x00FFFFFF (C++) is white.

If the highest-order byte is zero, the color obtained is the closest matching color in the system palette. If the highest-order byte is one ($01 or 0x01), the color obtained is the closest matching color in the currently realized palette. If the highest-order byte is two ($02 or 0x02), the value is matched with the nearest color in the logical palette of the current device context.

The following tables lists the color constants from the Graphics (see page 2619) unit. The first two columns list the colors that map to the closest matching color in the system palette, while the last two columns list the colors that are defined in the Windows Control panel.

Value	Meaning	Value	Meaning
clNone (see page 2914)	White on Windows 9x, Black on NT.	clScrollBar (see page 2916)	Current color for the of scroll bar track.
clAqua (see page 2898)	Aqua	clBackground (see page 2899)	Current background color of the Windows desktop
clBlack (see page 2899)	Black	clActiveCaption (see page 2897)	Current color of the title bar of the active window
clBlue (see page 2899)	Blue	clInactiveCaption (see page 2908)	Current color of the title bar of inactive windows

clCream (see page 2902)	Cream	clMenu (see page 2912)	Current background color of menus
clDkGray (see page 2903)	Dark Gray	clWindow (see page 2965)	Current background color of windows
clFuchsia (see page 2903)	Fuchsia	clWindowFrame (see page 2966)	Current color of window frames
clGray (see page 2905)	Gray	clMenuText (see page 2913)	Current color of text on menus
clGreen (see page 2906)	Green	clWindowText (see page 2966)	Current color of text in windows
clLime (see page 2910)	Lime green	clCaptionText (see page 2902)	Current color of the text on the title bar of the active window
clLtGray (see page 2910)	Light Gray	clActiveBorder (see page 2897)	Current border color of the active window
clMaroon (see page 2911)	Maroon	clInactiveBorder (see page 2908)	Current border color of inactive windows
clMedGray (see page 2911)	Medium Gray	clAppWorkSpace (see page 2898)	Current color of the application workspace
clMoneyGreen (see page 2913)	Mint green	clHighlight (see page 2906)	Current background color of selected text
clNavy (see page 2914)	Navy blue	clHightlightText	Current color of selected text
clOlive (see page 2915)	Olive green	clBtnFace (see page 2900)	Current color of a button face
clPurple (see page 2915)	Purple	clBtnShadow (see page 2901)	Current color of a shadow cast by a button
clRed (see page 2916)	Red	clGrayText (see page 2905)	Current color of text that is dimmed
clSilver (see page 2916)	Silver	clBtnText (see page 2901)	Current color of text on a button
clSkyBlue (see page 2917)	Sky blue	clInactiveCaptionText (see page 2908)	Current color of the text on the title bar of an inactive window
clTeal (see page 2918)	Teal	clBtnHighlight (see page 2900)	Current color of the highlighting on a button
clWhite (see page 2965)	White	cl3DDkShadow (see page 2896)	Windows 95 or NT 4.0 only: Dark shadow for three-dimensional display elements

clYellow (see page 2967)	Yellow	cl3DLight (see page 2896)	Windows 95 or NT 4.0 only: Light color for three-dimensional display elements (for edges facing the light source)
		clInfoText (see page 2909)	Windows 95 or NT 4.0 only: Text color for tool tip controls
		clInfoBk (see page 2909)	Windows 95 or NT 4.0 only: Background color for tool tip controls
		clGradientActiveCaption (see page 2904)	Windows 98 or Windows 2000: Right side color in the color gradient of an active window's title bar. clActiveCaption (see page 2897) specifies the left side color.
		clGradientInactiveCaption (see page 2904)	Windows 98 or Windows 2000: Right side color in the color gradient of an inactive window's title bar. clInactiveCaption (see page 2908) specifies the left side color.
		clDefault (see page 2903)	The default color for the control to which the color is assigned.

8.5.9 Graphics.TCopyMode Type

TCopyMode specifies how a graphical image is copied onto a canvas.

Pascal

```
TCopyMode = Longint;
```

C++

```
Longint TCopyMode;
```

File

Graphics

Description

TCopyMode values describe how to combine the colors of a source bitmap and a destination bitmap.

The Windows unit defines the following constants for TCopyMode values:

Value	Meaning
cmBlackness (see page 2967)	Fills the destination rectangle on the canvas with black.
cmDstInvert (see page 2968)	Inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 2968)	Combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 2969)	Combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.

cmNotSrcCopy (see page 2969)	Copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 2970)	Combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.
cmPatCopy (see page 2970)	Copies the source pattern to the canvas.
cmPatInvert (see page 2971)	Combines the source pattern with the image on the canvas using the Boolean XOR operator
cmPatPaint (see page 2971)	Combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 2972)	Combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 2972)	Copies the source bitmap to the canvas.
cmSrcErase (see page 2973)	Inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 2973)	Combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 2974)	Combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 2974)	Fills the destination rectangle on the canvas with white.

8.5.10 Graphics.TFontCharset Type

TFontCharset represents the character set of a font.

Pascal

```
TFontCharset = 0..255;
```

C++

```
0..255 TFontCharset;
```

File

Graphics

Description

Each Windows typeface supports one or more character sets, which indicate what characters can be written using the font. For example, most fonts that can represent japanese characters use a particular multi-byte character set.

Each font supports its own, unique set of character sets. Check with the font vendor to determine which character sets are supported.

The following table lists the predefined constants provided for standard character sets:

Constant	Value	Description
ANSI_CHARSET	0	ANSI characters.
DEFAULT_CHARSET	1	Font is chosen based solely on Name and Size. If the described font is not available on the system, Windows will substitute another font.
SYMBOL_CHARSET	2	Standard symbol set.
MAC_CHARSET	77	Macintosh characters. Not available on NT 3.51.
SHIFTJIS_CHARSET	128	Japanese shift-JIS characters.
HANGEUL_CHARSET	129	Korean characters (Wansung).
JOHAB_CHARSET	130	Korean characters (Johab). Not available on NT 3.51
GB2312_CHARSET	134	Simplified Chinese characters (mainland china).
CHINESEBIG5_CHARSET	136	Traditional Chinese characters (Taiwanese).
GREEK_CHARSET	161	Greek characters. Not available on NT 3.51.
TURKISH_CHARSET	162	Turkish characters. Not available on NT 3.51
VIETNAMESE_CHARSET	163	Vietnamese characters. Not available on NT 3.51.
HEBREW_CHARSET	177	Hebrew characters. Not available on NT 3.51
ARABIC_CHARSET	178	Arabic characters. Not available on NT 3.51
BALTIC_CHARSET	186	Baltic characters. Not available on NT 3.51.
RUSSIAN_CHARSET	204	Cyrillic characters. Not available on NT 3.51.
THAI_CHARSET	222	Thai characters. Not available on NT 3.51
EASTEUROPE_CHARSET	238	Includes diacritical marks for eastern european countries. Not available on NT 3.51.
OEM_CHARSET	255	Depends on the codepage of the operating system.

8.5.11 Graphics.TFontDataName Type

Pascal

```
TFontDataName = string[(LF_FACESIZE - 1) * 4];
```

C++

```
string[(LF_FACESIZE - 1) * 4] TFontDataName;
```

File

Graphics

Description

This is type Graphics.TFontDataName.

8.5.12 Graphics.TFontName Type

TFontName is a string that holds the typeface name of a font.

Pascal

```
TFontName = type string;
```

C++

```
type string TFontName;
```

File

Graphics

Description

TFontName maps to a string. It is the type for representing the typeface name of a font.

8.5.13 Graphics.TFontStyles Type

TFontStyle (see page 2866) and TFontStyles specify style characteristics of a font.

Pascal

```
TFontStyles = set of TFontStyle;
```

C++

```
TFontStyle TFontStyles;
```

File

Graphics

Description

TFontStyles is a set of TFontStyle (see page 2866) values. The following table lists the possible values of the TFontStyle (see page 2866) type:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

C++ Examples:

```
/*
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
*/
// global variables to show the order of events
int  XPos, YPos, Delta;

// This is a utility procedure to display messages
// add this at the beginning of the unit

void StatusMsg(TForm *Form, TCanvas *Canvas, char *Message, bool Bkg)
{
  if (!Bkg)
    Canvas->Font->Style = TFontStyles() << fsBold; // foreground messages are bold
  Canvas->TextOut(XPos, YPos, Message);
```

```
    if (!Bkg)
      Canvas->Font->Style = TFontStyles();
    // change Xpos and YPos to prepare for the next message
    YPos += Delta;
    if (YPos >= Form->ClientHeight - 10)
    {
      YPos = 10;
      XPos += 180;
    }
    if (XPos >= Form->ClientWidth - 100)
    {
      if (Canvas->Font->Color == clRed)
        Canvas->Font->Color = clBlack;
      else
        Canvas->Font->Color = clRed;
      XPos = 10;
    }
  }

  /*
  This is the OnCreate event handler for Form1.  It
  initializes global values and sets the application's OnIdle
  event handler
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Button1->Caption = "Do not yield";
    Button2->Caption  = "Handle Message";
    Application->OnIdle = MyIdleHandler;
    XPos = 10;
    YPos = 10;
    if (Canvas->Font->Height > 0)
      Delta = Canvas->Font->Height + 1;
    else
      Delta = 1 - Canvas->Font->Height;
  }

  /*
  This is the OnIdle event handler. It is set in the Form's
  OnCreate event handler, so you need only add it as a private
  method of the form.  Usually it would perform some background
  processing for the application.  This one simply writes a
  message to let you know when it's there.
  */
  void __fastcall TForm1::MyIdleHandler(TObject *Sender, bool &Done)
  {
    StatusMsg(Form1, Canvas, "This represents a background process.", true);
  }

  /*
  This is the OnClick event handler for Button1.  It
  simulates a lengthy process
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    int X, Y;
    StatusMsg(Form1, Canvas, "The Button1Click handler is starting", false);
    for (int I = 0; I < 100; I++)
    {
      randomize();
      for (int J = 0; J < 100; J++)
        Y = random(J);
      X = random(I);
    }
    StatusMsg(Form1, Canvas, "The Button1Click handler is done", false);
  }
```

```
/*
This is the OnClick event handler for button2.  It
simulates a lengthy process that allows other messages to be
processed.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button2Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
    {
      Y = random(J);
    }
    // Yield to OnIdle or other messages
    Application->HandleMessage();
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button2Click handler is done", false);
}
```

Delphi Examples:

```
{
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
}
var
{ global variables to show the order of events. }
  XPos, YPos, Delta: integer;

{ This is a utility procedure to display messages }
{ add this at the beginning of the implementation section }
procedure StatusMsg(
  MyForm : TForm1; Canvas : TCanvas; Message : string; Bkg : Boolean);
begin
  if not bkg then
    Canvas.Font.Style := [fsBold]; {foreground messages are bold }
  Canvas.TextOut(XPos, YPos, Message);
  if not bkg then
    Canvas.Font.Style := [];
  { change Xpos and YPos to prepare for the next message }
  Ypos := Ypos + Delta;
  if YPos >= MyForm.ClientHeight - 10 then
  begin
    YPos := 10;
    Xpos := Xpos + 180;
  end;
  if (Xpos >= MyForm.ClientWidth - 100) then
  begin
    if (Canvas.Font.Color = clRed) then
      Canvas.Font.Color := clBlack
    else
      Canvas.Font.Color := clRed;
    Xpos := 10;
  end;
end;

{This is the Form's OnCreate event handler.  }
{ It initializes global variables and sets the  the OnIdle
event handler }
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Button1.Caption := 'Do not yield';
  Button2.Caption := 'Handle Message';
  Application.OnIdle:= MyIdleHandler;
  XPos := 10;
  YPos := 10;
  Delta := Abs(Canvas.Font.Height) + 1;
 end;

{
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some
background processing for the application.  This one simply
writes a message to let you know when it's there.
}
procedure TForm1.MyIdleHandler(Sender: TObject; var Done: Boolean);
begin
  StatusMsg(
    Form1, Canvas, 'This represents a background process.', True);
end;

{
Set this method as the OnClick event handler of Button1.  It
performs a calculation without yielding to other messages or
idle time.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is starting', False);
  I := 0;
  J := 0;
  while I < 10 do
  begin
    Randomize;
    while J < 10 do
    begin
      Y := Random(J);
      Inc(J);
    end;
    X := Random(I);
    Inc(I);
  end;
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is done', False);
end;

{
Set this method as the OnClick event handler of Button2.  It
performs a calculation but calls HandleMessage to allow idle
time or asynchronous message processing.
}
procedure TForm1.Button2Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
 StatusMsg(
    Form1, Canvas, 'The asynchronous handler is starting', False);
  I := 0;
  J := 0;
  while I < 10 do
  begin
    Randomize;
```

```
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
      { yield to OnIdle or other messages }
      Application.HandleMessage;
    end;
    StatusMsg(
      Form1, Canvas, 'The asynchronous handler is done', False);
  end;
```

8.5.14 Graphics.TFontStylesBase Type

Pascal

```
TFontStylesBase = set of TFontStyle;
```

C++

```
TFontStyle TFontStylesBase;
```

File

Graphics

Description

This is type Graphics.TFontStylesBase.

8.5.15 Graphics.TGraphicClass Type

TGraphicClass defines the metaclass for TGraphic (see page 2748).

Pascal

```
TGraphicClass = class of TGraphic;
```

C++

```
TGraphic TGraphicClass;
```

File

Graphics

Description

TGraphicClass is the metaclass for TGraphic (see page 2748). Its value is the class reference for TGraphic (see page 2748) or for one of its descendants.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
```

```
*/
#include <memory>       //for STL auto_ptr class

TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TClassFinder> myClassFinder(new TClassFinder(MetaClass, False));
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
  { Usually, this goes in the initialization section where it is only executed once }
  RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
  try
    myClassFinder := TClassFinder.Create(TIcon, False);
    mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
    Edit1.Text := GraphicExtension(mygraphclass);
  except
    on EAccessViolation do
      MessageDlg('Invalid class name.', mtError, [mbOK], 0)
  end;
end;
```

8.5.16 Graphics.TProgressEvent Type

TProgressEvent is the type of event handlers that provide feedback during lengthy operations.

Pascal

```
TProgressEvent = procedure (Sender: TObject; Stage: TProgressStage; PercentDone: Byte;
RedrawNow: Boolean; const R: TRect; const Msg: string) of object;
```

C++

```
(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow: Boolean; const R:
TRect; const Msg: string) ( TProgressEvent)();
```

File

Graphics

Description

Sender is the object that receives the event notification. This is typically the component that initiated the lengthy operation.

Stage indicates whether the operation is beginning, continuing, or ending. If the event handler displays an indicator such as a progress bar, the indicator can be created when Stage is psStarting, updated while Stage is psRunning, and removed when Stage is psEnding.

PercentDone is an approximation of how much of the operation has completed. Use PercentDone to update the position of a progress bar or other indicator.

RedrawNow indicates whether the partial results of the operation can safely be drawn on screen at this point.

R gives the dimensions of the portion of the display that has changed and needs to be redrawn.

Msg contains one or two words that describe what operation is occurring. For example, the value of Msg could be a string such as Loading, Storing, or Reducing colors. The Msg string can also be empty.

8.5.17 Graphics.TTextFormat Type

TTextFormat indicates how a text viewer control displays its contents.

Pascal

```
TTextFormat = set of TTextFormats;
```

C++

```
TTextFormats TTextFormat;
```

File

Graphics

Description

TTextFormat has the following values:

Value	Meaning
tfAutoText	The control attempts to determine whether Text contains rich text. If a valid tag is seen on the first line, Text is assumed to be a rich text document. Otherwise, Text is displayed as plain text.
tfText	The control assumes that Text contains rich text, even if no tags are present. Note that in rich text, any sequence consisting entirely of spaces, tabs, and line breaks counts as a single space.
tfPlainText	The control assumes that Text contains plain text.

8.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DDBsOnly (see page 2890)	Determines whether bitmap handles are created as device-dependent bitmaps (DDB).
DefFontData (see page 2891)	This is variable Graphics.DefFontData.
SystemPalette16 (see page 2891)	Contains the 16 VGA system colors.

8.6.1 Graphics.DDBsOnly Variable

Determines whether bitmap handles are created as device-dependent bitmaps (DDB).

Pascal

```
DDBsOnly: Boolean = False;
```

C++

```
Boolean DDBsOnly = False;
```

File

Graphics

Description

Set DDBsOnly to force all bitmap handles to be created as device-dependent bitmaps (DDB). The default bitmap handle format is DIB (device-independent bitmap).

Setting DDBsOnly to true undercuts much of the graphics architecture; it should not be used unless it is absolutely required. Examples of when to set it to true are:

A graphics-intensive application does bitmap operations that Windows does not support with DIBs; you can either set each bitmap object to DDB before you access its handle property, or you can set this global variable to Truetrue.

An application uses hundreds of very small bitmaps. Each DIB consumes a separate 4k memory page. DIBs are less efficient for storing many small images than DDBs. Before resorting to changing the DDBsOnly flag to address this, however, consider switching your code to use image lists instead of individual tiny bitmaps. Image lists are more efficient than DIBs and DDBs for storing many same-size small images.

Note: Setting DDBsOnly to true introduces the issue of image alteration. When a BMP file or resource is loaded into a DDB, the color information in the original image is changed to fit the pixel format of the current video device and mode. If you read an image, touch its handle, and write it out, you will not get the same file you read in. DIBs solve this problem. Do not change DDBsOnly unless you have a specific need for device-dependent bitmaps and are willing to accept the image-altering DDB side effects.

See Also

GetDIB (see page 2849)

TImageList

8.6.2 Graphics.DefFontData Variable

Pascal

```
DefFontData: TFontData = ( Handle: 0; Height: 0; Orientation: 0; Pitch: fpDefault; Style: [];
Charset : DEFAULT_CHARSET; Name: 'MS Sans Serif');
```

C++

```
TFontData DefFontData = ( Handle: 0; Height: 0; Orientation: 0; Pitch: fpDefault; Style: [];
Charset : DEFAULT_CHARSET; Name: 'MS Sans Serif');
```

File

Graphics

Description

This is variable Graphics.DefFontData.

8.6.3 Graphics.SystemPalette16 Variable

Contains the 16 VGA system colors.

Pascal

```
SystemPalette16: HPalette;
```

C++

```
HPalette SystemPalette16;
```

File

Graphics

Description

SystemPalette16 holds a palette handle that contains the 16 VGA system colors. This palette is built at runtime from the system colors defined by the current video card and driver. The actual color values of the stock VGA palette can vary from one video card vendor to the next. Therefore it is not safe to assume, for example, that clGray (see page 2905) will be RGB(128,128,128) on all systems.

This handle is available as a convenience. This is the palette that VCL forces on all 16 color bitmaps that are loaded from files or resources.

Note: Using custom 16-color palettes does not work on 16-color video systems. It is problematic on many 256 color drivers because the variation in colors interferes with transparency operations.

See Also

TGraphic (see page 2748)

Graphics

8.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cl3DDkShadow (see page 2896)	Represents the dark shadow of 3D components.
cl3DLight (see page 2896)	Represents the light part of 3D components.
clActiveBorder (see page 2897)	Represents the default border color of an active window.
clActiveCaption (see page 2897)	Represents the default color of the title bar of an active window.
clAppWorkSpace (see page 2898)	Represents the default Application WorkSpace color.
clAqua (see page 2898)	Represents the color Aqua.
clBackground (see page 2899)	Represents the default Windows background color.
clBlack (see page 2899)	Represents the color black.
clBlue (see page 2899)	Represents the color blue.
clBtnFace (see page 2900)	Represents the color of the face of a button.
clBtnHighlight (see page 2900)	Represents the color of the highlighting on a button.
clBtnShadow (see page 2901)	Represents the color of the shadow cast by of a button.
clBtnText (see page 2901)	Represents the default color of the text on a button.
clCaptionText (see page 2902)	Represents the default color of the text on the title bar of an active window.
clCream (see page 2902)	Represents the color cream.
clDefault (see page 2903)	Specifies the default color of a component.
clDkGray (see page 2903)	Represents the color dark gray.
clFuchsia (see page 2903)	Represents the color fuchsia.
clGradientActiveCaption (see page 2904)	Right side color in the color gradient of an active window's title bar.
clGradientInactiveCaption (see page 2904)	Right side color in the color gradient of an inactive window's title bar.
clGray (see page 2905)	Represents the color gray.
clGrayText (see page 2905)	Represents the color of text that is grayed out.
clGreen (see page 2906)	Represents the color green.
clHighlight (see page 2906)	Represents the color of the background of a block of selected text.
clHighlightText (see page 2907)	Represents the color of selected text.
clHotLight (see page 2907)	This is constant Graphics.clHotLight.
clInactiveBorder (see page 2908)	Represents the default border color of an inactive window.
clInactiveCaption (see page 2908)	Represents the default color of the title bar of an inactive window.
clInactiveCaptionText (see page 2908)	Represents the default color of the title text in an inactive window.
clInfoBk (see page 2909)	Represents the color of the background of a tool tip.
clInfoText (see page 2909)	Represents the color of tool tip text.
clLime (see page 2910)	Represents the color lime.
clLtGray (see page 2910)	Represents the color light gray.
clMaroon (see page 2911)	Represents the color maroon.
clMedGray (see page 2911)	Represents the color medium gray.
clMenu (see page 2912)	Represents the color of the background of an unselected menu sub item.
clMenuBar (see page 2912)	Represents the color of the background of an unselected main menu item.
clMenuHighlight (see page 2912)	Represents the color of the background of a selected menu item.
clMenuText (see page 2913)	Represents the default menu text color.
clMoneyGreen (see page 2913)	Represents the color money green.
clNavy (see page 2914)	Represents the color navy.
clNone (see page 2914)	Represents no selected color.
clOlive (see page 2915)	Represents the color olive.
clPurple (see page 2915)	Represents the color purple.

clRed (see page 2916)	Represents the color red.
clScrollBar (see page 2916)	This is constant Graphics.clScrollBar.
clSilver (see page 2916)	Represents the color silver.
clSkyBlue (see page 2917)	Represents the color sky blue.
clSystemColor (see page 2917)	Is a mask used to identify the set of system colors.
clTeal (see page 2918)	Represents the color teal.
clWebAliceBlue (see page 2918)	This is constant Graphics.clWebAliceBlue.
clWebAntiqueWhite (see page 2919)	This is constant Graphics.clWebAntiqueWhite.
clWebAqua (see page 2919)	This is constant Graphics.clWebAqua.
clWebAquamarine (see page 2919)	This is constant Graphics.clWebAquamarine.
clWebAzure (see page 2920)	This is constant Graphics.clWebAzure.
clWebBeige (see page 2920)	This is constant Graphics.clWebBeige.
clWebBisque (see page 2920)	This is constant Graphics.clWebBisque.
clWebBlack (see page 2921)	This is constant Graphics.clWebBlack.
clWebBlanchedAlmond (see page 2921)	This is constant Graphics.clWebBlanchedAlmond.
clWebBlue (see page 2921)	This is constant Graphics.clWebBlue.
clWebBlueViolet (see page 2922)	This is constant Graphics.clWebBlueViolet.
clWebBrown (see page 2922)	This is constant Graphics.clWebBrown.
clWebBurlywood (see page 2922)	This is constant Graphics.clWebBurlywood.
clWebCadetBlue (see page 2923)	This is constant Graphics.clWebCadetBlue.
clWebChartreuse (see page 2923)	This is constant Graphics.clWebChartreuse.
clWebChocolate (see page 2923)	This is constant Graphics.clWebChocolate.
clWebCoral (see page 2924)	This is constant Graphics.clWebCoral.
clWebCornFlowerBlue (see page 2924)	This is constant Graphics.clWebCornFlowerBlue.
clWebCornSilk (see page 2924)	This is constant Graphics.clWebCornSilk.
clWebCrimson (see page 2925)	This is constant Graphics.clWebCrimson.
clWebCyan (see page 2925)	This is constant Graphics.clWebCyan.
clWebDarkBlue (see page 2925)	This is constant Graphics.clWebDarkBlue.
clWebDarkCyan (see page 2926)	This is constant Graphics.clWebDarkCyan.
clWebDarkGoldenRod (see page 2926)	This is constant Graphics.clWebDarkGoldenRod.
clWebDarkGray (see page 2926)	This is constant Graphics.clWebDarkGray.
clWebDarkgreen (see page 2927)	This is constant Graphics.clWebDarkgreen.
clWebDarkKhaki (see page 2927)	This is constant Graphics.clWebDarkKhaki.
clWebDarkMagenta (see page 2927)	This is constant Graphics.clWebDarkMagenta.
clWebDarkOliveGreen (see page 2928)	This is constant Graphics.clWebDarkOliveGreen.
clWebDarkOrange (see page 2928)	This is constant Graphics.clWebDarkOrange.
clWebDarkOrchid (see page 2928)	This is constant Graphics.clWebDarkOrchid.
clWebDarkRed (see page 2929)	This is constant Graphics.clWebDarkRed.
clWebDarkSalmon (see page 2929)	This is constant Graphics.clWebDarkSalmon.
clWebDarkSeaGreen (see page 2929)	This is constant Graphics.clWebDarkSeaGreen.
clWebDarkSlateBlue (see page 2930)	This is constant Graphics.clWebDarkSlateBlue.
clWebDarkSlategray (see page 2930)	This is constant Graphics.clWebDarkSlategray.
clWebDarkTurquoise (see page 2930)	This is constant Graphics.clWebDarkTurquoise.
clWebDarkViolet (see page 2931)	This is constant Graphics.clWebDarkViolet.
clWebDeepPink (see page 2931)	This is constant Graphics.clWebDeepPink.
clWebDeepskyBlue (see page 2931)	This is constant Graphics.clWebDeepskyBlue.
clWebDimGray (see page 2932)	This is constant Graphics.clWebDimGray.
clWebDodgerBlue (see page 2932)	This is constant Graphics.clWebDodgerBlue.
clWebFirebrick (see page 2932)	This is constant Graphics.clWebFirebrick.
clWebFloralWhite (see page 2933)	This is constant Graphics.clWebFloralWhite.
clWebForestGreen (see page 2933)	This is constant Graphics.clWebForestGreen.
clWebFuchsia (see page 2933)	This is constant Graphics.clWebFuchsia.
clWebGainsboro (see page 2934)	This is constant Graphics.clWebGainsboro.
clWebGhostWhite (see page 2934)	This is constant Graphics.clWebGhostWhite.

clWebGold (see page 2934)	This is constant Graphics.clWebGold.
clWebGoldenRod (see page 2935)	This is constant Graphics.clWebGoldenRod.
clWebGray (see page 2935)	This is constant Graphics.clWebGray.
clWebGreen (see page 2935)	This is constant Graphics.clWebGreen.
clWebGreenYellow (see page 2936)	This is constant Graphics.clWebGreenYellow.
clWebHoneydew (see page 2936)	This is constant Graphics.clWebHoneydew.
clWebHotPink (see page 2936)	This is constant Graphics.clWebHotPink.
clWebIndianRed (see page 2937)	This is constant Graphics.clWebIndianRed.
clWebIndigo (see page 2937)	This is constant Graphics.clWebIndigo.
clWebIvory (see page 2937)	This is constant Graphics.clWebIvory.
clWebKhaki (see page 2938)	This is constant Graphics.clWebKhaki.
clWebLavender (see page 2938)	This is constant Graphics.clWebLavender.
clWebLavenderBlush (see page 2938)	This is constant Graphics.clWebLavenderBlush.
clWebLawnGreen (see page 2939)	This is constant Graphics.clWebLawnGreen.
clWebLemonChiffon (see page 2939)	This is constant Graphics.clWebLemonChiffon.
clWebLightBlue (see page 2939)	This is constant Graphics.clWebLightBlue.
clWebLightCoral (see page 2940)	This is constant Graphics.clWebLightCoral.
clWebLightCyan (see page 2940)	This is constant Graphics.clWebLightCyan.
clWebLightGoldenrodYellow (see page 2940)	This is constant Graphics.clWebLightGoldenrodYellow.
clWebLightGreen (see page 2941)	This is constant Graphics.clWebLightGreen.
clWebLightgrey (see page 2941)	This is constant Graphics.clWebLightgrey.
clWebLightPink (see page 2941)	This is constant Graphics.clWebLightPink.
clWebLightSalmon (see page 2942)	This is constant Graphics.clWebLightSalmon.
clWebLightSeaGreen (see page 2942)	This is constant Graphics.clWebLightSeaGreen.
clWebLightSkyBlue (see page 2942)	This is constant Graphics.clWebLightSkyBlue.
clWebLightSlateGray (see page 2943)	This is constant Graphics.clWebLightSlateGray.
clWebLightSteelBlue (see page 2943)	This is constant Graphics.clWebLightSteelBlue.
clWebLightYellow (see page 2943)	This is constant Graphics.clWebLightYellow.
clWebLime (see page 2944)	This is constant Graphics.clWebLime.
clWebLimeGreen (see page 2944)	This is constant Graphics.clWebLimeGreen.
clWebLinen (see page 2944)	This is constant Graphics.clWebLinen.
clWebMagenta (see page 2945)	This is constant Graphics.clWebMagenta.
clWebMaroon (see page 2945)	This is constant Graphics.clWebMaroon.
clWebMediumAquamarine (see page 2945)	This is constant Graphics.clWebMediumAquamarine.
clWebMediumBlue (see page 2946)	This is constant Graphics.clWebMediumBlue.
clWebMediumOrchid (see page 2946)	This is constant Graphics.clWebMediumOrchid.
clWebMediumPurple (see page 2946)	This is constant Graphics.clWebMediumPurple.
clWebMediumSeaGreen (see page 2947)	This is constant Graphics.clWebMediumSeaGreen.
clWebMediumSlateBlue (see page 2947)	This is constant Graphics.clWebMediumSlateBlue.
clWebMediumSpringGreen (see page 2947)	This is constant Graphics.clWebMediumSpringGreen.
clWebMediumTurquoise (see page 2948)	This is constant Graphics.clWebMediumTurquoise.
clWebMediumVioletRed (see page 2948)	This is constant Graphics.clWebMediumVioletRed.
clWebMidnightBlue (see page 2948)	This is constant Graphics.clWebMidnightBlue.
clWebMintcream (see page 2949)	This is constant Graphics.clWebMintcream.
clWebMistyRose (see page 2949)	This is constant Graphics.clWebMistyRose.
clWebMoccasin (see page 2949)	This is constant Graphics.clWebMoccasin.
clWebNavajoWhite (see page 2950)	This is constant Graphics.clWebNavajoWhite.
clWebNavy (see page 2950)	This is constant Graphics.clWebNavy.
clWebOldLace (see page 2950)	This is constant Graphics.clWebOldLace.
clWebOlive (see page 2951)	This is constant Graphics.clWebOlive.
clWebOliveDrab (see page 2951)	This is constant Graphics.clWebOliveDrab.
clWebOrange (see page 2951)	This is constant Graphics.clWebOrange.
clWebOrangeRed (see page 2952)	This is constant Graphics.clWebOrangeRed.
clWebOrchid (see page 2952)	This is constant Graphics.clWebOrchid.

clWebPaleGoldenrod (see page 2952)	This is constant Graphics.clWebPaleGoldenrod.
clWebPaleGreen (see page 2953)	This is constant Graphics.clWebPaleGreen.
clWebPaleTurquoise (see page 2953)	This is constant Graphics.clWebPaleTurquoise.
clWebPaleVioletRed (see page 2953)	This is constant Graphics.clWebPaleVioletRed.
clWebPapayaWhip (see page 2954)	This is constant Graphics.clWebPapayaWhip.
clWebPeachPuff (see page 2954)	This is constant Graphics.clWebPeachPuff.
clWebPeru (see page 2954)	This is constant Graphics.clWebPeru.
clWebPink (see page 2955)	This is constant Graphics.clWebPink.
clWebPlum (see page 2955)	This is constant Graphics.clWebPlum.
clWebPowderBlue (see page 2955)	This is constant Graphics.clWebPowderBlue.
clWebPurple (see page 2956)	This is constant Graphics.clWebPurple.
clWebRed (see page 2956)	This is constant Graphics.clWebRed.
clWebRosyBrown (see page 2956)	This is constant Graphics.clWebRosyBrown.
clWebRoyalBlue (see page 2957)	This is constant Graphics.clWebRoyalBlue.
clWebSaddleBrown (see page 2957)	This is constant Graphics.clWebSaddleBrown.
clWebSalmon (see page 2957)	This is constant Graphics.clWebSalmon.
clWebSandyBrown (see page 2958)	This is constant Graphics.clWebSandyBrown.
clWebSeaGreen (see page 2958)	This is constant Graphics.clWebSeaGreen.
clWebSeashell (see page 2958)	This is constant Graphics.clWebSeashell.
clWebSienna (see page 2959)	This is constant Graphics.clWebSienna.
clWebSilver (see page 2959)	This is constant Graphics.clWebSilver.
clWebSkyBlue (see page 2959)	This is constant Graphics.clWebSkyBlue.
clWebSlateBlue (see page 2960)	This is constant Graphics.clWebSlateBlue.
clWebSlateGray (see page 2960)	This is constant Graphics.clWebSlateGray.
clWebSnow (see page 2960)	This is constant Graphics.clWebSnow.
clWebSpringGreen (see page 2961)	This is constant Graphics.clWebSpringGreen.
clWebSteelBlue (see page 2961)	This is constant Graphics.clWebSteelBlue.
clWebTan (see page 2961)	This is constant Graphics.clWebTan.
clWebTeal (see page 2962)	This is constant Graphics.clWebTeal.
clWebThistle (see page 2962)	This is constant Graphics.clWebThistle.
clWebTomato (see page 2962)	This is constant Graphics.clWebTomato.
clWebTurquoise (see page 2963)	This is constant Graphics.clWebTurquoise.
clWebViolet (see page 2963)	This is constant Graphics.clWebViolet.
clWebWheat (see page 2963)	This is constant Graphics.clWebWheat.
clWebWhite (see page 2964)	This is constant Graphics.clWebWhite.
clWebWhiteSmoke (see page 2964)	This is constant Graphics.clWebWhiteSmoke.
clWebYellow (see page 2964)	This is constant Graphics.clWebYellow.
clWebYellowGreen (see page 2965)	This is constant Graphics.clWebYellowGreen.
clWhite (see page 2965)	Represents the color white.
clWindow (see page 2965)	Represents the default Window background color.
clWindowFrame (see page 2966)	Represents the default Window frame background color.
clWindowText (see page 2966)	Represents the default Window text color.
clYellow (see page 2967)	Represents the color yellow.
cmBlackness (see page 2967)	TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with black.
cmDstInvert (see page 2968)	TCopyMode (see page 2880) value : inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 2968)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 2969)	TCopyMode (see page 2880) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.
cmNotSrcCopy (see page 2969)	TCopyMode (see page 2880) value : copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 2970)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.
cmPatCopy (see page 2970)	TCopyMode (see page 2880) value : copies the source pattern to the canvas.

cmPatInvert (see page 2971)	TCopyMode (see page 2880) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.
cmPatPaint (see page 2971)	TCopyMode (see page 2880) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 2972)	TCopyMode (see page 2880) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 2972)	TCopyMode (see page 2880) value : copies the source bitmap to the canvas.
cmSrcErase (see page 2973)	TCopyMode (see page 2880) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 2973)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 2974)	TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 2974)	TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with white.
ExtendedColorsCount (see page 2975)	The total number of extended colors available
rc3_Cursor (see page 2976)	This is constant Graphics.rc3_Cursor.
rc3_Icon (see page 2976)	This is constant Graphics.rc3_Icon.
rc3_StockIcon (see page 2976)	This is constant Graphics.rc3_StockIcon.
StandardColorsCount (see page 2977)	The total number of standard colors available
WebColorsCount (see page 2978)	The total number of web colors available

8.7.1 Graphics.cl3DDkShadow Constant

Represents the dark shadow of 3D components.

Pascal

```
cl3DDkShadow = TColor(clSystemColor or COLOR_3DDKSHADOW);
```

C++

```
cl3DDkShadow = TColor(clSystemColor or COLOR_3DDKSHADOW);
```

File

Graphics

Description

For Windows 95 or NT 4.0 only, cl3DDkShadow represents the dark shadow for three-dimensional display elements.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.2 Graphics.cl3DLight Constant

Represents the light part of 3D components.

Pascal

```
cl3DLight = TColor(clSystemColor or COLOR_3DLIGHT);
```

C++

```
cl3DLight = TColor(clSystemColor or COLOR_3DLIGHT);
```

File

Graphics

Description

For Windows 95 or NT 4.0 only, cl3DLight represents the light color for three-dimensional display elements (for edges facing the light source).

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.3 Graphics.clActiveBorder Constant

Represents the default border color of an active window.

Pascal

```
clActiveBorder = TColor(clSystemColor or COLOR_ACTIVEBORDER);
```

C++

```
clActiveBorder = TColor(clSystemColor or COLOR_ACTIVEBORDER);
```

File

Graphics

Description

clActiveBorder is the default color used to paint the border of an active window. clActiveBorder is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.4 Graphics.clActiveCaption Constant

Represents the default color of the title bar of an active window.

Pascal

```
clActiveCaption = TColor(clSystemColor or COLOR_ACTIVECAPTION);
```

C++

```
clActiveCaption = TColor(clSystemColor or COLOR_ACTIVECAPTION);
```

File

Graphics

Description

clActiveCaption is the default color used to paint the border of an active window. clActiveCaption is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.5 Graphics.clAppWorkSpace Constant

Represents the default Application WorkSpace color.

Pascal

```
clAppWorkSpace = TColor(clSystemColor or COLOR_APPWORKSPACE);
```

C++

```
clAppWorkSpace = TColor(clSystemColor or COLOR_APPWORKSPACE);
```

File

Graphics

Description

clAppWorkSpace is the default Application WorkSpace color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.6 Graphics.clAqua Constant

Represents the color Aqua.

Pascal

```
clAqua = TColor($FFFF00);
```

C++

```
clAqua = TColor($FFFF00);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to Aqua. Aqua is the Delphi name for cyan - 100% blue and green with no red. Aqua is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.7 Graphics.clBackground Constant

Represents the default Windows background color.

Pascal

```
clBackground = TColor(clSystemColor or COLOR_BACKGROUND);
```

C++

```
clBackground = TColor(clSystemColor or COLOR_BACKGROUND);
```

File

Graphics

Description

clBackground is the default Windows background color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.8 Graphics.clBlack Constant

Represents the color black.

Pascal

```
clBlack = TColor($000000);
```

C++

```
clBlack = TColor($000000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to black. Black is the complete absence of blue, green and red. Black is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.9 Graphics.clBlue Constant

Represents the color blue.

Pascal

```
clBlue = TColor($FF0000);
```

C++

```
clBlue = TColor($FF0000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to blue. Blue is 100% blue with no green or red. Blue is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.10 Graphics.clBtnFace Constant

Represents the color of the face of a button.

Pascal

```
clBtnFace = TColor(clSystemColor or COLOR_BTNFACE);
```

C++

```
clBtnFace = TColor(clSystemColor or COLOR_BTNFACE);
```

File

Graphics

Description

clBtnFace is the color used to paint the face of a button.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.11 Graphics.clBtnHighlight Constant

Represents the color of the highlighting on a button.

Pascal

```
clBtnHighlight = TColor(clSystemColor or COLOR_BTNHIGHLIGHT);
```

C++

```
clBtnHighlight = TColor(clSystemColor or COLOR_BTNHIGHLIGHT);
```

File

Graphics

Description

clBtnHighlight is the color used to paint the highlighting on a button.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.12 Graphics.clBtnShadow Constant

Represents the color of the shadow cast by of a button.

Pascal

```
clBtnShadow = TColor(clSystemColor or COLOR_BTNSHADOW);
```

C++

```
clBtnShadow = TColor(clSystemColor or COLOR_BTNSHADOW);
```

File

Graphics

Description

clBtnShadow is the color used to paint the shadow cast by a button.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.13 Graphics.clBtnText Constant

Represents the default color of the text on a button.

Pascal

```
clBtnText = TColor(clSystemColor or COLOR_BTNTEXT);
```

C++

```
clBtnText = TColor(clSystemColor or COLOR_BTNTEXT);
```

File

Graphics

Description

clBtnText is the default color used to paint the text on a button.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.14 Graphics.clCaptionText Constant

Represents the default color of the text on the title bar of an active window.

Pascal

```
clCaptionText = TColor(clSystemColor or COLOR_CAPTIONTEXT);
```

C++

```
clCaptionText = TColor(clSystemColor or COLOR_CAPTIONTEXT);
```

File

Graphics

Description

clCaptionText is the default color used for the text on the title bar of an active window. clCaptionText is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.15 Graphics.clCream Constant

Represents the color cream.

Pascal

```
clCream = TColor($F0FBFF);
```

C++

```
clCream = TColor($F0FBFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to cream. Cream is an off white color. Cream is an extended color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.16 Graphics.clDefault Constant

Specifies the default color of a component.

Pascal

```
clDefault = TColor($20000000);
```

C++

```
clDefault = TColor($20000000);
```

File

Graphics

Description

clDefault is used to determine or set the default color of a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.17 Graphics.clDkGray Constant

Represents the color dark gray.

Pascal

```
clDkGray = TColor($808080);
```

C++

```
clDkGray = TColor($808080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to dark gray. Dark gray comprises 50% intensity of each of blue, green and red - identical to clGray (see page 2905). Dark Gray is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.18 Graphics.clFuchsia Constant

Represents the color fuchsia.

Pascal

```
clFuchsia = TColor($FF00FF);
```

C++

```
clFuchsia = TColor($FF00FF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to fuchsia. Fuchsia is 100% intensity red, 100% intensity blue and 0% green. Fuchsia is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.19 Graphics.clGradientActiveCaption Constant

Right side color in the color gradient of an active window's title bar.

Pascal

```
clGradientActiveCaption = TColor(clSystemColor or COLOR_GRADIENTACTIVECAPTION);
```

C++

```
clGradientActiveCaption = TColor(clSystemColor or COLOR_GRADIENTACTIVECAPTION);
```

File

Graphics

Description

For a Windows 98 and Windows 2000 color gradient in an active caption, clActiveCaption (see page 2897) defines the left side color, and clGradientActiveCaption defines the right side color. The left side color blends to the right side color.

See Also

clActiveCaption (see page 2897)

clInactiveCaption (see page 2908)

clGradientInactiveCaption (see page 2904)

Color (see page 2669)

TColor (see page 2878)

8.7.20 Graphics.clGradientInactiveCaption Constant

Right side color in the color gradient of an inactive window's title bar.

Pascal

```
clGradientInactiveCaption = TColor(clSystemColor or COLOR_GRADIENTINACTIVECAPTION);
```

C++

```
clGradientInactiveCaption = TColor(clSystemColor or COLOR_GRADIENTINACTIVECAPTION);
```

File

Graphics

Description

For a Windows 98 and Windows 2000 color gradient in an inactive caption, clInactiveCaption (see page 2908) defines the left side color, and clGradientInactiveCaption defines the right side color. The left side color blends to the right side color.

See Also

clActiveCaption (see page 2897)

clGradientActiveCaption (see page 2904)

clInactiveCaption (see page 2908)

Color (see page 2669)

TColor (see page 2878)

8.7.21 Graphics.clGray Constant

Represents the color gray.

Pascal

```
clGray = TColor($808080);
```

C++

```
clGray = TColor($808080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to gray. Gray comprises 50% intensity of each of blue, green and red. Gray is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.22 Graphics.clGrayText Constant

Represents the color of text that is grayed out.

Pascal

```
clGrayText = TColor(clSystemColor or COLOR_GRAYTEXT);
```

C++

```
clGrayText = TColor(clSystemColor or COLOR_GRAYTEXT);
```

File

Graphics

Description

clGrayText is the color used to paint text that has been set inactive, such as in an inactive menu item. clGrayText is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.23 Graphics.clGreen Constant

Represents the color green.

Pascal

```
clGreen = TColor($008000);
```

C++

```
clGreen = TColor($008000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to green. Green is 100% green with no blue or red. Green is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.24 Graphics.clHighlight Constant

Represents the color of the background of a block of selected text.

Pascal

```
clHighlight = TColor(clSystemColor or COLOR_HIGHLIGHT);
```

C++

```
clHighlight = TColor(clSystemColor or COLOR_HIGHLIGHT);
```

File

Graphics

Description

clHighlight is the default color used to paint the background of a selected block of text. clHighlight is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.25 Graphics.clHighlightText Constant

Represents the color of selected text.

Pascal

```
clHighlightText = TColor(clSystemColor or COLOR_HIGHLIGHTTEXT);
```

C++

```
clHighlightText = TColor(clSystemColor or COLOR_HIGHLIGHTTEXT);
```

File

Graphics

Description

clHighlightText is the default color used for a selected block of text. clHighlightText is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.26 Graphics.clHotLight Constant

Pascal

```
clHotLight = TColor(clSystemColor or COLOR_HOTLIGHT);
```

C++

```
clHotLight = TColor(clSystemColor or COLOR_HOTLIGHT);
```

File

Graphics

Description

This is constant Graphics.clHotLight.

8.7.27 Graphics.clInactiveBorder Constant

Represents the default border color of an inactive window.

Pascal

```
clInactiveBorder = TColor(clSystemColor or COLOR_INACTIVEBORDER);
```

C++

```
clInactiveBorder = TColor(clSystemColor or COLOR_INACTIVEBORDER);
```

File

Graphics

Description

clInactiveBorder is the default color used to paint the border of an inactive window. clInactiveBorder is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.28 Graphics.clInactiveCaption Constant

Represents the default color of the title bar of an inactive window.

Pascal

```
clInactiveCaption = TColor(clSystemColor or COLOR_INACTIVECAPTION);
```

C++

```
clInactiveCaption = TColor(clSystemColor or COLOR_INACTIVECAPTION);
```

File

Graphics

Description

clInactiveCaption is the default color used to paint the border of an inactive window. clInactiveCaption is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.29 Graphics.clInactiveCaptionText Constant

Represents the default color of the title text in an inactive window.

Pascal

```
clInactiveCaptionText = TColor(clSystemColor or COLOR_INACTIVECAPTIONTEXT);
```

C++

```
clInactiveCaptionText = TColor(clSystemColor or COLOR_INACTIVECAPTIONTEXT);
```

File

Graphics

Description

clInactiveCaptionText is the default color used for the title font an inactive window. clInactiveCaptionText is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.30 Graphics.clInfoBk Constant

Represents the color of the background of a tool tip.

Pascal

```
clInfoBk = TColor(clSystemColor or COLOR_INFOBK);
```

C++

```
clInfoBk = TColor(clSystemColor or COLOR_INFOBK);
```

File

Graphics

Description

clInfoBk is the default color used for the background of a tool tip. clInfoBk is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.31 Graphics.clInfoText Constant

Represents the color of tool tip text.

Pascal

```
clInfoText = TColor(clSystemColor or COLOR_INFOTEXT);
```

C++

```
clInfoText = TColor(clSystemColor or COLOR_INFOTEXT);
```

File

Graphics

Description

clInfoText is the default color used for tool tip text. clInfoText is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.32 Graphics.clLime Constant

Represents the color lime.

Pascal

```
clLime = TColor($00FF00);
```

C++

```
clLime = TColor($00FF00);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to lime. Lime is 100% green with no blue or red. It is identical to clGreen (see page 2906). Lime is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.33 Graphics.clLtGray Constant

Represents the color light gray.

Pascal

```
clLtGray = TColor($C0C0C0);
```

C++

```
clLtGray = TColor($C0C0C0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to light gray. Light gray comprises 75% intensity of each of blue, green and red. It is identical to clSilver (see page 2916). Light Gray is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.34 Graphics.clMaroon Constant

Represents the color maroon.

Pascal

```
clMaroon = TColor($000080);
```

C++

```
clMaroon = TColor($000080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to maroon. Maroon gray comprises 50% intensity red, with no blue or green. Maroon is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.35 Graphics.clMedGray Constant

Represents the color medium gray.

Pascal

```
clMedGray = TColor($A4A0A0);
```

C++

```
clMedGray = TColor($A4A0A0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to medium gray. Medium gray lies between clGray (see page 2905) and clLtGray (see page 2910), but, unusually, with a blue tint. Medium Gray is an extended color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.36 Graphics.clMenu Constant

Represents the color of the background of an unselected menu sub item.

Pascal

```
clMenu = TColor(clSystemColor or COLOR_MENU);
```

C++

```
clMenu = TColor(clSystemColor or COLOR_MENU);
```

File

Graphics

Description

clMenu is the default color used for the background of a sub menu item. clMenuBar (see page 2912) defines the color used for the background of the main menu items. clMenu is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.37 Graphics.clMenuBar Constant

Represents the color of the background of an unselected main menu item.

Pascal

```
clMenuBar = TColor(clSystemColor or COLOR_MENUBAR);
```

C++

```
clMenuBar = TColor(clSystemColor or COLOR_MENUBAR);
```

File

Graphics

Description

clMenuBar is the default color used for the background of a main menu item. clMenu (see page 2912) defines the color used for the background of menu sub items. clMenuBar is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.38 Graphics.clMenuHighlight Constant

Represents the color of the background of a selected menu item.

Pascal

```
clMenuHighlight = TColor(clSystemColor or COLOR_MENUHILIGHT);
```

C++

```
clMenuHighlight = TColor(clSystemColor or COLOR_MENUHILIGHT);
```

File

Graphics

Description

clMenuHighlight is the default color used for the background of a selcetd main or sub menu item. clMenuHighlight is a System color - used for painting a component.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.39 Graphics.clMenuText Constant

Represents the default menu text color.

Pascal

```
clMenuText = TColor(clSystemColor or COLOR_MENUTEXT);
```

C++

```
clMenuText = TColor(clSystemColor or COLOR_MENUTEXT);
```

File

Graphics

Description

clMenuText is the default color for menu text.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.40 Graphics.clMoneyGreen Constant

Represents the color money green.

Pascal

```
clMoneyGreen = TColor($C0DCC0);
```

C++

```
clMoneyGreen = TColor($C0DCC0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to money green. Money green is a green tinted light gray. Money green is an extended color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.41 Graphics.clNavy Constant

Represents the color navy.

Pascal

```
clNavy = TColor($800000);
```

C++

```
clNavy = TColor($800000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to navy. Navy gray comprises 50% intensity blue, with no red or green. Navy is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.42 Graphics.clNone Constant

Represents no selected color.

Pascal

```
clNone = TColor($1FFFFFFF);
```

C++

```
clNone = TColor($1FFFFFFF);
```

File

Graphics

Description

clNone represents no selected color.

Note: clNone is white on Windows 9.x systems, and black on Windows NT systems.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.43 Graphics.clOlive Constant

Represents the color olive.

Pascal

```
clOlive = TColor($008080);
```

C++

```
clOlive = TColor($008080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to olive. Olive is 50% intensity green, 50% intensity red with no blue. Olive is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.44 Graphics.clPurple Constant

Represents the color purple.

Pascal

```
clPurple = TColor($800080);
```

C++

```
clPurple = TColor($800080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to purple. Purple is 50% intensity red, 50% intensity blue and 0% green. Purple is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.45 Graphics.clRed Constant

Represents the color red.

Pascal

```
clRed = TColor($0000FF);
```

C++

```
clRed = TColor($0000FF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to red. Red is 100% red with no green or blue. Red is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.46 Graphics.clScrollBar Constant

Pascal

```
clScrollBar = TColor(clSystemColor or COLOR_SCROLLBAR);
```

C++

```
clScrollBar = TColor(clSystemColor or COLOR_SCROLLBAR);
```

File

Graphics

Description

This is constant Graphics.clScrollBar.

8.7.47 Graphics.clSilver Constant

Represents the color silver.

Pascal

```
clSilver = TColor($C0C0C0);
```

C++

```
clSilver = TColor($C0C0C0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to silver. Silver comprises 75% intensity of each of blue, green and red. It is identical to clLtGray (see page 2910). Silver is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.48 Graphics.clSkyBlue Constant

Represents the color sky blue.

Pascal

```
clSkyBlue = TColor($F0CAA6);
```

C++

```
clSkyBlue = TColor($F0CAA6);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to sky blue. Sky blue light blue that is similar to the sky color. Sky blue is an extended color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.49 Graphics.clSystemColor Constant

Is a mask used to identify the set of system colors.

Pascal

```
clSystemColor = $FF000000;
```

C++

```
clSystemColor = 0xFF000000;
```

File

Graphics

Description

clSystemColor has the 1st byte bits set on as ameans of identifying a system color.

For example, clActiveBorder (see page 2897) is system color, used as the default color to paint the active border of a component - it has the high order byte set to $FF. clRed (see page 2916) is not a system color, since the high order byte is 0.

See Also

clActiveBorder (see page 2897)

clRed (see page 2916)

Color (see page 2669)

TColor (see page 2878)

8.7.50 Graphics.clTeal Constant

Represents the color teal.

Pascal

```
clTeal = TColor($808000);
```

C++

```
clTeal = TColor($808000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to teal. Teal is dark cyan : 50% blue, 50% green, 0% red. Teal is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.51 Graphics.clWebAliceBlue Constant

Pascal

```
clWebAliceBlue = $FFF8F0;
```

C++

```
clWebAliceBlue = 0xFFF8F0;
```

File

Graphics

Description

This is constant Graphics.clWebAliceBlue.

8.7.52 Graphics.clWebAntiqueWhite Constant

Pascal

```
clWebAntiqueWhite = $D7EBFA;
```

C++

```
clWebAntiqueWhite = 0xD7EBFA;
```

File

Graphics

Description

This is constant Graphics.clWebAntiqueWhite.

8.7.53 Graphics.clWebAqua Constant

Pascal

```
clWebAqua = $FFFF00;
```

C++

```
clWebAqua = 0xFFFF00;
```

File

Graphics

Description

This is constant Graphics.clWebAqua.

8.7.54 Graphics.clWebAquamarine Constant

Pascal

```
clWebAquamarine = $D4FF7F;
```

C++

```
clWebAquamarine = 0xD4FF7F;
```

File

Graphics

Description

This is constant Graphics.clWebAquamarine.

8.7.55 Graphics.clWebAzure Constant

Pascal

```
clWebAzure = $FFFFF0;
```

C++

```
clWebAzure = 0xFFFFF0;
```

File

Graphics

Description

This is constant Graphics.clWebAzure.

8.7.56 Graphics.clWebBeige Constant

Pascal

```
clWebBeige = $DCF5F5;
```

C++

```
clWebBeige = 0xDCF5F5;
```

File

Graphics

Description

This is constant Graphics.clWebBeige.

8.7.57 Graphics.clWebBisque Constant

Pascal

```
clWebBisque = $C4E4FF;
```

C++

```
clWebBisque = 0xC4E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebBisque.

8.7.58 Graphics.clWebBlack Constant

Pascal

```
clWebBlack = $000000;
```

C++

```
clWebBlack = 0x000000;
```

File

Graphics

Description

This is constant Graphics.clWebBlack.

8.7.59 Graphics.clWebBlanchedAlmond Constant

Pascal

```
clWebBlanchedAlmond = $CDEBFF;
```

C++

```
clWebBlanchedAlmond = 0xCDEBFF;
```

File

Graphics

Description

This is constant Graphics.clWebBlanchedAlmond.

8.7.60 Graphics.clWebBlue Constant

Pascal

```
clWebBlue = $FF0000;
```

C++

```
clWebBlue = 0xFF0000;
```

File

Graphics

Description

This is constant Graphics.clWebBlue.

8.7.61 Graphics.clWebBlueViolet Constant

Pascal

```
clWebBlueViolet = $E22B8A;
```

C++

```
clWebBlueViolet = 0xE22B8A;
```

File

Graphics

Description

This is constant Graphics.clWebBlueViolet.

8.7.62 Graphics.clWebBrown Constant

Pascal

```
clWebBrown = $2A2AA5;
```

C++

```
clWebBrown = 0x2A2AA5;
```

File

Graphics

Description

This is constant Graphics.clWebBrown.

8.7.63 Graphics.clWebBurlywood Constant

Pascal

```
clWebBurlywood = $87B8DE;
```

C++

```
clWebBurlywood = 0x87B8DE;
```

File

Graphics

Description

This is constant Graphics.clWebBurlywood.

8.7.64 Graphics.clWebCadetBlue Constant

Pascal

```
clWebCadetBlue = $A09E5F;
```

C++

```
clWebCadetBlue = 0xA09E5F;
```

File

Graphics

Description

This is constant Graphics.clWebCadetBlue.

8.7.65 Graphics.clWebChartreuse Constant

Pascal

```
clWebChartreuse = $00FF7F;
```

C++

```
clWebChartreuse = 0x00FF7F;
```

File

Graphics

Description

This is constant Graphics.clWebChartreuse.

8.7.66 Graphics.clWebChocolate Constant

Pascal

```
clWebChocolate = $1E69D2;
```

C++

```
clWebChocolate = 0x1E69D2;
```

File

Graphics

Description

This is constant Graphics.clWebChocolate.

8.7.67 Graphics.clWebCoral Constant

Pascal

```
clWebCoral = $507FFF;
```

C++

```
clWebCoral = 0x507FFF;
```

File

Graphics

Description

This is constant Graphics.clWebCoral.

8.7.68 Graphics.clWebCornFlowerBlue Constant

Pascal

```
clWebCornFlowerBlue = $ED9564;
```

C++

```
clWebCornFlowerBlue = 0xED9564;
```

File

Graphics

Description

This is constant Graphics.clWebCornFlowerBlue.

8.7.69 Graphics.clWebCornSilk Constant

Pascal

```
clWebCornSilk = $DCF8FF;
```

C++

```
clWebCornSilk = 0xDCF8FF;
```

File

Graphics

Description

This is constant Graphics.clWebCornSilk.

8.7.70 Graphics.clWebCrimson Constant

Pascal

```
clWebCrimson = $3C14DC;
```

C++

```
clWebCrimson = 0x3C14DC;
```

File

Graphics

Description

This is constant Graphics.clWebCrimson.

8.7.71 Graphics.clWebCyan Constant

Pascal

```
clWebCyan = $FFFF00;
```

C++

```
clWebCyan = 0xFFFF00;
```

File

Graphics

Description

This is constant Graphics.clWebCyan.

8.7.72 Graphics.clWebDarkBlue Constant

Pascal

```
clWebDarkBlue = $8B0000;
```

C++

```
clWebDarkBlue = 0x8B0000;
```

File

Graphics

Description

This is constant Graphics.clWebDarkBlue.

8.7.73 Graphics.clWebDarkCyan Constant

Pascal

```
clWebDarkCyan = $8B8B00;
```

C++

```
clWebDarkCyan = 0x8B8B00;
```

File

Graphics

Description

This is constant Graphics.clWebDarkCyan.

8.7.74 Graphics.clWebDarkGoldenRod Constant

Pascal

```
clWebDarkGoldenRod = $0B86B8;
```

C++

```
clWebDarkGoldenRod = 0x0B86B8;
```

File

Graphics

Description

This is constant Graphics.clWebDarkGoldenRod.

8.7.75 Graphics.clWebDarkGray Constant

Pascal

```
clWebDarkGray = $A9A9A9;
```

C++

```
clWebDarkGray = 0xA9A9A9;
```

File

Graphics

Description

This is constant Graphics.clWebDarkGray.

8.7.76 Graphics.clWebDarkgreen Constant

Pascal

```
clWebDarkgreen = $006400;
```

C++

```
clWebDarkgreen = 0x006400;
```

File

Graphics

Description

This is constant Graphics.clWebDarkgreen.

8.7.77 Graphics.clWebDarkKhaki Constant

Pascal

```
clWebDarkKhaki = $6BB7BD;
```

C++

```
clWebDarkKhaki = 0x6BB7BD;
```

File

Graphics

Description

This is constant Graphics.clWebDarkKhaki.

8.7.78 Graphics.clWebDarkMagenta Constant

Pascal

```
clWebDarkMagenta = $8B008B;
```

C++

```
clWebDarkMagenta = 0x8B008B;
```

File

Graphics

Description

This is constant Graphics.clWebDarkMagenta.

8.7.79 Graphics.clWebDarkOliveGreen Constant

Pascal

```
clWebDarkOliveGreen = $2F6B55;
```

C++

```
clWebDarkOliveGreen = 0x2F6B55;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOliveGreen.

8.7.80 Graphics.clWebDarkOrange Constant

Pascal

```
clWebDarkOrange = $008CFF;
```

C++

```
clWebDarkOrange = 0x008CFF;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOrange.

8.7.81 Graphics.clWebDarkOrchid Constant

Pascal

```
clWebDarkOrchid = $CC3299;
```

C++

```
clWebDarkOrchid = 0xCC3299;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOrchid.

8.7.82 Graphics.clWebDarkRed Constant

Pascal

```
clWebDarkRed = $00008B;
```

C++

```
clWebDarkRed = 0x00008B;
```

File

Graphics

Description

This is constant Graphics.clWebDarkRed.

8.7.83 Graphics.clWebDarkSalmon Constant

Pascal

```
clWebDarkSalmon = $7A96E9;
```

C++

```
clWebDarkSalmon = 0x7A96E9;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSalmon.

8.7.84 Graphics.clWebDarkSeaGreen Constant

Pascal

```
clWebDarkSeaGreen = $8FBC8F;
```

C++

```
clWebDarkSeaGreen = 0x8FBC8F;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSeaGreen.

8.7.85 Graphics.clWebDarkSlateBlue Constant

Pascal

```
clWebDarkSlateBlue = $8B3D48;
```

C++

```
clWebDarkSlateBlue = 0x8B3D48;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSlateBlue.

8.7.86 Graphics.clWebDarkSlategray Constant

Pascal

```
clWebDarkSlategray = $4F4F2F;
```

C++

```
clWebDarkSlategray = 0x4F4F2F;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSlategray.

8.7.87 Graphics.clWebDarkTurquoise Constant

Pascal

```
clWebDarkTurquoise = $D1CE00;
```

C++

```
clWebDarkTurquoise = 0xD1CE00;
```

File

Graphics

Description

This is constant Graphics.clWebDarkTurquoise.

8.7.88 Graphics.clWebDarkViolet Constant

Pascal

```
clWebDarkViolet = $D30094;
```

C++

```
clWebDarkViolet = 0xD30094;
```

File

Graphics

Description

This is constant Graphics.clWebDarkViolet.

8.7.89 Graphics.clWebDeepPink Constant

Pascal

```
clWebDeepPink = $9314FF;
```

C++

```
clWebDeepPink = 0x9314FF;
```

File

Graphics

Description

This is constant Graphics.clWebDeepPink.

8.7.90 Graphics.clWebDeepskyBlue Constant

Pascal

```
clWebDeepskyBlue = $FFBF00;
```

C++

```
clWebDeepskyBlue = 0xFFBF00;
```

File

Graphics

Description

This is constant Graphics.clWebDeepskyBlue.

8.7.91 Graphics.clWebDimGray Constant

Pascal

```
clWebDimGray = $696969;
```

C++

```
clWebDimGray = 0x696969;
```

File

Graphics

Description

This is constant Graphics.clWebDimGray.

8.7.92 Graphics.clWebDodgerBlue Constant

Pascal

```
clWebDodgerBlue = $FF901E;
```

C++

```
clWebDodgerBlue = 0xFF901E;
```

File

Graphics

Description

This is constant Graphics.clWebDodgerBlue.

8.7.93 Graphics.clWebFirebrick Constant

Pascal

```
clWebFirebrick = $2222B2;
```

C++

```
clWebFirebrick = 0x2222B2;
```

File

Graphics

Description

This is constant Graphics.clWebFirebrick.

8.7.94 Graphics.clWebFloralWhite Constant

Pascal

```
clWebFloralWhite = $F0FAFF;
```

C++

```
clWebFloralWhite = 0xF0FAFF;
```

File

Graphics

Description

This is constant Graphics.clWebFloralWhite.

8.7.95 Graphics.clWebForestGreen Constant

Pascal

```
clWebForestGreen = $228B22;
```

C++

```
clWebForestGreen = 0x228B22;
```

File

Graphics

Description

This is constant Graphics.clWebForestGreen.

8.7.96 Graphics.clWebFuchsia Constant

Pascal

```
clWebFuchsia = $FF00FF;
```

C++

```
clWebFuchsia = 0xFF00FF;
```

File

Graphics

Description

This is constant Graphics.clWebFuchsia.

8.7.97 Graphics.clWebGainsboro Constant

Pascal

```
clWebGainsboro = $DCDCDC;
```

C++

```
clWebGainsboro = 0xDCDCDC;
```

File

Graphics

Description

This is constant Graphics.clWebGainsboro.

8.7.98 Graphics.clWebGhostWhite Constant

Pascal

```
clWebGhostWhite = $FFF8F8;
```

C++

```
clWebGhostWhite = 0xFFF8F8;
```

File

Graphics

Description

This is constant Graphics.clWebGhostWhite.

8.7.99 Graphics.clWebGold Constant

Pascal

```
clWebGold = $00D7FF;
```

C++

```
clWebGold = 0x00D7FF;
```

File

Graphics

Description

This is constant Graphics.clWebGold.

8.7.100 Graphics.clWebGoldenRod Constant

Pascal

```
clWebGoldenRod = $20A5DA;
```

C++

```
clWebGoldenRod = 0x20A5DA;
```

File

Graphics

Description

This is constant Graphics.clWebGoldenRod.

8.7.101 Graphics.clWebGray Constant

Pascal

```
clWebGray = $808080;
```

C++

```
clWebGray = 0x808080;
```

File

Graphics

Description

This is constant Graphics.clWebGray.

8.7.102 Graphics.clWebGreen Constant

Pascal

```
clWebGreen = $008000;
```

C++

```
clWebGreen = 0x008000;
```

File

Graphics

Description

This is constant Graphics.clWebGreen.

8.7.103 Graphics.clWebGreenYellow Constant

Pascal

```
clWebGreenYellow = $2FFFAD;
```

C++

```
clWebGreenYellow = 0x2FFFAD;
```

File

Graphics

Description

This is constant Graphics.clWebGreenYellow.

8.7.104 Graphics.clWebHoneydew Constant

Pascal

```
clWebHoneydew = $F0FFF0;
```

C++

```
clWebHoneydew = 0xF0FFF0;
```

File

Graphics

Description

This is constant Graphics.clWebHoneydew.

8.7.105 Graphics.clWebHotPink Constant

Pascal

```
clWebHotPink = $B469FF;
```

C++

```
clWebHotPink = 0xB469FF;
```

File

Graphics

Description

This is constant Graphics.clWebHotPink.

8.7.106 Graphics.clWebIndianRed Constant

Pascal

```
clWebIndianRed = $5C5CCD;
```

C++

```
clWebIndianRed = 0x5C5CCD;
```

File

Graphics

Description

This is constant Graphics.clWebIndianRed.

8.7.107 Graphics.clWebIndigo Constant

Pascal

```
clWebIndigo = $82004B;
```

C++

```
clWebIndigo = 0x82004B;
```

File

Graphics

Description

This is constant Graphics.clWebIndigo.

8.7.108 Graphics.clWebIvory Constant

Pascal

```
clWebIvory = $F0FFFF;
```

C++

```
clWebIvory = 0xF0FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebIvory.

8.7.109 Graphics.clWebKhaki Constant

Pascal

```
clWebKhaki = $8CE6F0;
```

C++

```
clWebKhaki = 0x8CE6F0;
```

File

Graphics

Description

This is constant Graphics.clWebKhaki.

8.7.110 Graphics.clWebLavender Constant

Pascal

```
clWebLavender = $FAE6E6;
```

C++

```
clWebLavender = 0xFAE6E6;
```

File

Graphics

Description

This is constant Graphics.clWebLavender.

8.7.111 Graphics.clWebLavenderBlush Constant

Pascal

```
clWebLavenderBlush = $F5F0FF;
```

C++

```
clWebLavenderBlush = 0xF5F0FF;
```

File

Graphics

Description

This is constant Graphics.clWebLavenderBlush.

8.7.112 Graphics.clWebLawnGreen Constant

Pascal

```
clWebLawnGreen = $00FC7C;
```

C++

```
clWebLawnGreen = 0x00FC7C;
```

File

Graphics

Description

This is constant Graphics.clWebLawnGreen.

8.7.113 Graphics.clWebLemonChiffon Constant

Pascal

```
clWebLemonChiffon = $CDFAFF;
```

C++

```
clWebLemonChiffon = 0xCDFAFF;
```

File

Graphics

Description

This is constant Graphics.clWebLemonChiffon.

8.7.114 Graphics.clWebLightBlue Constant

Pascal

```
clWebLightBlue = $E6D8AD;
```

C++

```
clWebLightBlue = 0xE6D8AD;
```

File

Graphics

Description

This is constant Graphics.clWebLightBlue.

8.7.115 Graphics.clWebLightCoral Constant

Pascal

```
clWebLightCoral = $8080F0;
```

C++

```
clWebLightCoral = 0x8080F0;
```

File

Graphics

Description

This is constant Graphics.clWebLightCoral.

8.7.116 Graphics.clWebLightCyan Constant

Pascal

```
clWebLightCyan = $FFFFE0;
```

C++

```
clWebLightCyan = 0xFFFFE0;
```

File

Graphics

Description

This is constant Graphics.clWebLightCyan.

8.7.117 Graphics.clWebLightGoldenrodYellow Constant

Pascal

```
clWebLightGoldenrodYellow = $D2FAFA;
```

C++

```
clWebLightGoldenrodYellow = 0xD2FAFA;
```

File

Graphics

Description

This is constant Graphics.clWebLightGoldenrodYellow.

8.7.118 Graphics.clWebLightGreen Constant

Pascal

```
clWebLightGreen = $90EE90;
```

C++

```
clWebLightGreen = 0x90EE90;
```

File

Graphics

Description

This is constant Graphics.clWebLightGreen.

8.7.119 Graphics.clWebLightgrey Constant

Pascal

```
clWebLightgrey = $D3D3D3;
```

C++

```
clWebLightgrey = 0xD3D3D3;
```

File

Graphics

Description

This is constant Graphics.clWebLightgrey.

8.7.120 Graphics.clWebLightPink Constant

Pascal

```
clWebLightPink = $C1B6FF;
```

C++

```
clWebLightPink = 0xC1B6FF;
```

File

Graphics

Description

This is constant Graphics.clWebLightPink.

8.7.121 Graphics.clWebLightSalmon Constant

Pascal

```
clWebLightSalmon = $7AA0FF;
```

C++

```
clWebLightSalmon = 0x7AA0FF;
```

File

Graphics

Description

This is constant Graphics.clWebLightSalmon.

8.7.122 Graphics.clWebLightSeaGreen Constant

Pascal

```
clWebLightSeaGreen = $AAB220;
```

C++

```
clWebLightSeaGreen = 0xAAB220;
```

File

Graphics

Description

This is constant Graphics.clWebLightSeaGreen.

8.7.123 Graphics.clWebLightSkyBlue Constant

Pascal

```
clWebLightSkyBlue = $FACE87;
```

C++

```
clWebLightSkyBlue = 0xFACE87;
```

File

Graphics

Description

This is constant Graphics.clWebLightSkyBlue.

8.7.124 Graphics.clWebLightSlateGray Constant

Pascal

```
clWebLightSlateGray = $998877;
```

C++

```
clWebLightSlateGray = 0x998877;
```

File

Graphics

Description

This is constant Graphics.clWebLightSlateGray.

8.7.125 Graphics.clWebLightSteelBlue Constant

Pascal

```
clWebLightSteelBlue = $DEC4B0;
```

C++

```
clWebLightSteelBlue = 0xDEC4B0;
```

File

Graphics

Description

This is constant Graphics.clWebLightSteelBlue.

8.7.126 Graphics.clWebLightYellow Constant

Pascal

```
clWebLightYellow = $E0FFFF;
```

C++

```
clWebLightYellow = 0xE0FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebLightYellow.

8.7.127 Graphics.clWebLime Constant

Pascal

```
clWebLime = $00FF00;
```

C++

```
clWebLime = 0x00FF00;
```

File

Graphics

Description

This is constant Graphics.clWebLime.

8.7.128 Graphics.clWebLimeGreen Constant

Pascal

```
clWebLimeGreen = $32CD32;
```

C++

```
clWebLimeGreen = 0x32CD32;
```

File

Graphics

Description

This is constant Graphics.clWebLimeGreen.

8.7.129 Graphics.clWebLinen Constant

Pascal

```
clWebLinen = $E6F0FA;
```

C++

```
clWebLinen = 0xE6F0FA;
```

File

Graphics

Description

This is constant Graphics.clWebLinen.

8.7.130 Graphics.clWebMagenta Constant

Pascal

```
clWebMagenta = $FF00FF;
```

C++

```
clWebMagenta = 0xFF00FF;
```

File

Graphics

Description

This is constant Graphics.clWebMagenta.

8.7.131 Graphics.clWebMaroon Constant

Pascal

```
clWebMaroon = $000080;
```

C++

```
clWebMaroon = 0x000080;
```

File

Graphics

Description

This is constant Graphics.clWebMaroon.

8.7.132 Graphics.clWebMediumAquamarine Constant

Pascal

```
clWebMediumAquamarine = $AACD66;
```

C++

```
clWebMediumAquamarine = 0xAACD66;
```

File

Graphics

Description

This is constant Graphics.clWebMediumAquamarine.

8.7.133 Graphics.clWebMediumBlue Constant

Pascal

```
clWebMediumBlue = $CD0000;
```

C++

```
clWebMediumBlue = 0xCD0000;
```

File

Graphics

Description

This is constant Graphics.clWebMediumBlue.

8.7.134 Graphics.clWebMediumOrchid Constant

Pascal

```
clWebMediumOrchid = $D355BA;
```

C++

```
clWebMediumOrchid = 0xD355BA;
```

File

Graphics

Description

This is constant Graphics.clWebMediumOrchid.

8.7.135 Graphics.clWebMediumPurple Constant

Pascal

```
clWebMediumPurple = $DB7093;
```

C++

```
clWebMediumPurple = 0xDB7093;
```

File

Graphics

Description

This is constant Graphics.clWebMediumPurple.

8.7.136 Graphics.clWebMediumSeaGreen Constant

Pascal

```
clWebMediumSeaGreen = $71B33C;
```

C++

```
clWebMediumSeaGreen = 0x71B33C;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSeaGreen.

8.7.137 Graphics.clWebMediumSlateBlue Constant

Pascal

```
clWebMediumSlateBlue = $EE687B;
```

C++

```
clWebMediumSlateBlue = 0xEE687B;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSlateBlue.

8.7.138 Graphics.clWebMediumSpringGreen Constant

Pascal

```
clWebMediumSpringGreen = $9AFA00;
```

C++

```
clWebMediumSpringGreen = 0x9AFA00;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSpringGreen.

8.7.139 Graphics.clWebMediumTurquoise Constant

Pascal

```
clWebMediumTurquoise = $CCD148;
```

C++

```
clWebMediumTurquoise = 0xCCD148;
```

File

Graphics

Description

This is constant Graphics.clWebMediumTurquoise.

8.7.140 Graphics.clWebMediumVioletRed Constant

Pascal

```
clWebMediumVioletRed = $8515C7;
```

C++

```
clWebMediumVioletRed = 0x8515C7;
```

File

Graphics

Description

This is constant Graphics.clWebMediumVioletRed.

8.7.141 Graphics.clWebMidnightBlue Constant

Pascal

```
clWebMidnightBlue = $701919;
```

C++

```
clWebMidnightBlue = 0x701919;
```

File

Graphics

Description

This is constant Graphics.clWebMidnightBlue.

8.7.142 Graphics.clWebMintcream Constant

Pascal

```
clWebMintcream = $FAFFF5;
```

C++

```
clWebMintcream = 0xFAFFF5;
```

File

Graphics

Description

This is constant Graphics.clWebMintcream.

8.7.143 Graphics.clWebMistyRose Constant

Pascal

```
clWebMistyRose = $E1E4FF;
```

C++

```
clWebMistyRose = 0xE1E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebMistyRose.

8.7.144 Graphics.clWebMoccasin Constant

Pascal

```
clWebMoccasin = $B5E4FF;
```

C++

```
clWebMoccasin = 0xB5E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebMoccasin.

8.7.145 Graphics.clWebNavajoWhite Constant

Pascal

```
clWebNavajoWhite = $ADDEFF;
```

C++

```
clWebNavajoWhite = 0xADDEFF;
```

File

Graphics

Description

This is constant Graphics.clWebNavajoWhite.

8.7.146 Graphics.clWebNavy Constant

Pascal

```
clWebNavy = $800000;
```

C++

```
clWebNavy = 0x800000;
```

File

Graphics

Description

This is constant Graphics.clWebNavy.

8.7.147 Graphics.clWebOldLace Constant

Pascal

```
clWebOldLace = $E6F5FD;
```

C++

```
clWebOldLace = 0xE6F5FD;
```

File

Graphics

Description

This is constant Graphics.clWebOldLace.

8.7.148 Graphics.clWebOlive Constant

Pascal

```
clWebOlive = $008080;
```

C++

```
clWebOlive = 0x008080;
```

File

Graphics

Description

This is constant Graphics.clWebOlive.

8.7.149 Graphics.clWebOliveDrab Constant

Pascal

```
clWebOliveDrab = $238E6B;
```

C++

```
clWebOliveDrab = 0x238E6B;
```

File

Graphics

Description

This is constant Graphics.clWebOliveDrab.

8.7.150 Graphics.clWebOrange Constant

Pascal

```
clWebOrange = $00A5FF;
```

C++

```
clWebOrange = 0x00A5FF;
```

File

Graphics

Description

This is constant Graphics.clWebOrange.

8.7.151 Graphics.clWebOrangeRed Constant

Pascal

```
clWebOrangeRed = $0045FF;
```

C++

```
clWebOrangeRed = 0x0045FF;
```

File

Graphics

Description

This is constant Graphics.clWebOrangeRed.

8.7.152 Graphics.clWebOrchid Constant

Pascal

```
clWebOrchid = $D670DA;
```

C++

```
clWebOrchid = 0xD670DA;
```

File

Graphics

Description

This is constant Graphics.clWebOrchid.

8.7.153 Graphics.clWebPaleGoldenrod Constant

Pascal

```
clWebPaleGoldenrod = $AAE8EE;
```

C++

```
clWebPaleGoldenrod = 0xAAE8EE;
```

File

Graphics

Description

This is constant Graphics.clWebPaleGoldenrod.

8.7.154 Graphics.clWebPaleGreen Constant

Pascal

```
clWebPaleGreen = $98FB98;
```

C++

```
clWebPaleGreen = 0x98FB98;
```

File

Graphics

Description

This is constant Graphics.clWebPaleGreen.

8.7.155 Graphics.clWebPaleTurquoise Constant

Pascal

```
clWebPaleTurquoise = $EEEEAF;
```

C++

```
clWebPaleTurquoise = 0xEEEEAF;
```

File

Graphics

Description

This is constant Graphics.clWebPaleTurquoise.

8.7.156 Graphics.clWebPaleVioletRed Constant

Pascal

```
clWebPaleVioletRed = $9370DB;
```

C++

```
clWebPaleVioletRed = 0x9370DB;
```

File

Graphics

Description

This is constant Graphics.clWebPaleVioletRed.

8.7.157 Graphics.clWebPapayaWhip Constant

Pascal

```
clWebPapayaWhip = $D5EFFF;
```

C++

```
clWebPapayaWhip = 0xD5EFFF;
```

File

Graphics

Description

This is constant Graphics.clWebPapayaWhip.

8.7.158 Graphics.clWebPeachPuff Constant

Pascal

```
clWebPeachPuff = $B9DAFF;
```

C++

```
clWebPeachPuff = 0xB9DAFF;
```

File

Graphics

Description

This is constant Graphics.clWebPeachPuff.

8.7.159 Graphics.clWebPeru Constant

Pascal

```
clWebPeru = $3F85CD;
```

C++

```
clWebPeru = 0x3F85CD;
```

File

Graphics

Description

This is constant Graphics.clWebPeru.

8.7.160 Graphics.clWebPink Constant

Pascal

```
clWebPink = $CBC0FF;
```

C++

```
clWebPink = 0xCBC0FF;
```

File

Graphics

Description

This is constant Graphics.clWebPink.

8.7.161 Graphics.clWebPlum Constant

Pascal

```
clWebPlum = $DDA0DD;
```

C++

```
clWebPlum = 0xDDA0DD;
```

File

Graphics

Description

This is constant Graphics.clWebPlum.

8.7.162 Graphics.clWebPowderBlue Constant

Pascal

```
clWebPowderBlue = $E6E0B0;
```

C++

```
clWebPowderBlue = 0xE6E0B0;
```

File

Graphics

Description

This is constant Graphics.clWebPowderBlue.

8.7.163 Graphics.clWebPurple Constant

Pascal

```
clWebPurple = $800080;
```

C++

```
clWebPurple = 0x800080;
```

File

Graphics

Description

This is constant Graphics.clWebPurple.

8.7.164 Graphics.clWebRed Constant

Pascal

```
clWebRed = $0000FF;
```

C++

```
clWebRed = 0x0000FF;
```

File

Graphics

Description

This is constant Graphics.clWebRed.

8.7.165 Graphics.clWebRosyBrown Constant

Pascal

```
clWebRosyBrown = $8F8FBC;
```

C++

```
clWebRosyBrown = 0x8F8FBC;
```

File

Graphics

Description

This is constant Graphics.clWebRosyBrown.

8.7.166 Graphics.clWebRoyalBlue Constant

Pascal

```
clWebRoyalBlue = $E16941;
```

C++

```
clWebRoyalBlue = 0xE16941;
```

File

Graphics

Description

This is constant Graphics.clWebRoyalBlue.

8.7.167 Graphics.clWebSaddleBrown Constant

Pascal

```
clWebSaddleBrown = $13458B;
```

C++

```
clWebSaddleBrown = 0x13458B;
```

File

Graphics

Description

This is constant Graphics.clWebSaddleBrown.

8.7.168 Graphics.clWebSalmon Constant

Pascal

```
clWebSalmon = $7280FA;
```

C++

```
clWebSalmon = 0x7280FA;
```

File

Graphics

Description

This is constant Graphics.clWebSalmon.

8.7.169 Graphics.clWebSandyBrown Constant

Pascal

```
clWebSandyBrown = $60A4F4;
```

C++

```
clWebSandyBrown = 0x60A4F4;
```

File

Graphics

Description

This is constant Graphics.clWebSandyBrown.

8.7.170 Graphics.clWebSeaGreen Constant

Pascal

```
clWebSeaGreen = $578B2E;
```

C++

```
clWebSeaGreen = 0x578B2E;
```

File

Graphics

Description

This is constant Graphics.clWebSeaGreen.

8.7.171 Graphics.clWebSeashell Constant

Pascal

```
clWebSeashell = $EEF5FF;
```

C++

```
clWebSeashell = 0xEEF5FF;
```

File

Graphics

Description

This is constant Graphics.clWebSeashell.

8.7.172 Graphics.clWebSienna Constant

Pascal

```
clWebSienna = $2D52A0;
```

C++

```
clWebSienna = 0x2D52A0;
```

File

Graphics

Description

This is constant Graphics.clWebSienna.

8.7.173 Graphics.clWebSilver Constant

Pascal

```
clWebSilver = $C0C0C0;
```

C++

```
clWebSilver = 0xC0C0C0;
```

File

Graphics

Description

This is constant Graphics.clWebSilver.

8.7.174 Graphics.clWebSkyBlue Constant

Pascal

```
clWebSkyBlue = $EBCE87;
```

C++

```
clWebSkyBlue = 0xEBCE87;
```

File

Graphics

Description

This is constant Graphics.clWebSkyBlue.

8.7.175 Graphics.clWebSlateBlue Constant

Pascal

```
clWebSlateBlue = $CD5A6A;
```

C++

```
clWebSlateBlue = 0xCD5A6A;
```

File

Graphics

Description

This is constant Graphics.clWebSlateBlue.

8.7.176 Graphics.clWebSlateGray Constant

Pascal

```
clWebSlateGray = $908070;
```

C++

```
clWebSlateGray = 0x908070;
```

File

Graphics

Description

This is constant Graphics.clWebSlateGray.

8.7.177 Graphics.clWebSnow Constant

Pascal

```
clWebSnow = $FAFAFF;
```

C++

```
clWebSnow = 0xFAFAFF;
```

File

Graphics

Description

This is constant Graphics.clWebSnow.

8.7.178 Graphics.clWebSpringGreen Constant

Pascal

```
clWebSpringGreen = $7FFF00;
```

C++

```
clWebSpringGreen = 0x7FFF00;
```

File

Graphics

Description

This is constant Graphics.clWebSpringGreen.

8.7.179 Graphics.clWebSteelBlue Constant

Pascal

```
clWebSteelBlue = $B48246;
```

C++

```
clWebSteelBlue = 0xB48246;
```

File

Graphics

Description

This is constant Graphics.clWebSteelBlue.

8.7.180 Graphics.clWebTan Constant

Pascal

```
clWebTan = $8CB4D2;
```

C++

```
clWebTan = 0x8CB4D2;
```

File

Graphics

Description

This is constant Graphics.clWebTan.

8.7.181 Graphics.clWebTeal Constant

Pascal

```
clWebTeal = $808000;
```

C++

```
clWebTeal = 0x808000;
```

File

Graphics

Description

This is constant Graphics.clWebTeal.

8.7.182 Graphics.clWebThistle Constant

Pascal

```
clWebThistle = $D8BFD8;
```

C++

```
clWebThistle = 0xD8BFD8;
```

File

Graphics

Description

This is constant Graphics.clWebThistle.

8.7.183 Graphics.clWebTomato Constant

Pascal

```
clWebTomato = $4763FF;
```

C++

```
clWebTomato = 0x4763FF;
```

File

Graphics

Description

This is constant Graphics.clWebTomato.

8.7.184 Graphics.clWebTurquoise Constant

Pascal

```
clWebTurquoise = $D0E040;
```

C++

```
clWebTurquoise = 0xD0E040;
```

File

Graphics

Description

This is constant Graphics.clWebTurquoise.

8.7.185 Graphics.clWebViolet Constant

Pascal

```
clWebViolet = $EE82EE;
```

C++

```
clWebViolet = 0xEE82EE;
```

File

Graphics

Description

This is constant Graphics.clWebViolet.

8.7.186 Graphics.clWebWheat Constant

Pascal

```
clWebWheat = $B3DEF5;
```

C++

```
clWebWheat = 0xB3DEF5;
```

File

Graphics

Description

This is constant Graphics.clWebWheat.

8.7.187 Graphics.clWebWhite Constant

Pascal

```
clWebWhite = $FFFFFF;
```

C++

```
clWebWhite = 0xFFFFFF;
```

File

Graphics

Description

This is constant Graphics.clWebWhite.

8.7.188 Graphics.clWebWhiteSmoke Constant

Pascal

```
clWebWhiteSmoke = $F5F5F5;
```

C++

```
clWebWhiteSmoke = 0xF5F5F5;
```

File

Graphics

Description

This is constant Graphics.clWebWhiteSmoke.

8.7.189 Graphics.clWebYellow Constant

Pascal

```
clWebYellow = $00FFFF;
```

C++

```
clWebYellow = 0x00FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebYellow.

8.7.190 Graphics.clWebYellowGreen Constant

Pascal

```
clWebYellowGreen = $32CD9A;
```

C++

```
clWebYellowGreen = 0x32CD9A;
```

File

Graphics

Description

This is constant Graphics.clWebYellowGreen.

8.7.191 Graphics.clWhite Constant

Represents the color white.

Pascal

```
clWhite = TColor($FFFFFF);
```

C++

```
clWhite = TColor($FFFFFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to white. White is 100% intensity blue, green and red. White is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.192 Graphics.clWindow Constant

Represents the default Window background color.

Pascal

```
clWindow = TColor(clSystemColor or COLOR_WINDOW);
```

C++

```
clWindow = TColor(clSystemColor or COLOR_WINDOW);
```

File

Graphics

Description

clWindow is the default background color for a window.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.193 Graphics.clWindowFrame Constant

Represents the default Window frame background color.

Pascal

```
clWindowFrame = TColor(clSystemColor or COLOR_WINDOWFRAME);
```

C++

```
clWindowFrame = TColor(clSystemColor or COLOR_WINDOWFRAME);
```

File

Graphics

Description

clWindowFrame is the default background color for a window frame.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.194 Graphics.clWindowText Constant

Represents the default Window text color.

Pascal

```
clWindowText = TColor(clSystemColor or COLOR_WINDOWTEXT);
```

C++

```
clWindowText = TColor(clSystemColor or COLOR_WINDOWTEXT);
```

File

Graphics

Description

clWindowText is the default color for window text.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.195 Graphics.clYellow Constant

Represents the color yellow.

Pascal

```
clYellow = TColor($00FFFF);
```

C++

```
clYellow = TColor($00FFFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to yellow. Yellow is 100% green, 100% red and 0% blue. Yellow is a standard color.

See Also

Color (see page 2669)

TColor (see page 2878)

8.7.196 Graphics.cmBlackness Constant

TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with black.

Pascal

```
cmBlackness = BLACKNESS;
```

C++

```
cmBlackness = BLACKNESS;
```

File

Graphics

Description

cmBlackness is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmBlackness results in the destination rectangle on the canvas simply being filled with the clBlack (see page 2899) color.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.197 Graphics.cmDstInvert Constant

TCopyMode (see page 2880) value : inverts the image on the canvas and ignores the source.

Pascal

```
cmDstInvert = DSTINVERT;
```

C++

```
cmDstInvert = DSTINVERT;
```

File

Graphics

Description

cmDstInvert is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmDstInvert results in the the bits of each pixel in destination rectangle on the canvas simply being inverted. The source rectangle is ignored.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.198 Graphics.cmMergeCopy Constant

TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.

Pascal

```
cmMergeCopy = MERGECOPY;
```

C++

```
cmMergeCopy = MERGECOPY;
```

File

Graphics

Description

cmMergeCopy is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmMergeCopy results in the pixels in the source and destination rectangles being logically ANDed. For example, a green pixel in the source will merge with a blue destination pixel to result in a cyan pixel.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.199 Graphics.cmMergePaint Constant

TCopyMode (see page 2880) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.

Pascal

```
cmMergePaint = MERGEPAINT;
```

C++

```
cmMergePaint = MERGEPAINT;
```

File

Graphics

Description

cmMergePaint is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmMergePaint, the source pixels are inverted before being logically ORed with the destination pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.200 Graphics.cmNotSrcCopy Constant

TCopyMode (see page 2880) value : copies the inverted source bitmap to the canvas.

Pascal

```
cmNotSrcCopy = NOTSRCCOPY;
```

C++

```
cmNotSrcCopy = NOTSRCCOPY;
```

File

Graphics

Description

cmNotSrcCopy is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmNotSrcCopy, the source pixels are inverted before being written to the target rectangle. The existing target rectangle contents are ignored and overwritten.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.201 Graphics.cmNotSrcErase Constant

TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.

Pascal

```
cmNotSrcErase = NOTSRCERASE;
```

C++

```
cmNotSrcErase = NOTSRCERASE;
```

File

Graphics

Description

cmNotSrcErase is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmNotSrcErase, the source and destination pixels are ORed together and the result inverted and stored in the target rectangle.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.202 Graphics.cmPatCopy Constant

TCopyMode (see page 2880) value : copies the source pattern to the canvas.

Pascal

```
cmPatCopy = PATCOPY;
```

C++

```
cmPatCopy = PATCOPY;
```

File

Graphics

Description

cmPatCopy is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmPatCopy, the source pattern is written to the target rectangle.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.203 Graphics.cmPatInvert Constant

TCopyMode (see page 2880) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.

Pascal

```
cmPatInvert = PATINVERT;
```

C++

```
cmPatInvert = PATINVERT;
```

File

Graphics

Description

cmPatInvert is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmPatInvert, XORs each source pattern pixel with the target image pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.204 Graphics.cmPatPaint Constant

TCopyMode (see page 2880) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.

Pascal

```
cmPatPaint = PATPAINT;
```

C++

```
cmPatPaint = PATPAINT;
```

File

Graphics

Description

cmPatPaint is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmPatPaint, the source pattern and inverted bitmap pixels are ORed before ORing the resulting pixels with the target image pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.205 Graphics.cmSrcAnd Constant

TCopyMode (see page 2880) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.

Pascal

```
cmSrcAnd = SRCAND;
```

C++

```
cmSrcAnd = SRCAND;
```

File

Graphics

Description

cmSrcAnd is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcAnd, the source pixels are ANDed with the target target pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.206 Graphics.cmSrcCopy Constant

TCopyMode (see page 2880) value : copies the source bitmap to the canvas.

Pascal

```
cmSrcCopy = SRCCOPY;
```

C++

```
cmSrcCopy = SRCCOPY;
```

File

Graphics

Description

cmSrcCopy is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcCopy, the source pixels are written unchanged to the target rectangle.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.207 Graphics.cmSrcErase Constant

TCopyMode (see page 2880) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.

Pascal

```
cmSrcErase = SRCERASE;
```

C++

```
cmSrcErase = SRCERASE;
```

File

Graphics

Description

cmSrcErase is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcErase, the source pixels are ANDed with the inverted target pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.208 Graphics.cmSrcInvert Constant

TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.

Pascal

```
cmSrcInvert = SRCINVERT;
```

C++

```
cmSrcInvert = SRCINVERT;
```

File

Graphics

Description

cmSrcInvert is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcInvert, the source pixels are XORed with the target pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.209 Graphics.cmSrcPaint Constant

TCopyMode (see page 2880) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.

Pascal

```
cmSrcPaint = SRCPAINT;
```

C++

```
cmSrcPaint = SRCPAINT;
```

File

Graphics

Description

cmSrcPaint is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcPaint, the source pixels are ORed with the target pixels.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.210 Graphics.cmWhiteness Constant

TCopyMode (see page 2880) value : fills the destination rectangle on the canvas with white.

Pascal

```
cmWhiteness = WHITENESS;
```

C++

```
cmWhiteness = WHITENESS;
```

File

Graphics

Description

cmWhiteness is a TCopyMode (see page 2880) value used in the TCanvas (see page 2673) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmWhiteness results in the destination rectangle on the canvas simply being filled with the clWhite (see page 2965) color.

See Also

CopyMode (see page 2719)

CopyRect (see page 2683)

TCopyMode (see page 2880)

8.7.211 Graphics.ExtendedColorsCount Constant

The total number of extended colors available

Pascal

```
ExtendedColorsCount = 4;
```

C++

```
ExtendedColorsCount = 4;
```

File

Graphics

Description

TColor (see page 2878) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Extended colors are :

Name	Meaning
clMoneyGreen (see page 2913)	Money Green.
clSkyBlue (see page 2917)	Sky blue.
clCream (see page 2902)	Cream.
clMedGray (see page 2911)	Medium Gray.

See Also

Color (see page 2669)

StandardColorsCount (see page 2977)

WebColorsCount (see page 2978)

TColor (see page 2878)

8.7.212 Graphics.rc3_Cursor Constant

Pascal

```
rc3_Cursor = 2;
```

C++

```
rc3_Cursor = 2;
```

File

Graphics

Description

This is constant Graphics.rc3_Cursor.

8.7.213 Graphics.rc3_Icon Constant

Pascal

```
rc3_Icon = 1;
```

C++

```
rc3_Icon = 1;
```

File

Graphics

Description

This is constant Graphics.rc3_Icon.

8.7.214 Graphics.rc3_StockIcon Constant

Pascal

```
rc3_StockIcon = 0;
```

C++

```
rc3_StockIcon = 0;
```

File

Graphics

Description

This is constant Graphics.rc3_StockIcon.

8.7.215 Graphics.StandardColorsCount Constant

The total number of standard colors available

Pascal

```
StandardColorsCount = 16;
```

C++

```
StandardColorsCount = 16;
```

File

Graphics

Description

TColor (see page 2878) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Extended colors include clCream (see page 2902). The standard colors are :

Name	Meaning
clBlack (see page 2899)	Black.
clMaroon (see page 2911)	Maroon.
clOlive (see page 2915)	Olive.
clNavy (see page 2914)	Navy.
clPurple (see page 2915)	Purple.
clTeal (see page 2918)	Teal.
clGray (see page 2905)	Gray.
clSilver (see page 2916)	Silver.
clRed (see page 2916)	Red.
clLime (see page 2910)	Lime.
clYellow (see page 2967)	Yellow.
clBlue (see page 2899)	Blue.
clFuchsia (see page 2903)	Fuchsia.
clAqua (see page 2898)	Aqua (cyan).
clLtGray (see page 2910)	Light Gray.
clDkGray (see page 2903)	Dark Gray.
clWhite (see page 2965)	White.

See Also

Color (see page 2669)

ExtendedColorsCount (see page 2975)

WebColorsCount (see page 2978)

TColor (see page 2878)

8.7.216 Graphics.WebColorsCount Constant

The total number of web colors available

Pascal

```
WebColorsCount = 140;
```

C++

```
WebColorsCount = 140;
```

File

Graphics

Description

TColor (see page 2878) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Web colors are those defined as uniformly displayable across the Internet - they represent the Internet standard colors.

See Also

Color (see page 2669)

ExtendedColorsCount (see page 2975)

StandardColorsCount (see page 2977)

TColor (see page 2878)

9 GraphUtil Namespace

This is namespace GraphUtil.

Topics

Name	Description
Functions (see page 2981)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 2989)	The following table lists structs, records, enums in this documentation.
Types (see page 2992)	The following table lists types in this documentation.
Constants (see page 2993)	The following table lists constants in this documentation.

Constants

Name	Description
WebNamedColors (see page 2993)	This is constant GraphUtil.WebNamedColors.
WebNamedColorsCount (see page 2995)	This is constant GraphUtil.WebNamedColorsCount.

Functions

	Name	Description
	ColorAdjustLuma (see page 2981)	This is function GraphUtil.ColorAdjustLuma.
	ColorHLSToRGB (see page 2982)	This is function GraphUtil.ColorHLSToRGB.
	ColorRGBToHLS (see page 2982)	This is function GraphUtil.ColorRGBToHLS.
	ColorToWebColorName (see page 2982)	This is function GraphUtil.ColorToWebColorName.
	ColorToWebColorStr (see page 2983)	This is function GraphUtil.ColorToWebColorStr.
	DrawArrow (see page 2983)	This is function GraphUtil.DrawArrow.
	DrawCheck (see page 2983)	This is function GraphUtil.DrawCheck.
	DrawChevron (see page 2984)	This is function GraphUtil.DrawChevron.
	DrawTransparentBitmap (see page 2984)	This is function GraphUtil.DrawTransparentBitmap.
	DrawTransparentBitmap (see page 2984)	This is function GraphUtil.DrawTransparentBitmap.
	GetHighLightColor (see page 2985)	This is function GraphUtil.GetHighLightColor.
	GetShadowColor (see page 2985)	This is function GraphUtil.GetShadowColor.
	GradientFillCanvas (see page 2985)	This is function GraphUtil.GradientFillCanvas.
	LoadCompressedResourceBitmap (see page 2986)	This is function GraphUtil.LoadCompressedResourceBitmap.
	RGBToWebColorName (see page 2986)	This is function GraphUtil.RGBToWebColorName.
	RGBToWebColorStr (see page 2986)	This is function GraphUtil.RGBToWebColorStr.
	ScaleImage (see page 2987)	This is function GraphUtil.ScaleImage.
	SortColorArray (see page 2987)	This is function GraphUtil.SortColorArray.
	SplitTransparentBitmap (see page 2987)	This is function GraphUtil.SplitTransparentBitmap.

	WebColorNameToColor (see page 2988)	This is function GraphUtil.WebColorNameToColor.
	WebColorStrToColor (see page 2988)	This is function GraphUtil.WebColorStrToColor.
	WebColorToRGB (see page 2988)	This is function GraphUtil.WebColorToRGB.

Structs, Records, Enums

	Name	Description
	TArrowType (see page 2989)	This is record GraphUtil.TArrowType.
	TColorArraySortType (see page 2989)	This is record GraphUtil.TColorArraySortType.
	TGradientDirection (see page 2990)	This is record GraphUtil.TGradientDirection.
	TScrollDirection (see page 2990)	This is record GraphUtil.TScrollDirection.

Types

Name	Description
TColorArray (see page 2992)	This is type GraphUtil.TColorArray.

9.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ColorAdjustLuma (see page 2981)	This is function GraphUtil.ColorAdjustLuma.
	ColorHLSToRGB (see page 2982)	This is function GraphUtil.ColorHLSToRGB.
	ColorRGBToHLS (see page 2982)	This is function GraphUtil.ColorRGBToHLS.
	ColorToWebColorName (see page 2982)	This is function GraphUtil.ColorToWebColorName.
	ColorToWebColorStr (see page 2983)	This is function GraphUtil.ColorToWebColorStr.
	DrawArrow (see page 2983)	This is function GraphUtil.DrawArrow.
	DrawCheck (see page 2983)	This is function GraphUtil.DrawCheck.
	DrawChevron (see page 2984)	This is function GraphUtil.DrawChevron.
	DrawTransparentBitmap (see page 2984)	This is function GraphUtil.DrawTransparentBitmap.
	DrawTransparentBitmap (see page 2984)	This is function GraphUtil.DrawTransparentBitmap.
	GetHighLightColor (see page 2985)	This is function GraphUtil.GetHighLightColor.
	GetShadowColor (see page 2985)	This is function GraphUtil.GetShadowColor.
	GradientFillCanvas (see page 2985)	This is function GraphUtil.GradientFillCanvas.
	LoadCompressedResourceBitmap (see page 2986)	This is function GraphUtil.LoadCompressedResourceBitmap.
	RGBToWebColorName (see page 2986)	This is function GraphUtil.RGBToWebColorName.
	RGBToWebColorStr (see page 2986)	This is function GraphUtil.RGBToWebColorStr.
	ScaleImage (see page 2987)	This is function GraphUtil.ScaleImage.
	SortColorArray (see page 2987)	This is function GraphUtil.SortColorArray.
	SplitTransparentBitmap (see page 2987)	This is function GraphUtil.SplitTransparentBitmap.
	WebColorNameToColor (see page 2988)	This is function GraphUtil.WebColorNameToColor.
	WebColorStrToColor (see page 2988)	This is function GraphUtil.WebColorStrToColor.
	WebColorToRGB (see page 2988)	This is function GraphUtil.WebColorToRGB.

9.1.1 GraphUtil.ColorAdjustLuma Function

Pascal

```
function ColorAdjustLuma(clrRGB: TColor; n: Integer; fScale: BOOL): TColor;
```

C++

```
TColor ColorAdjustLuma(TColor clrRGB, int n, BOOL fScale);
```

File

GraphUtil

Description

This is function GraphUtil.ColorAdjustLuma.

9.1.2 GraphUtil.ColorHLSToRGB Function

Pascal

```
function ColorHLSToRGB(Hue: Word; Luminance: Word; Saturation: Word): TColorRef;
```

C++

```
TColorRef ColorHLSToRGB(Word Hue, Word Luminance, Word Saturation);
```

File

GraphUtil

Description

This is function GraphUtil.ColorHLSToRGB.

9.1.3 GraphUtil.ColorRGBToHLS Function

Pascal

```
procedure ColorRGBToHLS(clrRGB: TColorRef; var Hue: Word; var Luminance: Word; var Saturation:
Word);
```

C++

```
ColorRGBToHLS(TColorRef clrRGB, Word Hue, Word Luminance, Word Saturation);
```

File

GraphUtil

Description

This is function GraphUtil.ColorRGBToHLS.

9.1.4 GraphUtil.ColorToWebColorName Function

Pascal

```
function ColorToWebColorName(Color: TColor): string;
```

C++

```
UnicodeString ColorToWebColorName(TColor Color);
```

File

GraphUtil

Description

This is function GraphUtil.ColorToWebColorName.

9.1.5 GraphUtil.ColorToWebColorStr Function

Pascal

```
function ColorToWebColorStr(Color: TColor): string;
```

C++

```
UnicodeString ColorToWebColorStr(TColor Color);
```

File

GraphUtil

Description

This is function GraphUtil.ColorToWebColorStr.

9.1.6 GraphUtil.DrawArrow Function

Pascal

```
procedure DrawArrow(ACanvas: TCanvas; Direction: TScrollDirection; Location: TPoint; Size:
Integer);
```

C++

```
DrawArrow(TCanvas ACanvas, TScrollDirection Direction, TPoint Location, int Size);
```

File

GraphUtil

Description

This is function GraphUtil.DrawArrow.

9.1.7 GraphUtil.DrawCheck Function

Pascal

```
procedure DrawCheck(ACanvas: TCanvas; Location: TPoint; Size: Integer; Shadow: Boolean = True);
```

C++

```
DrawCheck(TCanvas ACanvas, TPoint Location, int Size, Boolean Shadow = True);
```

File

GraphUtil

Description

This is function GraphUtil.DrawCheck.

9.1.8 GraphUtil.DrawChevron Function

Pascal

```
procedure DrawChevron(ACanvas: TCanvas; Direction: TScrollDirection; Location: TPoint; Size:
Integer);
```

C++

```
DrawChevron(TCanvas ACanvas, TScrollDirection Direction, TPoint Location, int Size);
```

File

GraphUtil

Description

This is function GraphUtil.DrawChevron.

9.1.9 GraphUtil.DrawTransparentBitmap Function

Pascal

```
procedure DrawTransparentBitmap(Source: TBitmap; Destination: TCanvas; DestRect: TRect;
Opacity: Byte); overload;
```

C++

```
DrawTransparentBitmap(TBitmap Source, TCanvas Destination, TRect DestRect, Byte Opacity);
```

File

GraphUtil

Description

This is function GraphUtil.DrawTransparentBitmap.

9.1.10 GraphUtil.DrawTransparentBitmap Function

Pascal

```
procedure DrawTransparentBitmap(Source: TBitmap; SourceRect: TRect; Destination: TCanvas;
DestRect: TRect; Opacity: Byte); overload;
```

C++

```
DrawTransparentBitmap(TBitmap Source, TRect SourceRect, TCanvas Destination, TRect DestRect,
Byte Opacity);
```

File

GraphUtil

Description

This is function GraphUtil.DrawTransparentBitmap.

9.1.11 GraphUtil.GetHighLightColor Function

Pascal

```
function GetHighLightColor(const Color: TColor; Luminance: Integer = 19): TColor;
```

C++

```
TColor GetHighLightColor(const TColor Color, int Luminance = 19);
```

File

GraphUtil

Description

This is function GraphUtil.GetHighLightColor.

9.1.12 GraphUtil.GetShadowColor Function

Pascal

```
function GetShadowColor(const Color: TColor; Luminance: Integer = -50): TColor;
```

C++

```
TColor GetShadowColor(const TColor Color, int Luminance = -50);
```

File

GraphUtil

Description

This is function GraphUtil.GetShadowColor.

9.1.13 GraphUtil.GradientFillCanvas Function

Pascal

```
procedure GradientFillCanvas(const ACanvas: TCanvas; const AStartColor: TColor; const
AEndColor: TColor; const ARect: TRect; const Direction: TGradientDirection);
```

C++

```
GradientFillCanvas(const TCanvas ACanvas, const TColor AStartColor, const TColor AEndColor,
const TRect ARect, const TGradientDirection Direction);
```

File

GraphUtil

Description

This is function GraphUtil.GradientFillCanvas.

9.1.14 GraphUtil.LoadCompressedResourceBitmap Function

Pascal

```
function LoadCompressedResourceBitmap(ResID: string): TBitmap;
```

C++

```
TBitmap LoadCompressedResourceBitmap(UnicodeString ResID);
```

File

GraphUtil

Description

This is function GraphUtil.LoadCompressedResourceBitmap.

9.1.15 GraphUtil.RGBToWebColorName Function

Pascal

```
function RGBToWebColorName(RGB: Integer): string;
```

C++

```
UnicodeString RGBToWebColorName(int RGB);
```

File

GraphUtil

Description

This is function GraphUtil.RGBToWebColorName.

9.1.16 GraphUtil.RGBToWebColorStr Function

Pascal

```
function RGBToWebColorStr(RGB: Integer): string;
```

C++

```
UnicodeString RGBToWebColorStr(int RGB);
```

File

GraphUtil

Description

This is function GraphUtil.RGBToWebColorStr.

9.1.17 GraphUtil.ScaleImage Function

Pascal

```
procedure ScaleImage(const SourceBitmap: TBitmap; const ResizedBitmap: TBitmap; const
ScaleAmount: Double);
```

C++

```
ScaleImage(const TBitmap SourceBitmap, const TBitmap ResizedBitmap, const Double ScaleAmount);
```

File

GraphUtil

Description

This is function GraphUtil.ScaleImage.

9.1.18 GraphUtil.SortColorArray Function

Pascal

```
procedure SortColorArray(ColorArray: TColorArray; L: Integer; R: Integer; SortType:
TColorArraySortType; Reverse: Boolean = False);
```

C++

```
SortColorArray(TColorArray ColorArray, int L, int R, TColorArraySortType SortType, Boolean
Reverse = False);
```

File

GraphUtil

Description

This is function GraphUtil.SortColorArray.

9.1.19 GraphUtil.SplitTransparentBitmap Function

Pascal

```
function SplitTransparentBitmap(Source: TBitmap; SourceRect: TRect): TBitmap;
```

C++

```
TBitmap SplitTransparentBitmap(TBitmap Source, TRect SourceRect);
```

File

GraphUtil

Description

This is function GraphUtil.SplitTransparentBitmap.

9.1.20 GraphUtil.WebColorNameToColor Function

Pascal

```
function WebColorNameToColor(WebColorName: string): TColor;
```

C++

```
TColor WebColorNameToColor(UnicodeString WebColorName);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorNameToColor.

9.1.21 GraphUtil.WebColorStrToColor Function

Pascal

```
function WebColorStrToColor(WebColor: string): TColor;
```

C++

```
TColor WebColorStrToColor(UnicodeString WebColor);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorStrToColor.

9.1.22 GraphUtil.WebColorToRGB Function

Pascal

```
function WebColorToRGB(WebColor: Integer): Integer;
```

C++

```
int WebColorToRGB(int WebColor);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorToRGB.

9.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TArrowType (see page 2989)	This is record GraphUtil.TArrowType.
	TColorArraySortType (see page 2989)	This is record GraphUtil.TColorArraySortType.
	TGradientDirection (see page 2990)	This is record GraphUtil.TGradientDirection.
	TScrollDirection (see page 2990)	This is record GraphUtil.TScrollDirection.

9.2.1 GraphUtil.TArrowType Enumeration

Pascal

```
TArrowType = (
  atSolid,
  atArrows
);
```

C++

```
enum TArrowType {
  atSolid,
  atArrows
};
```

File

GraphUtil

Description

This is record GraphUtil.TArrowType.

9.2.2 GraphUtil.TColorArraySortType Enumeration

Pascal

```
TColorArraySortType = (
  stHue,
  stSaturation,
  stLuminance,
  stRed,
  stGreen,
  stBlue,
  stCombo
);
```

C++

```
enum TColorArraySortType {
  stHue,
  stSaturation,
```

```
    stLuminance,
    stRed,
    stGreen,
    stBlue,
    stCombo
  };
```

File

GraphUtil

Description

This is record GraphUtil.TColorArraySortType.

9.2.3 GraphUtil.TGradientDirection Enumeration

Pascal

```
  TGradientDirection = (
    gdHorizontal,
    gdVertical
  );
```

C++

```
  enum TGradientDirection {
    gdHorizontal,
    gdVertical
  };
```

File

GraphUtil

Description

This is record GraphUtil.TGradientDirection.

9.2.4 GraphUtil.TScrollDirection Enumeration

Pascal

```
  TScrollDirection = (
    sdLeft,
    sdRight,
    sdUp,
    sdDown
  );
```

C++

```
  enum TScrollDirection {
    sdLeft,
    sdRight,
    sdUp,
    sdDown
  };
```

File

GraphUtil

Description

This is record GraphUtil.TScrollDirection.

9.3 Types

The following table lists types in this documentation.

Types

Name	Description
TColorArray (see page 2992)	This is type GraphUtil.TColorArray.

9.3.1 GraphUtil.TColorArray Type

Pascal

```
TColorArray = array of TIdentMapEntry;
```

C++

```
array of TIdentMapEntry TColorArray;
```

File

GraphUtil

Description

This is type GraphUtil.TColorArray.

9.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WebNamedColors (see page 2993)	This is constant GraphUtil.WebNamedColors.
WebNamedColorsCount (see page 2995)	This is constant GraphUtil.WebNamedColorsCount.

9.4.1 GraphUtil.WebNamedColors Constant

Pascal

```
WebNamedColors: array[0..WebNamedColorsCount - 1] of TIdentMapEntry = ( (Value: clWebSnow;
Name: 'clWebSnow'), (Value: clWebFloralWhite; Name: 'clWebFloralWhite'), (Value:
clWebLavenderBlush; Name: 'clWebLavenderBlush'), (Value: clWebOldLace; Name: 'clWebOldLace'),
(Value: clWebIvory; Name: 'clWebIvory'), (Value: clWebCornSilk; Name: 'clWebCornSilk'),
(Value: clWebBeige; Name: 'clWebBeige'), (Value: clWebAntiqueWhite; Name:
'clWebAntiqueWhite'), (Value: clWebWheat; Name: 'clWebWheat'), (Value: clWebAliceBlue; Name:
'clWebAliceBlue'), (Value: clWebGhostWhite; Name: 'clWebGhostWhite'), (Value: clWebLavender;
Name: 'clWebLavender'), (Value: clWebSeashell; Name: 'clWebSeashell'), (Value:
clWebLightYellow; Name: 'clWebLightYellow'), (Value: clWebPapayaWhip; Name:
'clWebPapayaWhip'), (Value: clWebNavajoWhite; Name: 'clWebNavajoWhite'), (Value:
clWebMoccasin; Name: 'clWebMoccasin'), (Value: clWebBurlywood; Name: 'clWebBurlywood'),
(Value: clWebAzure; Name: 'clWebAzure'), (Value: clWebMintcream; Name: 'clWebMintcream'),
(Value: clWebHoneydew; Name: 'clWebHoneydew'), (Value: clWebLinen; Name: 'clWebLinen'),
(Value: clWebLemonChiffon; Name: 'clWebLemonChiffon'), (Value: clWebBlanchedAlmond; Name:
'clWebBlanchedAlmond'), (Value: clWebBisque; Name: 'clWebBisque'), (Value: clWebPeachPuff;
Name: 'clWebPeachPuff'), (Value: clWebTan; Name: 'clWebTan'), (Value: clWebYellow; Name:
'clWebYellow'), (Value: clWebDarkOrange; Name: 'clWebDarkOrange'), (Value: clWebRed; Name:
'clWebRed'), (Value: clWebDarkRed; Name: 'clWebDarkRed'), (Value: clWebMaroon; Name:
'clWebMaroon'), (Value: clWebIndianRed; Name: 'clWebIndianRed'), (Value: clWebSalmon; Name:
'clWebSalmon'), (Value: clWebCoral; Name: 'clWebCoral'), (Value: clWebGold; Name:
'clWebGold'), (Value: clWebTomato; Name: 'clWebTomato'), (Value: clWebCrimson; Name:
'clWebCrimson'), (Value: clWebBrown; Name: 'clWebBrown'), (Value: clWebChocolate; Name:
'clWebChocolate'), (Value: clWebSandyBrown; Name: 'clWebSandyBrown'), (Value:
clWebLightSalmon; Name: 'clWebLightSalmon'), (Value: clWebLightCoral; Name:
'clWebLightCoral'), (Value: clWebOrange; Name: 'clWebOrange'), (Value: clWebOrangeRed; Name:
'clWebOrangeRed'), (Value: clWebFirebrick; Name: 'clWebFirebrick'), (Value: clWebSaddleBrown;
Name: 'clWebSaddleBrown'), (Value: clWebSienna; Name: 'clWebSienna'), (Value: clWebPeru; Name:
'clWebPeru'), (Value: clWebDarkSalmon; Name: 'clWebDarkSalmon'), (Value: clWebRosyBrown; Name:
'clWebRosyBrown'), (Value: clWebPaleGoldenrod; Name: 'clWebPaleGoldenrod'), (Value:
clWebLightGoldenrodYellow; Name: 'clWebLightGoldenrodYellow'), (Value: clWebOlive; Name:
'clWebOlive'), (Value: clWebForestGreen; Name: 'clWebForestGreen'), (Value: clWebGreenYellow;
Name: 'clWebGreenYellow'), (Value: clWebChartreuse; Name: 'clWebChartreuse'), (Value:
clWebLightGreen; Name: 'clWebLightGreen'), (Value: clWebAquamarine; Name: 'clWebAquamarine'),
(Value: clWebSeaGreen; Name: 'clWebSeaGreen'), (Value: clWebGoldenRod; Name:
'clWebGoldenRod'), (Value: clWebKhaki; Name: 'clWebKhaki'), (Value: clWebOliveDrab; Name:
'clWebOliveDrab'), (Value: clWebGreen; Name: 'clWebGreen'), (Value: clWebYellowGreen; Name:
'clWebYellowGreen'), (Value: clWebLawnGreen; Name: 'clWebLawnGreen'), (Value: clWebPaleGreen;
Name: 'clWebPaleGreen'), (Value: clWebMediumAquamarine; Name: 'clWebMediumAquamarine'),
(Value: clWebMediumSeaGreen; Name: 'clWebMediumSeaGreen'), (Value: clWebDarkGoldenRod; Name:
'clWebDarkGoldenRod'), (Value: clWebDarkKhaki; Name: 'clWebDarkKhaki'), (Value:
clWebDarkOliveGreen; Name: 'clWebDarkOliveGreen'), (Value: clWebDarkgreen; Name:
'clWebDarkgreen'), (Value: clWebLimeGreen; Name: 'clWebLimeGreen'), (Value: clWebLime; Name:
'clWebLime'), (Value: clWebSpringGreen; Name: 'clWebSpringGreen'), (Value:
```

```
clWebMediumSpringGreen; Name: 'clWebMediumSpringGreen'), (Value: clWebDarkSeaGreen; Name:
'clWebDarkSeaGreen'), (Value: clWebLightSeaGreen; Name: 'clWebLightSeaGreen'), (Value:
clWebPaleTurquoise; Name: 'clWebPaleTurquoise'), (Value: clWebLightCyan; Name:
'clWebLightCyan'), (Value: clWebLightBlue; Name: 'clWebLightBlue'), (Value: clWebLightSkyBlue;
Name: 'clWebLightSkyBlue'), (Value: clWebCornFlowerBlue; Name: 'clWebCornFlowerBlue'), (Value:
clWebDarkBlue; Name: 'clWebDarkBlue'), (Value: clWebIndigo; Name: 'clWebIndigo'), (Value:
clWebMediumTurquoise; Name: 'clWebMediumTurquoise'), (Value: clWebTurquoise; Name:
'clWebTurquoise'), (Value: clWebCyan; Name: 'clWebCyan'), (Value: clWebPowderBlue; Name:
'clWebPowderBlue'), (Value: clWebSkyBlue; Name: 'clWebSkyBlue'), (Value: clWebRoyalBlue; Name:
'clWebRoyalBlue'), (Value: clWebMediumBlue; Name: 'clWebMediumBlue'), (Value:
clWebMidnightBlue; Name: 'clWebMidnightBlue'), (Value: clWebDarkTurquoise; Name:
'clWebDarkTurquoise'), (Value: clWebCadetBlue; Name: 'clWebCadetBlue'), (Value: clWebDarkCyan;
Name: 'clWebDarkCyan'), (Value: clWebTeal; Name: 'clWebTeal'), (Value: clWebDeepSkyBlue; Name:
'clWebDeepskyBlue'), (Value: clWebDodgerBlue; Name: 'clWebDodgerBlue'), (Value: clWebBlue;
Name: 'clWebBlue'), (Value: clWebNavy; Name: 'clWebNavy'), (Value: clWebDarkViolet; Name:
'clWebDarkViolet'), (Value: clWebDarkOrchid; Name: 'clWebDarkOrchid'), (Value: clWebMagenta;
Name: 'clWebMagenta'), (Value: clWebDarkMagenta; Name: 'clWebDarkMagenta'), (Value:
clWebMediumVioletRed; Name: 'clWebMediumVioletRed'), (Value: clWebPaleVioletRed; Name:
'clWebPaleVioletRed'), (Value: clWebBlueViolet; Name: 'clWebBlueViolet'), (Value:
clWebMediumOrchid; Name: 'clWebMediumOrchid'), (Value: clWebMediumPurple; Name:
'clWebMediumPurple'), (Value: clWebPurple; Name: 'clWebPurple'), (Value: clWebDeepPink; Name:
'clWebDeepPink'), (Value: clWebLightPink; Name: 'clWebLightPink'), (Value: clWebViolet; Name:
'clWebViolet'), (Value: clWebOrchid; Name: 'clWebOrchid'), (Value: clWebPlum; Name:
'clWebPlum'), (Value: clWebThistle; Name: 'clWebThistle'), (Value: clWebHotPink; Name:
'clWebHotPink'), (Value: clWebPink; Name: 'clWebPink'), (Value: clWebLightSteelBlue; Name:
'clWebLightSteelBlue'), (Value: clWebMediumSlateBlue; Name: 'clWebMediumSlateBlue'), (Value:
clWebLightSlateGray; Name: 'clWebLightSlateGray'), (Value: clWebWhite; Name: 'clWebWhite'),
(Value: clWebLightgrey; Name: 'clWebLightgrey'), (Value: clWebGray; Name: 'clWebGray'),
(Value: clWebSteelBlue; Name: 'clWebSteelBlue'), (Value: clWebSlateBlue; Name:
'clWebSlateBlue'), (Value: clWebSlateGray; Name: 'clWebSlateGray'), (Value: clWebWhiteSmoke;
Name: 'clWebWhiteSmoke'), (Value: clWebSilver; Name: 'clWebSilver'), (Value: clWebDimGray;
Name: 'clWebDimGray'), (Value: clWebMistyRose; Name: 'clWebMistyRose'), (Value:
clWebDarkSlateBlue; Name: 'clWebDarkSlateBlue'), (Value: clWebDarkSlategray; Name:
'clWebDarkSlategray'), (Value: clWebGainsboro; Name: 'clWebGainsboro'), (Value: clWebDarkGray;
Name: 'clWebDarkGray'), (Value: clWebBlack; Name: 'clWebBlack'));
```

C++

```
array[0..WebNamedColorsCount - 1] of TIdentMapEntry WebNamedColors = ( (Value: clWebSnow;
Name: 'clWebSnow'), (Value: clWebFloralWhite; Name: 'clWebFloralWhite'), (Value:
clWebLavenderBlush; Name: 'clWebLavenderBlush'), (Value: clWebOldLace; Name: 'clWebOldLace'),
(Value: clWebIvory; Name: 'clWebIvory'), (Value: clWebCornSilk; Name: 'clWebCornSilk'),
(Value: clWebBeige; Name: 'clWebBeige'), (Value: clWebAntiqueWhite; Name:
'clWebAntiqueWhite'), (Value: clWebWheat; Name: 'clWebWheat'), (Value: clWebAliceBlue; Name:
'clWebAliceBlue'), (Value: clWebGhostWhite; Name: 'clWebGhostWhite'), (Value: clWebLavender;
Name: 'clWebLavender'), (Value: clWebSeashell; Name: 'clWebSeashell'), (Value:
clWebLightYellow; Name: 'clWebLightYellow'), (Value: clWebPapayaWhip; Name:
'clWebPapayaWhip'), (Value: clWebNavajoWhite; Name: 'clWebNavajoWhite'), (Value:
clWebMoccasin; Name: 'clWebMoccasin'), (Value: clWebBurlywood; Name: 'clWebBurlywood'),
(Value: clWebAzure; Name: 'clWebAzure'), (Value: clWebMintcream; Name: 'clWebMintcream'),
(Value: clWebHoneydew; Name: 'clWebHoneydew'), (Value: clWebLinen; Name: 'clWebLinen'),
(Value: clWebLemonChiffon; Name: 'clWebLemonChiffon'), (Value: clWebBlanchedAlmond; Name:
'clWebBlanchedAlmond'), (Value: clWebBisque; Name: 'clWebBisque'), (Value: clWebPeachPuff;
Name: 'clWebPeachPuff'), (Value: clWebTan; Name: 'clWebTan'), (Value: clWebYellow; Name:
'clWebYellow'), (Value: clWebDarkOrange; Name: 'clWebDarkOrange'), (Value: clWebRed; Name:
'clWebRed'), (Value: clWebDarkRed; Name: 'clWebDarkRed'), (Value: clWebMaroon; Name:
'clWebMaroon'), (Value: clWebIndianRed; Name: 'clWebIndianRed'), (Value: clWebSalmon; Name:
'clWebSalmon'), (Value: clWebCoral; Name: 'clWebCoral'), (Value: clWebGold; Name:
'clWebGold'), (Value: clWebTomato; Name: 'clWebTomato'), (Value: clWebCrimson; Name:
'clWebCrimson'), (Value: clWebBrown; Name: 'clWebBrown'), (Value: clWebChocolate; Name:
'clWebChocolate'), (Value: clWebSandyBrown; Name: 'clWebSandyBrown'), (Value:
clWebLightSalmon; Name: 'clWebLightSalmon'), (Value: clWebLightCoral; Name:
'clWebLightCoral'), (Value: clWebOrange; Name: 'clWebOrange'), (Value: clWebOrangeRed; Name:
'clWebOrangeRed'), (Value: clWebFirebrick; Name: 'clWebFirebrick'), (Value: clWebSaddleBrown;
Name: 'clWebSaddleBrown'), (Value: clWebSienna; Name: 'clWebSienna'), (Value: clWebPeru; Name:
'clWebPeru'), (Value: clWebDarkSalmon; Name: 'clWebDarkSalmon'), (Value: clWebRosyBrown; Name:
'clWebRosyBrown'), (Value: clWebPaleGoldenrod; Name: 'clWebPaleGoldenrod'), (Value:
```

```
clWebLightGoldenrodYellow; Name: 'clWebLightGoldenrodYellow'), (Value: clWebOlive; Name:
'clWebOlive'), (Value: clWebForestGreen; Name: 'clWebForestGreen'), (Value: clWebGreenYellow;
Name: 'clWebGreenYellow'), (Value: clWebChartreuse; Name: 'clWebChartreuse'), (Value:
clWebLightGreen; Name: 'clWebLightGreen'), (Value: clWebAquamarine; Name: 'clWebAquamarine'),
(Value: clWebSeaGreen; Name: 'clWebSeaGreen'), (Value: clWebGoldenRod; Name:
'clWebGoldenRod'), (Value: clWebKhaki; Name: 'clWebKhaki'), (Value: clWebOliveDrab; Name:
'clWebOliveDrab'), (Value: clWebGreen; Name: 'clWebGreen'), (Value: clWebYellowGreen; Name:
'clWebYellowGreen'), (Value: clWebLawnGreen; Name: 'clWebLawnGreen'), (Value: clWebPaleGreen;
Name: 'clWebPaleGreen'), (Value: clWebMediumAquamarine; Name: 'clWebMediumAquamarine'),
(Value: clWebMediumSeaGreen; Name: 'clWebMediumSeaGreen'), (Value: clWebDarkGoldenRod; Name:
'clWebDarkGoldenRod'), (Value: clWebDarkKhaki; Name: 'clWebDarkKhaki'), (Value:
clWebDarkOliveGreen; Name: 'clWebDarkOliveGreen'), (Value: clWebDarkgreen; Name:
'clWebDarkgreen'), (Value: clWebLimeGreen; Name: 'clWebLimeGreen'), (Value: clWebLime; Name:
'clWebLime'), (Value: clWebSpringGreen; Name: 'clWebSpringGreen'), (Value:
clWebMediumSpringGreen; Name: 'clWebMediumSpringGreen'), (Value: clWebDarkSeaGreen; Name:
'clWebDarkSeaGreen'), (Value: clWebLightSeaGreen; Name: 'clWebLightSeaGreen'), (Value:
clWebPaleTurquoise; Name: 'clWebPaleTurquoise'), (Value: clWebLightCyan; Name:
'clWebLightCyan'), (Value: clWebLightBlue; Name: 'clWebLightBlue'), (Value: clWebLightSkyBlue;
Name: 'clWebLightSkyBlue'), (Value: clWebCornFlowerBlue; Name: 'clWebCornFlowerBlue'), (Value:
clWebDarkBlue; Name: 'clWebDarkBlue'), (Value: clWebIndigo; Name: 'clWebIndigo'), (Value:
clWebMediumTurquoise; Name: 'clWebMediumTurquoise'), (Value: clWebTurquoise; Name:
'clWebTurquoise'), (Value: clWebCyan; Name: 'clWebCyan'), (Value: clWebPowderBlue; Name:
'clWebPowderBlue'), (Value: clWebSkyBlue; Name: 'clWebSkyBlue'), (Value: clWebRoyalBlue; Name:
'clWebRoyalBlue'), (Value: clWebMediumBlue; Name: 'clWebMediumBlue'), (Value:
clWebMidnightBlue; Name: 'clWebMidnightBlue'), (Value: clWebDarkTurquoise; Name:
'clWebDarkTurquoise'), (Value: clWebCadetBlue; Name: 'clWebCadetBlue'), (Value: clWebDarkCyan;
Name: 'clWebDarkCyan'), (Value: clWebTeal; Name: 'clWebTeal'), (Value: clWebDeepSkyBlue; Name:
'clWebDeepskyBlue'), (Value: clWebDodgerBlue; Name: 'clWebDodgerBlue'), (Value: clWebBlue;
Name: 'clWebBlue'), (Value: clWebNavy; Name: 'clWebNavy'), (Value: clWebDarkViolet; Name:
'clWebDarkViolet'), (Value: clWebDarkOrchid; Name: 'clWebDarkOrchid'), (Value: clWebMagenta;
Name: 'clWebMagenta'), (Value: clWebDarkMagenta; Name: 'clWebDarkMagenta'), (Value:
clWebMediumVioletRed; Name: 'clWebMediumVioletRed'), (Value: clWebPaleVioletRed; Name:
'clWebPaleVioletRed'), (Value: clWebBlueViolet; Name: 'clWebBlueViolet'), (Value:
clWebMediumOrchid; Name: 'clWebMediumOrchid'), (Value: clWebMediumPurple; Name:
'clWebMediumPurple'), (Value: clWebPurple; Name: 'clWebPurple'), (Value: clWebDeepPink; Name:
'clWebDeepPink'), (Value: clWebLightPink; Name: 'clWebLightPink'), (Value: clWebViolet; Name:
'clWebViolet'), (Value: clWebOrchid; Name: 'clWebOrchid'), (Value: clWebPlum; Name:
'clWebPlum'), (Value: clWebThistle; Name: 'clWebThistle'), (Value: clWebHotPink; Name:
'clWebHotPink'), (Value: clWebPink; Name: 'clWebPink'), (Value: clWebLightSteelBlue; Name:
'clWebLightSteelBlue'), (Value: clWebMediumSlateBlue; Name: 'clWebMediumSlateBlue'), (Value:
clWebLightSlateGray; Name: 'clWebLightSlateGray'), (Value: clWebWhite; Name: 'clWebWhite'),
(Value: clWebLightgrey; Name: 'clWebLightgrey'), (Value: clWebGray; Name: 'clWebGray'),
(Value: clWebSteelBlue; Name: 'clWebSteelBlue'), (Value: clWebSlateBlue; Name:
'clWebSlateBlue'), (Value: clWebSlateGray; Name: 'clWebSlateGray'), (Value: clWebWhiteSmoke;
Name: 'clWebWhiteSmoke'), (Value: clWebSilver; Name: 'clWebSilver'), (Value: clWebDimGray;
Name: 'clWebDimGray'), (Value: clWebMistyRose; Name: 'clWebMistyRose'), (Value:
clWebDarkSlateBlue; Name: 'clWebDarkSlateBlue'), (Value: clWebDarkSlategray; Name:
'clWebDarkSlategray'), (Value: clWebGainsboro; Name: 'clWebGainsboro'), (Value: clWebDarkGray;
Name: 'clWebDarkGray'), (Value: clWebBlack; Name: 'clWebBlack'));
```

File

GraphUtil

Description

This is constant GraphUtil.WebNamedColors.

9.4.2 GraphUtil.WebNamedColorsCount Constant

Pascal

```
WebNamedColorsCount = 138;
```

C++

```
WebNamedColorsCount = 138;
```

File

GraphUtil

Description

This is constant GraphUtil.WebNamedColorsCount.

10 Grids Namespace

This is namespace Grids.

Topics

Name	Description
Classes (see page 2999)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3199)	The following table lists structs, records, enums in this documentation.
Types (see page 3205)	The following table lists types in this documentation.
Constants (see page 3212)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidGridOperation (see page 2999)	EInvalidGridOperation is the exception class for illegal grid operations.
	TCustomDrawGrid (see page 3000)	TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.
	TCustomGrid (see page 3027)	TCustomGrid is the base type for all components that present information in a two-dimensional grid.
	TDrawGrid (see page 3083)	TDrawGrid represents a grid control that displays information in column and row format.
	TInplaceEdit (see page 3161)	TInplaceEdit provides text editing capabilities for grid controls.
	TInplaceEditList (see page 3171)	This is class Grids.TInplaceEditList.
	TStringGrid (see page 3180)	TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.
	TStringGridStrings (see page 3192)	TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

Constants

Name	Description
MaxCustomExtents (see page 3212)	This is constant Grids.MaxCustomExtents.
MaxShortInt (see page 3212)	This is constant Grids.MaxShortInt.

Structs, Records, Enums

	Name	Description
	TEditStyle (see page 3199)	TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.
	TGridAxisDrawInfo (see page 3200)	TGridAxisDrawInfo and TGridDrawInfo (see page 3201) represent information that is useful when rendering a grid.
	TGridCoord (see page 3200)	TGridCoord specifies a cell in a grid control.

	TGridDrawInfo (see page 3201)	This is record Grids.TGridDrawInfo.
	TGridOption (see page 3201)	TGridOption and TGridOptions (see page 3207) customize the appearance and behavior of a grid control.
	TGridRect (see page 3203)	TGridRect specifies a set of cells that occupy a rectangular region in a grid control.
	TGridState (see page 3203)	TGridState indicates any user actions that are currently changing the layout of the grid.

Types

Name	Description
TDrawCellEvent (see page 3205)	Function type for OnDrawCell event.
TGetEditEvent (see page 3206)	TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.
TGetExtentsFunc (see page 3206)	This is type Grids.TGetExtentsFunc.
TGridDrawState (see page 3206)	TGridDrawState represents information that is useful when rendering the cell of a grid.
TGridMovement (see page 3207)	This is type Grids.TGridMovement.
TGridOptions (see page 3207)	TGridOptions and TGridOption (see page 3201) customize the appearance and behavior of a grid control.
TGridScrollDirection (see page 3209)	TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.
TMovedEvent (see page 3210)	This is type Grids.TMovedEvent.
TOnGetPickListItems (see page 3210)	This is type Grids.TOnGetPickListItems.
TSelectCellEvent (see page 3210)	This is type Grids.TSelectCellEvent.
TSetEditEvent (see page 3211)	TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

10.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidGridOperation (see page 2999)	EInvalidGridOperation is the exception class for illegal grid operations.
	TCustomDrawGrid (see page 3000)	TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.
	TCustomGrid (see page 3027)	TCustomGrid is the base type for all components that present information in a two-dimensional grid.
	TDrawGrid (see page 3083)	TDrawGrid represents a grid control that displays information in column and row format.
	TInplaceEdit (see page 3161)	TInplaceEdit provides text editing capabilities for grid controls.
	TInplaceEditList (see page 3171)	This is class Grids.TInplaceEditList.
	TStringGrid (see page 3180)	TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.
	TStringGridStrings (see page 3192)	TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

10.1.1 EInvalidGridOperation Class

EInvalidGridOperation is the exception class for illegal grid operations.

Class Hierarchy

Pascal

```
EInvalidGridOperation = class(Exception);
```

C++

```
class EInvalidGridOperation : public Exception;
```

File

Grids

Description

EInvalidGridOperation is raised when an application attempts an illegal operation on a grid. For example, EInvalidGridOperation is raised if an application tries to access a cell or column that does not exist within the grid.

See Also

TStringGrid (see page 3180)

TCustomGrid (see page 3027)

10.1.2 TCustomDrawGrid Class

TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.

Class Hierarchy

Pascal

```
TCustomDrawGrid = class(TCustomGrid);
```

C++

```
class TCustomDrawGrid : public TCustomGrid;
```

File

Grids

Description

Use TCustomDrawGrid as a base class when defining grid components that publish events for painting or manipulating the cells in the grid. TCustomDrawGrid introduces a number of events and methods that take advantage of the tabular organization of the grid in responding to user actions.

Do not create instances of TCustomDrawGrid. Instead, put a grid control in a form by instantiating a descendant of TCustomDrawGrid. Use TDrawGrid (see page 3083) to represent non-textual material in a tabular grid. For text, use a TStringGrid (see page 3180) object. If the text represents field values from a dataset, use TDBGrid. If you are displaying Name/Value pairs, use TValueListEditor.

See Also

TDrawGrid (see page 3083)

TStringGrid (see page 3180)

10.1.2.1 TCustomDrawGrid Members

The following tables list the members exposed by TCustomDrawGrid.

Protected Events

	Name	Description
	OnColumnMoved (see page 3019)	Occurs immediately after the position of a column changes.
	OnDrawCell (see page 3020)	Occurs when a cell in the grid needs to be drawn.
	OnGetEditMask (see page 3021)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3022)	Occurs when the in-place editor requests the value of a cell.
	OnRowMoved (see page 3023)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3025)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3026)	Occurs when the user edits the value of a cell.
	OnTopLeftChanged (see page 3027)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.

	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	ColumnMoved (see page 3004)	Generates an OnColumnMoved (see page 3019) event.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
	CreateEditor (see page 3040)	Creates the inplace editor control.
	CreateParams (see page 3041)	Initializes the window-creation parameter record.
	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.
	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
	DeleteRow (see page 3042)	Deletes a specified from the grid.
	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
	DrawCell (see page 3004)	Generates an OnDrawCell (see page 3020) event.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3004)	Generates an OnGetEditMask (see page 3021) event.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 3005)	Generates an OnGetEditText (see page 3022) event.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	KeyDown (see page 3050)	Provides special processing when the user presses a key.
	KeyPress (see page 3050)	Provides special processing when the user presses a key.

	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3005)	Generates an OnRowMoved (see page 3023) event.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 3006)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 3006)	Generates an OnSetEditText (see page 3026) event.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3007)	Generates an OnTopLeftChanged (see page 3027) event.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	BorderStyle (see page 3060)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	Color (see page 3063)	Specifies the background color of the control.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	Options (see page 3072)	Specifies various display and behavioral properties of the grid.
	ParentColor (see page 3073)	Determines where a control looks for its color information.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	ScrollBars (see page 3077)	Specifies whether the grid includes horizontal and vertical scroll bars.
	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Public Methods

	Name	Description
	CellRect (see page 3003)	Returns the screen coordinates of a cell in the grid.
	Create (see page 3030)	Creates an instance of TCustomGrid (see page 3027).
	Destroy (see page 3031)	Destroys an instance of TCustomGrid (see page 3027).
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 3005)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

Public Properties

	Name	Description
	Canvas (see page 3007)	Specifies the TCanvas object that presents a drawing surface for the control.
	Col (see page 3009)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 3011)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 3011)	Determines whether the current cell can be edited.
	GridHeight (see page 3012)	Specifies the height of the grid in pixels.
	GridWidth (see page 3012)	Specifies the width of the grid in pixels.
	LeftCol (see page 3012)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 3013)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 3014)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 3015)	Indicates the boundaries of the current selection.
	TabStops (see page 3017)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3017)	Specifies the index of the first visible scrollable row in the grid.

Published Properties

	Name	Description
	TabStop (see page 3079)	Determines if the user can tab to a control.

10.1.2.2 TCustomDrawGrid Methods

The methods of the TCustomDrawGrid class are listed here.

Protected Methods

	Name	Description
	ColumnMoved (see page 3004)	Generates an OnColumnMoved (see page 3019) event.
	DrawCell (see page 3004)	Generates an OnDrawCell (see page 3020) event.
	GetEditMask (see page 3004)	Generates an OnGetEditMask (see page 3021) event.
	GetEditText (see page 3005)	Generates an OnGetEditText (see page 3022) event.
	RowMoved (see page 3005)	Generates an OnRowMoved (see page 3023) event.
	SelectCell (see page 3006)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 3006)	Generates an OnSetEditText (see page 3026) event.
	TopLeftChanged (see page 3007)	Generates an OnTopLeftChanged (see page 3027) event.

Public Methods

	Name	Description
	CellRect (see page 3003)	Returns the screen coordinates of a cell in the grid.
	MouseToCell (see page 3005)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

10.1.2.2.1 TCustomDrawGrid.CellRect Method

Returns the screen coordinates of a cell in the grid.

Pascal

```
function CellRect(ACol: Longint; ARow: Longint): TRect;
```

C++

```
__fastcall TRect CellRect(Longint ACol, Longint ARow);
```

Description

Call CellRect to obtain the boundaries, in pixels, of the cell in the column and row indicated by the ACol and ARow parameters. The top left cell is specified when ACol and ARow are set to zero.

If the indicated cell is not visible, CellRect returns an empty rectangle.

See Also

MouseToCell (see page 3005)

10.1.2.2.2 TCustomDrawGrid.ColumnMoved Method

Generates an OnColumnMoved (see page 3019) event.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. It overrides the inherited method to generate an OnColumnMoved (see page 3019) event. Descendants of TCustomDrawGrid (see page 3000) can override this method to make internal adjustments or to block the OnColumnMoved (see page 3019) event.

See Also

OnColumnMoved (see page 3019)

10.1.2.2.3 TCustomDrawGrid.DrawCell Method

Generates an OnDrawCell (see page 3020) event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing (see page 3066) property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell. Descendants of TCustomDrawGrid (see page 3000) can override this method to draw the contents of the cell or to block the OnDrawCell (see page 3020) event.

See Also

DefaultDrawing (see page 3066)

FixedCols (see page 3067)

FixedRows (see page 3068)

OnDrawCell (see page 3020)

10.1.2.2.4 TCustomDrawGrid.GetEditMask Method

Generates an OnGetEditMask (see page 3021) event.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns the Value parameter set by the OnGetEditMask (see page 3021) event handler. If there is no OnGetEditMask (see page 3021) event handler, GetEditMask returns an empty string, indicating that there is no edit mask.

See Also

OnGetEditMask (see page 3021)

10.1.2.2.5 TCustomDrawGrid.GetEditText Method

Generates an OnGetEditText (see page 3022) event.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns the Value parameter set by the OnGetEditText (see page 3022) event handler. If there is no OnGetEditText (see page 3022) event handler, GetEditText returns an empty string.

See Also

OnGetEditText (see page 3022)

10.1.2.2.6 TCustomDrawGrid.MouseToCell Method

Returns the column and row of the cell at the position with screen coordinates (X,Y).

Pascal

```
procedure MouseToCell(X: Integer; Y: Integer; var ACol: Longint; var ARow: Longint);
```

C++

```
__fastcall MouseToCell(int X, int Y, Longint ACol, Longint ARow);
```

Description

Call MouseToCell to convert from grid-relative screen coordinates to row and column indexes. The X and Y parameters are the screen coordinates of the point to be converted. MouseToCell returns the ACol parameter as the number of the column over the point (X,Y), and the ARow parameter as the number of the row.

Usually the MouseToCell method is used in a mouse event handler, which supplies the mouse coordinates as the X and Y parameters of the method call.

See Also

CellRect (see page 3003)

10.1.2.2.7 TCustomDrawGrid.RowMoved Method

Generates an OnRowMoved (see page 3023) event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved (see page 3023) event. Descendants of TCustomDrawGrid (see page 3000) can override this method to make internal adjustments or to block the OnRowMoved (see page 3023) event.

See Also

OnRowMoved (see page 3023)

10.1.2.2.8 TCustomDrawGrid.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an OnSelectCell (see page 3025) event. This method then returns true, unless the OnSelectCell (see page 3025) event handler indicates that the cell should not be selected. Descendants of TCustomDrawGrid (see page 3000) can override this method to check the properties of the information displayed in the cell or block the OnSelectCell (see page 3025) event.

See Also

OnSelectCell (see page 3025)

10.1.2.2.9 TCustomDrawGrid.SetEditText Method

Generates an OnSetEditText (see page 3026) event.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(Longint ACol, Longint ARow, const UnicodeString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. SetEditText overrides the inherited parent class's method to generate an OnSetEditText (see page 3026) event. Descendants of TCustomDrawGrid (see page 3000) can override this method to update an internal representation of the cell's value, or to block the OnSetEditText (see page 3026) event.

See Also

OnSetEditText (see page 3026)

10.1.2.2.10 TCustomDrawGrid.TopLeftChanged Method

Generates an OnTopLeftChanged (see page 3027) event.

Pascal

```
procedure TopLeftChanged; override;
```

C++

```
virtual __fastcall TopLeftChanged();
```

Description

Grid controls call TopLeftChanged internally, whenever the LeftCol (see page 3012) or TopRow (see page 3017) property changes. After calling the inherited parent class's method to update the position of the inplace editor, TopLeftChanged generates an OnTopLeftChanged (see page 3027) event.

See Also

LeftCol (see page 3072)

OnTopLeftChanged (see page 3027)

TopRow (see page 3080)

10.1.2.3 TCustomDrawGrid Properties

The properties of the TCustomDrawGrid class are listed here.

Public Properties

	Name	Description
	Canvas (see page 3007)	Specifies the TCanvas object that presents a drawing surface for the control.
	Col (see page 3009)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 3011)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 3011)	Determines whether the current cell can be edited.
	GridHeight (see page 3012)	Specifies the height of the grid in pixels.
	GridWidth (see page 3012)	Specifies the width of the grid in pixels.
	LeftCol (see page 3012)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 3013)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 3014)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 3015)	Indicates the boundaries of the current selection.
	TabStops (see page 3017)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3017)	Specifies the index of the first visible scrollable row in the grid.

10.1.2.3.1 TCustomDrawGrid.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = dynamic_cast<TStringGrid *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
```

```
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint

10.1.2.3.2 TCustomDrawGrid.Col Property

Specifies the index of the column that contains the selected cell.

Pascal

```
property Col: Longint;
```

C++

```
__property Longint Col;
```

Description

Use Col at runtime to determine the current column in the grid. Setting Col moves focus to the cell in the current row that is in the new column. The first column has an index of 0, the second column an index of 1, and so on.

The selected cell in the grid can be located by reading the Col property and the Row property to obtain the indexes of its column

and row. When focus moves to the grid, the selected cell gets input focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
*/
void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
  Label1->Caption = "From OnClick: The cursor is in column " +
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(index) +
      " has the focus.";
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
}
procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
```

```
    DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

Row (see page 3074)

10.1.2.3.3 TCustomDrawGrid.ColWidths Property

Indicates the width (in pixels) of all the columns in the grid.

Pascal

```
property ColWidths [Index: Longint]: Integer;
```

C++

```
__property int ColWidths[Longint Index];
```

Description

Read ColWidths to determine the width of the cells in the column specified by the Index parameter. An index of 0 refers to the first column. Set ColWidths at runtime to change the width of an individual column.

If the width of a column has not been set explicitly by resizing with the mouse, or by using the ColWidths property, its width is DefaultColWidth.

Use the ColCount property to determine the number of entries in the ColWidths array.

See Also

ColCount (see page 3062)

DefaultColWidth (see page 3064)

RowHeights (see page 3076)

10.1.2.3.4 TCustomDrawGrid.EditorMode Property

Determines whether the current cell can be edited.

Pascal

```
property EditorMode: Boolean;
```

C++

```
__property Boolean EditorMode;
```

Description

Set EditorMode to true, at runtime, to put the grid in edit mode. When EditorMode is true, the user can edit cells in the grid. When the user presses F2, EditorMode is set to true. When the user presses Enter, the value of EditorMode is toggled.

Note: EditorMode has no effect if the Options property does not include goEditing or if the Options property includes goAlwaysShowEditor.

See Also

HideEditor (see page 3048)

Options (see page 3072)

ShowEditor (see page 3056)

10.1.2.3.5 TCustomDrawGrid.GridHeight Property

Specifies the height of the grid in pixels.

Pascal

```
property GridHeight: Integer;
```

C++

```
__property int GridHeight;
```

Description

Read GridHeight to determine the size of the grid. If GridHeight is less than the value of ClientHeight, all of the rows of the grid appear in the control with an empty region below the grid. If the underlying grid is too tall to appear in the control, GridHeight is the same as ClientHeight, and the user must scroll to see the entire contents of the grid.

See Also

ClientHeight

GridWidth (see page 3070)

10.1.2.3.6 TCustomDrawGrid.GridWidth Property

Specifies the width of the grid in pixels.

Pascal

```
property GridWidth: Integer;
```

C++

```
__property int GridWidth;
```

Description

Read GridWidth to determine the size of the grid. If GridWidth is less than the value of ClientWidth, all of the columns of the grid appear in the control with an empty region to the right of the grid. If the underlying grid is too wide to appear in the control, GridWidth is the same as ClientWidth, and the user must scroll to see the entire contents of the grid.

See Also

ClientWidth

GridHeight (see page 3068)

10.1.2.3.7 TCustomDrawGrid.LeftCol Property

Specifies the index of the first visible scrollable column in the grid.

Pascal

```
property LeftCol: Longint;
```

C++

```
__property Longint LeftCol;
```

Description

Read LeftCol to determine the index of the first column in the scrollable region that is visible. Set LeftCol to scroll the columns in the grid so that the column with index LeftCol is the first column after the fixed columns.

See Also

FixedCols (see page 3067)

ScrollData (see page 3055)

VisibleColCount (see page 3082)

10.1.2.3.8 TCustomDrawGrid.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

C++ Examples:

```
/*
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items = StringGrid1->Rows[StringGrid1->Row];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{

  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
```

```
    StringGrid1->Cells[3][1] = "is";
    StringGrid1->Cells[4][1] = "excellent";
    StringGrid1->Cells[0][2] = "Row 2";
    StringGrid1->Cells[1][2] = "Delphi";
    StringGrid1->Cells[2][2] = "is";
    StringGrid1->Cells[4][2] = "RAD";

}
```

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

Col (see page 3060)

10.1.2.3.9 TCustomDrawGrid.RowHeights Property

Gives the height (in pixels) of all rows in the grid.

Pascal

```
property RowHeights [Index: Longint]: Integer;
```

C++

```
__property int RowHeights[Longint Index];
```

Description

Read RowHeights to determine the height of the cells in the row specified by the Index parameter. An index of 0 refers to the first row. Set RowWidths at runtime to change the height of an individual row.

If the height of a row has not been set explicitly by resizing with the mouse, or by using the RowHeights property, it's height is DefaultRowHeight.

Use the RowCount property to determine the number of entries in the RowHeights array.

See Also

ColWidths (see page 3063)

DefaultRowHeight (see page 3066)

RowCount (see page 3075)

10.1.2.3.10 TCustomDrawGrid.Selection Property

Indicates the boundaries of the current selection.

Pascal

```
property Selection: TGridRect;
```

C++

```
__property TGridRect Selection;
```

Description

Set Selection to select a range of cells in the grid. Selection can either represent the first column, first row, last column and last row in the grid, or the row and column coordinates of the upper left and bottom right cells in the selected region.

Selection can only indicate more than one cell when Options includes goRangeSelect.

C++ Examples:

```
/*
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TGridRect myRect;
  myRect.Left = 3;
  myRect.Top = 1;
  myRect.Right = 2;
  myRect.Bottom = 4;
  DrawGrid1->Selection = myRect;
  DrawGrid1->Repaint();
}

void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
  Label1->Caption = "From OnClick: The cursor is in column " +
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = (ARow * DrawGrid1->ColCount + ACol) % (DrawGrid1->ColCount + 1);
  if (State.Contains(gdSelected))
    DrawGrid1->Canvas->Brush->Color = clYellow;
  else
    DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
```

```
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(ARow * DrawGrid1->ColCount + ACol) +
      " has the focus.";
  }
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  myRect: TGridRect;
begin
  myRect.Left := 2;
  myRect.Top := 1;
  myRect.Right := 3;
  myRect.Bottom := 4;
  DrawGrid1.Selection := myRect;
  DrawGrid1.Repaint;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(ARow * DrawGrid1.ColCount + ACol) +
      ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
```

```
end;
```

See Also

Col (see page 3060)

Options (see page 3072)

Row (see page 3074)

10.1.2.3.11 TCustomDrawGrid.TabStops Property

Indicates whether the user can tab to specified columns in the grid.

Pascal

```
property TabStops [Index: Longint]: Boolean;
```

C++

```
__property Boolean TabStops[Longint Index];
```

Description

Set TabStops to false to remove the column identified by Index from the tab order. The first column in the grid is identified by an Index of 0.

Note: When the Options property does not include goTabs, the user can't tab to any column, even if TabStops is true.

See Also

Options (see page 3072)

10.1.2.3.12 TCustomDrawGrid.TopRow Property

Specifies the index of the first visible scrollable row in the grid.

Pascal

```
property TopRow: Longint;
```

C++

```
__property Longint TopRow;
```

Description

Read TopRow to determine the index of the first row in the scrollable region that is visible. Set TopRow to scroll the rows in the grid so that the row with index TopRow is the first row after the fixed rows.

C++ Examples:

```
*/
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DrawGrid1->TopRow = DrawGrid1->RowCount - 1;
}

void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
  Label1->Caption = "From OnClick: The cursor is in column " +
```

```
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = (ARow * DrawGrid1->ColCount + ACol) % (DrawGrid1->ColCount + 1);
  if (State.Contains(gdSelected))
    DrawGrid1->Canvas->Brush->Color = clYellow;
  else
    DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(ARow * DrawGrid1->ColCount + ACol) +
      " has the focus.";
  }
  if ((ARow >=  DrawGrid1->FixedRows) && (ACol >=  DrawGrid1->FixedCols))
    ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DrawGrid1.TopRow := DrawGrid1.RowCount - 1;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(ARow * DrawGrid1.ColCount + ACol) + '
has the focus.';
```

```
    end;
    if ((ARow >=  DrawGrid1.FixedRows) AND (ACol >=  DrawGrid1.FixedCols)) then
      ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedRows (see page 3068)

ScrollData (see page 3055)

VisibleRowCount (see page 3082)

10.1.2.4 TCustomDrawGrid Events

The events of the TCustomDrawGrid class are listed here.

Protected Events

	Name	Description
	OnColumnMoved (see page 3019)	Occurs immediately after the position of a column changes.
	OnDrawCell (see page 3020)	Occurs when a cell in the grid needs to be drawn.
	OnGetEditMask (see page 3021)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3022)	Occurs when the in-place editor requests the value of a cell.
	OnRowMoved (see page 3023)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3025)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3026)	Occurs when the user edits the value of a cell.
	OnTopLeftChanged (see page 3027)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

10.1.2.4.1 TCustomDrawGrid.OnColumnMoved Event

Occurs immediately after the position of a column changes.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Use OnColumnMoved to perform special processing when the position of a grid column changes. Columns can be moved programmatically or by user manipulation. OnColumnMoved does not occur unless the Options (see page 3072) property includes goColMoving.

See Also

OnRowMoved (see page 3023)

Options (see page 3072)

10.1.2.4.2 TCustomDrawGrid.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell: TDrawCellEvent;
```

C++

```
__property TDrawCellEvent OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas (see page 3007) property. The Rect parameter indicates the location of the cell on the canvas. The Col (see page 3009) and Row (see page 3013) parameters indicate the column and row indexes of the cell that should be drawn. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing (see page 3066) property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing (see page 3066) property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
```

```
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

Canvas

DefaultDrawing (see page 3066)

10.1.2.4.3 TCustomDrawGrid.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the cell specified by the ACol and ARow parameters. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options (see page 3072) property includes goEditing.

C++ Examples:

```
/*
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
}
```

```
void __fastcall TForm1::StringGrid1GetEditMask(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  if (StringGrid1->Cells[ACol][0] == "Phone Number")
    Value =  "!\(999\)000-0000;1";
}
```

Delphi Examples:

```
{
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[2, 0]:= 'Phone Number';
end;

procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  if StringGrid1.Cells[ACol, 0] = 'Phone Number' then
    Value :=  '!\(999\)000-0000;1';
end;
```

See Also

Options (see page 3072)

EditMask (see page 4604)

10.1.2.4.4 TCustomDrawGrid.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the cell specified by the ACol and ARow parameters. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell (see page 3020) event handler.

OnGetEditText does not occur unless the Options (see page 3072) property includes goEditing.

C++ Examples:

```
/*
The following code lowercases the text for a string grid
cell when the user begins to edit it.
*/
void __fastcall TForm1::StringGrid1GetEditText(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  Value = AnsiLowerCase(Value);
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code lowercases the text for a string grid
cell when the user begins to edit it.
}
procedure TForm1.StringGrid1GetEditText(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  Value := AnsiLowerCase(Value);
end;
```

See Also

OnDrawCell (see page 3020)

OnSetEditText (see page 3026)

10.1.2.4.5 TCustomDrawGrid.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. OnRowMoved does not occur unless the Options (see page 3072) property includes goRowMoving.

C++ Examples:

```
/*
The following code displays the number of rows a row was
moved in the first cell of the row.  To move rows in a
TStringGrid, goRowMoving must be included in the Options property.
*/
void __fastcall TForm1::StringGrid1RowMoved(TObject *Sender, int FromIndex, int ToIndex)

{
  int delta = (FromIndex > ToIndex) ? (FromIndex-ToIndex) : (ToIndex-FromIndex);
  StringGrid1->Cells[0][ToIndex] = IntToStr(delta);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code displays the number of rows a row was
moved in a label.  To move rows in a TStringGrid, goRowMoving
must be included in the Options property.
}
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
  Label1.Caption := IntToStr(Abs(FromIndex-ToIndex));
end;
```

See Also

OnColumnMoved (see page 3019)

Options (see page 3072)

10.1.2.4.6 TCustomDrawGrid.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell: TSelectCellEvent;
```

C++

```
__property TSelectCellEvent OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected. The Col (see page 3009) and Row (see page 3013) parameters indicate the column and row indexes of the cell that is about to be selected. Set the CanSelect parameter to false to prevent the cell being selected.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code ensures that the user cannot select a cell
unless it is empty.
*/
void __fastcall TForm1::StringGrid1SelectCell(TObject *Sender, int ACol, int ARow,
          bool &CanSelect)
{
  CanSelect = (StringGrid1->Cells[ACol][ARow]== "");
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code ensures that the user cannot select a cell
unless it is empty.
}
procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Longint; var CanSelect:
Boolean);
begin
  CanSelect := (StringGrid1.Cells[ACol, ARow]='')
end;
```

See Also

Col (see page 3060)

Row (see page 3074)

TabStops (see page 3079)

10.1.2.4.7 TCustomDrawGrid.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText: TSetEditEvent;
```

C++

```
__property TSetEditEvent OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell (see page 3020) event handler. OnSetEditText occurs every time the user changes the text.

The Value parameter is the new value for the cell specified by the ACol and ARow parameters.

OnSetEditText does not occur unless the Options (see page 3072) property includes goEditing.

See Also

OnDrawCell (see page 3020)

OnGetEditText (see page 3022)

Options (see page 3072)

10.1.2.4.8 TCustomDrawGrid.OnTopLeftChanged Event

Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

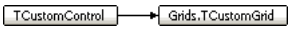
LeftCol (see page 3072)

TopRow (see page 3080)

10.1.3 TCustomGrid Class

TCustomGrid is the base type for all components that present information in a two-dimensional grid.

Class Hierarchy



Pascal

```
TCustomGrid = class(TCustomControl);
```

C++

```
class TCustomGrid : public TCustomControl;
```

File

Grids

Description

Use TCustomGrid as a base class when defining components that represent information in a tabular format. TCustomGrid introduces properties and methods to control the appearance and behavior of the grid, including responses to mouse actions and user editing.

Do not create instances of TCustomGrid. Instead, put a grid control in a form by instantiating a descendant of TCustomGrid. Use TDrawGrid (see page 3083) to represent non-textual material in a tabular grid. For text, use a TStringGrid (see page 3180) object. If the text represents field values from a dataset, use TDBGrid.

See Also

TDBGrid

TDrawGrid (see page 3083)

TStringGrid (see page 3180)

10.1.3.1 TCustomGrid Members

The following tables list the members exposed by TCustomGrid.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	ColumnMoved (see page 3039)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
V	CreateEditor (see page 3040)	Creates the inplace editor control.
V	CreateParams (see page 3041)	Initializes the window-creation parameter record.
V	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.
V	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
V	DeleteRow (see page 3042)	Deletes a specified from the grid.
V	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
V	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
A	DrawCell (see page 3043)	Provides the prototype for a method that draws the contents of a cell in the grid.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.

	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 3046)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	KeyDown (see page 3050)	Provides special processing when the user presses a key.
	KeyPress (see page 3050)	Provides special processing when the user presses a key.
	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3055)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 3056)	Provides the interface for a method that updates the text associated with a cell.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	BorderStyle (see page 3060)	Determines whether a single line border is drawn around the grid.
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	Color (see page 3063)	Specifies the background color of the control.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.

	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Options (see page 3072)	Specifies various display and behavioral properties of the grid.
	ParentColor (see page 3073)	Determines where a control looks for its color information.
	Row (see page 3074)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	ScrollBars (see page 3077)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Public Methods

	Name	Description
V	Create (see page 3030)	Creates an instance of TCustomGrid (see page 3027).
V	Destroy (see page 3031)	Destroys an instance of TCustomGrid (see page 3027).
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

Published Properties

	Name	Description
	TabStop (see page 3079)	Determines if the user can tab to a control.

10.1.3.2 TCustomGrid.Create Constructor

Creates an instance of TCustomGrid (see page 3027).

Pascal

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

C++

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

Description

Call Create to create a grid control at runtime. Grid controls placed on a form at design time are created automatically.

Owner is a component that is responsible for freeing the grid and for initiating the streaming process when it is saved to a form file.

See Also

Owner

10.1.3.3 TCustomGrid.Destroy Destructor

Destroys an instance of TCustomGrid (see page 3027).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomGrid();
```

Description

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

See Also

System::TObject::Free

10.1.3.4 TCustomGrid Fields

The fields of the TCustomGrid class are listed here.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

10.1.3.4.1 TCustomGrid.DesignOptionsBoost Field

Pascal

```
DesignOptionsBoost: TGridOptions;
```

C++

```
TGridOptions DesignOptionsBoost;
```

Description

This is DesignOptionsBoost, a member of class TCustomGrid.

10.1.3.4.2 TCustomGrid.FGridState Field

Pascal

```
FGridState: TGridState;
```

C++

```
TGridState FGridState;
```

Description

This is FGridState, a member of class TCustomGrid.

10.1.3.4.3 TCustomGrid.FSaveCellExtents Field

Pascal

```
FSaveCellExtents: Boolean;
```

C++

```
Boolean FSaveCellExtents;
```

Description

This is FSaveCellExtents, a member of class TCustomGrid.

10.1.3.4.4 TCustomGrid.VirtualView Field

Pascal

```
VirtualView: Boolean;
```

C++

```
Boolean VirtualView;
```

Description

This is VirtualView, a member of class TCustomGrid.

10.1.3.5 TCustomGrid Methods

The methods of the TCustomGrid class are listed here.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	ColumnMoved (see page 3039)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
V	CreateEditor (see page 3040)	Creates the inplace editor control.
V	CreateParams (see page 3041)	Initializes the window-creation parameter record.
V	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.

V	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
V	DeleteRow (see page 3042)	Deletes a specified from the grid.
V	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
V	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
A	DrawCell (see page 3043)	Provides the prototype for a method that draws the contents of a cell in the grid.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 3046)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
V	KeyDown (see page 3050)	Provides special processing when the user presses a key.
V	KeyPress (see page 3050)	Provides special processing when the user presses a key.
V	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.
V	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
V	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3055)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 3056)	Provides the interface for a method that updates the text associated with a cell.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Public Methods

	Name	Description
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

10.1.3.5.1 TCustomGrid.AdjustSize Method

Adds or deletes rows or columns from the grid.

Pascal

```
procedure AdjustSize(Index: Longint; Amount: Longint; Rows: Boolean); dynamic; reintroduce;
```

C++

```
__fastcall AdjustSize(Longint Index, Longint Amount, Boolean Rows);
```

Description

Grid components call AdjustSize internally to fix up the number of rows or columns when the user resizes them using the mouse. Index is the index of the row or column where the insertion or deletion should begin. Amount is the number of rows or columns to insert (Amount > 0) or delete (Amount < 0). Rows is a boolean that indicates whether the rows or the columns are changing.

See Also

DefaultColWidth (see page 3064)

DefaultRowHeight (see page 3066)

10.1.3.5.2 TCustomGrid.BeginColumnDrag Method

Starts the dragging of a column in the grid.

Pascal

```
function BeginColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean BeginColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

BeginColumnDrag is called automatically when the user clicks on a column to start dragging. It enables the grid to control whether the column can be repositioned and if so, which column.

Origin is the index of the column to be moved. When BeginColumnDrag is called, this is the index of the column in which the mouse was clicked. TCustomGrid (see page 3027) does not change this value. However, descendant classes can override this method to adjust which column can be moved.

Destination is the index of where the column should move. When BeginColumnDrag is called, this is the same as Origin. TCustomGrid (see page 3027) does not change this value because the mouse has not yet moved. Descendant classes can override this method to adjust the default destination of the column.

MousePt is the coordinates of the mouse button when the user clicked.

BeginColumnDrag returns true if the drag operation is permitted, false if the drag should be disallowed.

See Also

BeginRowDrag (see page 3034)

EndColumnDrag (see page 3044)

10.1.3.5.3 TCustomGrid.BeginRowDrag Method

Starts the dragging of a row in the grid.

Pascal

```
function BeginRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean BeginRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call BeginRowDrag to start a drag operation. BeginRowDrag enables the grid to control where the row can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 3027) can customize row dragging in a grid.

See Also

BeginColumnDrag (see page 3034)

EndRowDrag (see page 3044)

10.1.3.5.4 TCustomGrid.BoxRect Method

Returns the screen coordinates of a specified set of cells.

Pascal

```
function BoxRect(ALeft: Longint; ATop: Longint; ARight: Longint; ABottom: Longint): TRect;
```

C++

```
__fastcall TRect BoxRect(Longint ALeft, Longint ATop, Longint ARight, Longint ABottom);
```

Description

Call BoxRect to obtain the screen coordinates, in pixels, of the cells that run from the column with index ALeft to the column with index ARight and the row with index ATop to the row with index ABottom. The coordinates of cells that are not visible onscreen are not included in the final rectangle.

See Also

BoundsRect

CellRect (see page 3038)

10.1.3.5.5 TCustomGrid.CalcDrawInfo Method

Calculates the pixel locations of the grid boundaries.

Pascal

```
procedure CalcDrawInfo(var DrawInfo: TGridDrawInfo);
```

C++

```
__fastcall CalcDrawInfo(TGridDrawInfo DrawInfo);
```

Description

Use CalcDrawInfo to determine the pixel locations of the major grid boundaries in the X and Y directions, including things like the first visible column/row, the last visible column/row, the pixel extent of the fixed column/row, etc.

See Also

CalcFixedInfo (see page 3036)

10.1.3.5.6 TCustomGrid.CalcFixedInfo Method

Calculates the pixel locations of the fixed cells.

Pascal

```
procedure CalcFixedInfo(var DrawInfo: TGridDrawInfo);
```

C++

```
__fastcall CalcFixedInfo(TGridDrawInfo DrawInfo);
```

Description

Use CalcDrawInfo (see page 3035) to determine the pixel locations of the fixed cells in the X and Y directions.

See Also

CalcDrawInfo (see page 3035)

10.1.3.5.7 TCustomGrid.CalcSizingState Method

Determines the appearance of the mouse cursor during resizing.

Pascal

```
procedure CalcSizingState(X: Integer; Y: Integer; var State: TGridState; var Index: Longint;
var SizingPos: Integer; var SizingOfs: Integer; var FixedInfo: TGridDrawInfo); virtual;
```

C++

```
virtual __fastcall CalcSizingState(int X, int Y, TGridState State, Longint Index, int
SizingPos, int SizingOfs, TGridDrawInfo FixedInfo);
```

Description

Use CalcSizingState to determine if the mouse cursor should be changed to a resize or drag cursor based on where the mouse is in the grid. For example, cursor over a grid line, in the fixed cell region, and column resize is enabled = resize cursor.

See Also

CalcFixedInfo (see page 3036)

10.1.3.5.8 TCustomGrid.CanEditAcceptKey Method

Indicates whether the Key parameter can be entered into the current cell.

Pascal

```
function CanEditAcceptKey(Key: Char): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEditAcceptKey(Char Key);
```

Description

CanEditAcceptKey provides a simple screening of keyboard input while the user is editing the contents of a cell. The CanEditAcceptKey method for TCustomGrid (see page 3027) always returns true. Descendants of TCustomGrid (see page 3027) override this method to screen out illegal characters.

More refined validation can be obtained by using an edit mask.

See Also

CanGridAcceptKey (see page 3037)

GetEditMask (see page 3045)

KeyPress (see page 3050)

10.1.3.5.9 TCustomGrid.CanEditModify Method

Indicates whether the current cell in the grid can be edited.

Pascal

```
function CanEditModify: Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEditModify();
```

Description

Grid components use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify returns true unless the inplace editor has disallowed editing for the cell.

See Also

CanEditShow (see page 3037)

InplaceEditor (see page 3071)

10.1.3.5.10 TCustomGrid.CanEditShow Method

Indicates whether the inplace edit control can be created to allow editing.

Pascal

```
function CanEditShow: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanEditShow();
```

Description

Grid controls call CanEditShow internally before creating the editor for a cell. CanEditShow is called after the grid has determined that the field can be edited by calling CanEditModify (see page 3037).

CanEditShow returns false if the Options (see page 3072) property does not indicate that the grid allows editing, or if it indicates that the grid permits row selection. Otherwise, if the Options (see page 3072) property indicates that the grid is locked in edit mode, or if the grid has input focus, CanEditShow returns true. If the grid is not locked in edit mode and the grid does not have input focus, CanEditShow returns false.

See Also

CanEditModify (see page 3037)

InplaceEditor (see page 3071)

Options (see page 3072)

10.1.3.5.11 TCustomGrid.CanGridAcceptKey Method

Indicates whether the grid should respond to a given key combination.

Pascal

```
function CanGridAcceptKey(Key: Word; Shift: TShiftState): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanGridAcceptKey(Word Key, TShiftState Shift);
```

Description

Grid controls call CanGridAcceptKey internally to screen key combinations when the user first presses a key. The CanGridAcceptKey method for TCustomGrid (see page 3027) always returns true. Descendants of TCustomGrid (see page 3027) override this method to screen out irrelevant key combinations, or key combinations that indicate actions that would violate the Options (see page 3072) or mode of the grid.

See Also

CanEditAcceptKey (see page 3036)

KeyDown (see page 3050)

Options (see page 3072)

10.1.3.5.12 TCustomGrid.CellRect Method

Returns the screen coordinates of a cell in the grid.

Pascal

```
function CellRect(ACol: Longint; ARow: Longint): TRect;
```

C++

```
__fastcall TRect CellRect(Longint ACol, Longint ARow);
```

Description

Call CellRect to obtain the boundaries, in pixels, of the cell in the column and row indicated by the ACol and ARow parameters. The top left cell is specified when ACol and ARow are set to zero.

If the indicated cell is not visible, CellRect returns an empty rectangle.

See Also

BoxRect (see page 3035)

10.1.3.5.13 TCustomGrid.ChangeGridOrientation Method

Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.

Pascal

```
procedure ChangeGridOrientation(RightToLeftOrientation: Boolean);
```

C++

```
__fastcall ChangeGridOrientation(Boolean RightToLeftOrientation);
```

Description

TCustomGrid (see page 3027) uses ChangeGridOrientation internally when painting. ChangeGridOrientation sets the origin and dimensions of the grid's canvas so that painting occurs from right-to-left or from left-to-right based on the value of the RightToLeftOrientation parameter.

RightToLeftOrientation is true if the grid should prepare for painting from right to left. RightToLeftOrientation is false if the grid should restore the canvas to normal.

See Also

UseRightToLeftAlignment

BiDiMode

10.1.3.5.14 TCustomGrid.CheckColumnDrag Method

CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.

Pascal

```
function CheckColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Use CheckColumnDrag to validate that the column currently selected for dragging can be dropped at the current location. CheckColumnDrag can change the Origin (the column being dragged) and/or the Destination (the drop target), as needed, to keep the grid's columns in order.

See Also

BeginColumnDrag (see page 3034)

EndColumnDrag (see page 3044)

10.1.3.5.15 TCustomGrid.CheckRowDrag Method

CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.

Pascal

```
function CheckRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Use CheckRowDrag to validate that the row currently selected for dragging can be dropped at the current location. CheckRowDrag can change the Origin (the row being dragged) and/or the Destination (the drop target), as needed, to keep the grid's rows in order.

See Also

BeginRowDrag (see page 3034)

EndRowDrag (see page 3044)

10.1.3.5.16 TCustomGrid.ColumnMoved Method

Provides the interface for a method that responds when the position of a column changes.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); dynamic;
```

C++

```
__fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. The FromIndex parameter is the old index of the column, and the ToIndex parameter is the new index. The ColumnMoved method of TCustomGrid (see page 3027) does nothing. Descendants of TCustomGrid (see page 3027) override this method to make internal adjustments or to generate an OnColumnMoved event.

See Also

MoveColumn (see page 3053)

10.1.3.5.17 TCustomGrid.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; dynamic;
```

C++

```
__fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change. The change can result from setting the ColWidths (see page 3063) property, from setting the DefaultColWidth (see page 3064) property, from moving one of the columns, or from resizing the columns with the mouse.

ColWidthsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new column size and positions of the columns.

See Also

ColWidths (see page 3063)

DefaultColWidth (see page 3064)

InplaceEditor (see page 3071)

ScrollBars (see page 3077)

10.1.3.5.18 TCustomGrid.CreateEditor Method

Creates the inplace editor control.

Pascal

```
function CreateEditor: TInplaceEdit; virtual;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

Grid controls call CreateEditor internally the first time the grid goes into edit mode. CreateEditor is exposed as a protected method so that descendants of TCustomGrid (see page 3027) can substitute a specialized descendant of TInplaceEdit (see page 3161) to handle cell editing.

See Also

EditorMode (see page 3066)

InplaceEditor (see page 3071)

ShowEditor (see page 3056)

10.1.3.5.19 TCustomGrid.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method is called internally to specify the properties of the window that implements this control. These properties are set in the window-creation parameter record passed as Params. The fields of this record become the parameters to a call to the CreateWindowEx API function.

After calling the inherited method, CreateParams adjusts the value of Params to request that the grid receive double click messages, and to implement the values of the ScrollBars (see page 3077) and BorderStyle (see page 3060) properties.

See Also

BorderStyle (see page 3060)

ScrollBars (see page 3077)

CreateWnd

10.1.3.5.20 TCustomGrid.DefineProperties Method

Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties overrides the inherited method to save the ColWidths (see page 3063) and RowHeights (see page 3076) properties with the grid when they contain values other than the default widths and heights.

See Also

ColWidths (see page 3063)

DefaultColWidth (see page 3064)

DefaultRowHeight (see page 3066)

RowHeights (see page 3076)

TFiler

10.1.3.5.21 TCustomGrid.DeleteColumn Method

Deletes a specified column from the grid.

Pascal

```
procedure DeleteColumn(ACol: Longint); virtual;
```

C++

```
virtual __fastcall DeleteColumn(Longint ACol);
```

Description

Call DeleteColumn to remove a column from the middle of the grid. Provide the index of the column to remove as ACol. To remove the last column from the grid, set the ColCount (see page 3062) property instead.

See Also

ColCount (see page 3062)

10.1.3.5.22 TCustomGrid.DeleteRow Method

Deletes a specified from the grid.

Pascal

```
procedure DeleteRow(ARow: Longint); virtual;
```

C++

```
virtual __fastcall DeleteRow(Longint ARow);
```

Description

Call DeleteRow to remove a row from the middle of the grid. Provide the index of the row to remove as ARow. To remove the last row from the grid, set the RowCount (see page 3075) property instead.

See Also

RowCount (see page 3075)

10.1.3.5.23 TCustomGrid.DoExit Method

Hides the inplace editor when selection moves to another control in a form.

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

After calling the inherited method, which generates an OnExit event, DoExit hides the inplace editor unless the Options (see page 3072) property includes goAlwaysShowEditor.

See Also

HideEditor (see page 3048)

Options (see page 3072)

OnExit

10.1.3.5.24 TCustomGrid.DoMouseWheelDown Method

Performs tasks when the mouse wheel is rotated downward.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelDown to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelDown returns false if the control's parent should handle the message.

In TCustomGrid (see page 3027), DoMouseWheelDown increments the current row and returns true.

See Also

DoMouseWheelUp (see page 3043)

Row (see page 3074)

10.1.3.5.25 TCustomGrid.DoMouseWheelUp Method

Performs tasks when the mouse wheel is rotated upward.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelUp to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelUp should return false if the control's parent should handle the message.

In TCustomGrid (see page 3027), DoMouseWheelUp decrements the current row and returns true.

See Also

DoMouseWheelDown (see page 3042)

Row (see page 3074)

10.1.3.5.26 TCustomGrid.DrawCell Method

Provides the prototype for a method that draws the contents of a cell in the grid.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
virtual; abstract;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState) =
0;
```

Description

Descendants of TCustomGrid (see page 3027) implement the DrawCell method to draw the contents of the cell or to generate an OnDrawCell event. If the DefaultDrawing (see page 3066) property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell.

The ACol and ARow parameters give the column and row indexes of the cell that should be drawn. The ARect parameter gives the boundaries of the cell in the coordinates of the canvas. The AState parameter indicates whether the cell is selected, whether the cell has input focus, and whether the cell is in the fixed, non-scrolling portion of the grid.

See Also

DefaultDrawing (see page 3066)

FixedCols (see page 3067)

FixedRows (see page 3068)

Options (see page 3072)

10.1.3.5.27 TCustomGrid.EndColumnDrag Method

Ends the dragging of a column in the grid.

Pascal

```
function EndColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean EndColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call EndColumnDrag to end a drag operation. EndColumnDrag enables the grid to control where the column can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 3027) can customize column dragging in a grid.

See Also

BeginColumnDrag (see page 3034)

10.1.3.5.28 TCustomGrid.EndRowDrag Method

Ends the dragging of a row in the grid.

Pascal

```
function EndRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean EndRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call EndRowDrag to end a drag operation. EndRowDrag enables the grid to control where the row can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 3027) can customize row dragging in a grid.

See Also

BeginRowDrag (see page 3034)

10.1.3.5.29 TCustomGrid.FocusCell Method

Changes the current cell to a specified cell.

Pascal

```
procedure FocusCell(ACol: Longint; ARow: Longint; MoveAnchor: Boolean);
```

C++

```
__fastcall FocusCell(Longint ACol, Longint ARow, Boolean MoveAnchor);
```

Description

TCustomGrid (see page 3027) calls FocusCell internally to change the currently selected cell. FocusCell changes the current cell and updates the selection appropriately.

ACol is the column of the cell to select.

ARow is the row of the cell to select.

MoveAnchor determines how the selection changes. If the grid does not allow multiple cells to be selected (Options (see page 3072) does not include goRangeSelect), MoveAnchor is ignored and the specified cell becomes the selected cell. If the grid allows multiple cells to be selected, MoveAnchor determines whether the new cell becomes the starting cell (anchor) of a new selection, or whether the current selection is updated to range from the current anchor to the specified cell.

See Also

Col (see page 3060)

Row (see page 3074)

Selection (see page 3077)

Options (see page 3072)

10.1.3.5.30 TCustomGrid.GetEditLimit Method

Provides the interface for a method that limits the number of characters the user can type into cells.

Pascal

```
function GetEditLimit: Integer; dynamic;
```

C++

```
__fastcall int GetEditLimit();
```

Description

The inplace editor calls GetEditLimit to determine if a limit should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits. Descendants of TCustomGrid (see page 3027) override this method to limit the text for certain cells.

See Also

InplaceEditor (see page 3071)

10.1.3.5.31 TCustomGrid.GetEditMask Method

Provides the interface for a method that returns an input mask for the specified cell.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; dynamic;
```

C++

```
__fastcall UnicodeString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns an empty string, indicating that the cell has no input mask. Descendants of TCustomGrid (see page 3027) override this method to provide an input mask or generate an OnGetEditMask event.

See Also

InplaceEditor (see page 3071)

EditMask (see page 4604)

10.1.3.5.32 TCustomGrid.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; dynamic;
```

C++

```
__fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomGrid (see page 3027), GetEditStyle always returns esSimple, indicating that there is no ellipsis button or drop-down arrow. Descendant classes can override this methods if they provide a way to add pick lists or ellipsis buttons to cells.

10.1.3.5.33 TCustomGrid.GetEditText Method

Returns the value of the indicated cell formatted for editing.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; dynamic;
```

C++

```
__fastcall UnicodeString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns an empty string. Descendants of TCustomGrid (see page 3027) override the GetEditText method to return a string that represents the contents of the cell indicated by the ACol and ARow parameters.

See Also

InplaceEditor (see page 3071)

10.1.3.5.34 TCustomGrid.GetGridHeight Method

Returns the value of the GridHeight (see page 3068) property.

Pascal

```
function GetGridHeight: Integer;
```

C++

```
__fastcall int GetGridHeight();
```

Description

GetGridHeight is the protected property access method for the GridHeight (see page 3068) property. It computes the value of the GridHeight (see page 3068) property based on the row heights, separator width, and the size of the control.

See Also

ClientHeight

GetGridWidth (see page 3047)

GridHeight (see page 3068)

GridLineWidth (see page 3069)

RowHeights (see page 3076)

10.1.3.5.35 TCustomGrid.GetGridWidth Method

Returns the value of the GridWidth (see page 3070) property.

Pascal

```
function GetGridWidth: Integer;
```

C++

```
__fastcall int GetGridWidth();
```

Description

GetGridWidth is the protected property access method for the GridWidth (see page 3070) property. It computes the value of the GridWidth (see page 3070) property based on the column widths, separator width, and the size of the control.

See Also

ClientWidth

ColWidths (see page 3063)

GetGridHeight (see page 3046)

GridWidth (see page 3070)

GridLineWidth (see page 3069)

10.1.3.5.36 TCustomGrid.HideEdit Method

Hides the inplace editor.

Pascal

```
procedure HideEdit;
```

C++

```
__fastcall HideEdit();
```

Description

The HideEditor (see page 3048) method calls the protected HideEdit method to hide the editor. This method hides the editor window after updating the text to reflect the text in the editor.

See Also

HideEditor (see page 3048)

InplaceEditor (see page 3071)

10.1.3.5.37 TCustomGrid.HideEditor Method

Hides the inplace editor when it is not in use.

Pascal

```
procedure HideEditor;
```

C++

```
__fastcall HideEditor();
```

Description

Grid controls call HideEditor when the grid goes out of edit mode. HideEditor hides the window of the inplace editor.

See Also

EditorMode (see page 3066)

InplaceEditor (see page 3071)

ShowEditor (see page 3056)

10.1.3.5.38 TCustomGrid.InvalidateCell Method

Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateCell(ACol: Longint; ARow: Longint);
```

C++

```
__fastcall InvalidateCell(Longint ACol, Longint ARow);
```

Description

Call InvalidateCell when the contents of a cell change so that the cell will need to be repainted. The ACol parameter is the column index of the cell, and the ARow parameter is the row index.

See Also

CellRect (see page 3038)

InvalidateCol (see page 3048)

InvalidateEditor (see page 3049)

InvalidateRow (see page 3050)

10.1.3.5.39 TCustomGrid.InvalidateCol Method

Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateCol(ACol: Longint);
```

C++

```
__fastcall InvalidateCol(Longint ACol);
```

Description

Call InvalidateCol when a column changes so that the area it occupies will need to be repainted. The ACol parameter is the column index.

See Also

BoxRect (see page 3035)

InvalidateCell (see page 3048)

InvalidateRow (see page 3050)

10.1.3.5.40 TCustomGrid.InvalidateEditor Method

Causes the inplace editor to repaint itself, moving to a new location if necessary.

Pascal

```
procedure InvalidateEditor;
```

C++

```
__fastcall InvalidateEditor();
```

Description

InvalidateEditor is called when the selected cell changes while the grid is in edit mode.

See Also

EditorMode (see page 3066)

HideEditor (see page 3048)

InplaceEditor (see page 3071)

InvalidateCell (see page 3048)

ShowEditor (see page 3056)

10.1.3.5.41 TCustomGrid.InvalidateGrid Method

Tells Windows to repaint the grid after other important Windows messages are handled.

Pascal

```
procedure InvalidateGrid;
```

C++

```
__fastcall InvalidateGrid();
```

Description

InvalidateGrid is used internally to ensure that the grid is repainted when its properties change. It calls the Invalidate method to indicate that the grid needs repainting.

See Also

Invalidate

10.1.3.5.42 TCustomGrid.InvalidateRow Method

Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateRow(ARow: Longint);
```

C++

```
__fastcall InvalidateRow(Longint ARow);
```

Description

Call InvalidateRow when a row changes so that the area it occupies will need to be repainted. The ARow parameter is the row index.

See Also

BoxRect (see page 3035)

InvalidateCell (see page 3048)

InvalidateCol (see page 3048)

10.1.3.5.43 TCustomGrid.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After generating an OnKeyDown event, KeyDown calls CanGridAcceptKey (see page 3037) to screen out problematic key combinations. KeyDown moves to the appropriate cell in response to navigational key combinations, or sets the EditorMode (see page 3066) property to true when the user presses F2.

See Also

CanGridAcceptKey (see page 3037)

EditorMode (see page 3066)

KeyPress (see page 3050)

OnKeyDown

10.1.3.5.44 TCustomGrid.KeyPress Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically after the KeyDown (see page 3050) method, but before the user releases the key.

After generating an OnKeyPress event, KeyPress toggles the EditorMode (see page 3066) property when the user presses the Enter key.

See Also

EditorMode (see page 3066)

KeyDown (see page 3050)

OnKeyPress

10.1.3.5.45 TCustomGrid.MouseCoord Method

Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

Pascal

```
function MouseCoord(X: Integer; Y: Integer): TGridCoord;
```

C++

```
__fastcall TGridCoord MouseCoord(int X, int Y);
```

Description

Call MouseCoord to locate the column and row of the cell which contains a given screen coordinate. Usually, MouseCoord is used to locate the cell that is under the mouse.

See Also

CellRect (see page 3038)

MouseDown (see page 3051)

MouseMove (see page 3052)

MouseUp (see page 3052)

10.1.3.5.46 TCustomGrid.MouseDown Method

Provides special processing when the user presses a mouse button.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown hides the editor window so that it can move selection to the cell that receives the mouse click. If the mouse lands in the hot region for beginning a move or resize of a row or column, MouseDown starts the move or resize, and provides visual feedback by changing the cursor. If the editor window was hidden, it is brought back up in its new location. Finally, MouseDown generates an OnMouseDown event.

See Also

HideEditor (see page 3048)

InplaceEditor (see page 3071)

MouseMove (see page 3052)

MouseUp (see page 3052)

ShowEditor (see page 3056)

OnMouseDown

10.1.3.5.47 TCustomGrid.MouseMove Method

Provides special processing when the user moves the mouse.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

If the grid is in the middle of moving or resizing a row or column, or in the middle of selecting a range of cells using the mouse, MouseMove calculates the changes implied by the mouse move and provides visual feedback to the user. Finally, MouseMove generates an OnMouseMove event.

See Also

MouseDown (see page 3051)

MouseUp (see page 3052)

OnMouseMove

10.1.3.5.48 TCustomGrid.MouseUp Method

Provides special processing when the user releases the mouse button.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp finishes a move or resize operation on a row or column, or if the user is selecting a range of cells using the mouse, MouseUp finishes the select operation. Finally, MouseUp generates an OnMouseUp event.

See Also

MouseDown (see page 3051)

MouseMove (see page 3052)

MoveColumn (see page 3053)

MoveRow (see page 3053)

OnMouseUp

10.1.3.5.49 TCustomGrid.MoveColRow Method

Selects the cell indicated by the ACol and ARow parameters.

Pascal

```
procedure MoveColRow(ACol: Longint; ARow: Longint; MoveAnchor: Boolean; Show: Boolean);
```

C++

```
__fastcall MoveColRow(Longint ACol, Longint ARow, Boolean MoveAnchor, Boolean Show);
```

Description

Grid controls call MoveColRow internally when changing the selected cell or cells. MoveColRow selects the indicated cell, but does not take care of removing any previous selection.

The ACol and ARow parameters are the column index and the row index of the cell to be selected. The MoveAnchor parameter indicates whether the new cell becomes the upper left corner of the selected region. MoveAnchor is true unless MoveColRow is called when extending a multicell selected region. The Show parameter indicates whether MoveColRow should ensure that the newly selected cell is visible.

See Also

Col (see page 3060)

Row (see page 3074)

SelectCell (see page 3055)

Selection (see page 3077)

10.1.3.5.50 TCustomGrid.MoveColumn Method

Moves a column to a new position.

Pascal

```
procedure MoveColumn(FromIndex: Longint; ToIndex: Longint);
```

C++

```
__fastcall MoveColumn(Longint FromIndex, Longint ToIndex);
```

Description

Call MoveColumn to move the column identified by the FromIndex parameter to the position specified by the ToIndex parameter.

See Also

ColumnMoved (see page 3039)

ColWidthsChanged (see page 3040)

10.1.3.5.51 TCustomGrid.MoveRow Method

Moves a row to a new position.

Pascal

```
procedure MoveRow(FromIndex: Longint; ToIndex: Longint);
```

C++

```
__fastcall MoveRow(Longint FromIndex, Longint ToIndex);
```

Description

Call MoveRow to move the row identified by the FromIndex parameter to the position specified by the ToIndex parameter.

See Also

RowHeightsChanged (see page 3054)

RowMoved (see page 3055)

10.1.3.5.52 TCustomGrid.Paint Method

Draws the image of the grid control on the screen.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called when the grid receives a paint message. Paint draws the grid, calling the DrawCell (see page 3043) method when it is time to paint the contents of each cell. If DefaultDrawing (see page 3066) is false, Paint does not draw the background of the cells, provide the raised 3D effects on the fixed cells, or draw the focus rectangle around the selected cell when it has focus.

See Also

DefaultDrawing (see page 3066)

DrawCell (see page 3043)

InvalidateCell (see page 3048)

InvalidateCol (see page 3048)

InvalidateRow (see page 3050)

10.1.3.5.53 TCustomGrid.RowHeightsChanged Method

Responds when the row heights change.

Pascal

```
procedure RowHeightsChanged; dynamic;
```

C++

```
__fastcall RowHeightsChanged();
```

Description

RowHeightsChanged is called immediately after the row heights change. The change can result from setting the RowHeights (see page 3076) property, from setting the DefaultRowHeight (see page 3066) property, from moving one of the rows, or from resizing the rows with the mouse.

RowHeightsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new row size and positions of the rows.

See Also

DefaultRowHeight (see page 3066)

InplaceEditor (see page 3071)

RowHeights (see page 3076)

ScrollBars (see page 3077)

10.1.3.5.54 TCustomGrid.RowMoved Method

Provides the interface for a method that responds when the position of a row changes.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); dynamic;
```

C++

```
__fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. The FromIndex parameter is the old index of the row, and the ToIndex parameter is the new index. The RowMoved method of TCustomGrid (see page 3027) does nothing. Descendants of TCustomGrid (see page 3027) override this method to make internal adjustments or to generate an OnRowMoved event.

See Also

MoveRow (see page 3053)

10.1.3.5.55 TCustomGrid.ScrollData Method

Scrolls the visual image of the cells in the grid.

Pascal

```
procedure ScrollData(DX: Integer; DY: Integer);
```

C++

```
__fastcall ScrollData(int DX, int DY);
```

Description

Grid controls call ScrollData internally when the nonfixed rows or columns scroll. ScrollData updates the image of the grid.

See Also

LeftCol (see page 3072)

TopRow (see page 3080)

10.1.3.5.56 TCustomGrid.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

The SelectCell method for TCustomGrid (see page 3027) always returns true, allowing the cell to be selected. Descendants of TCustomGrid (see page 3027) override this method to check the properties of the information displayed in the cell or generating an OnSelectCell event.

See Also

MoveColRow (see page 3053)

TabStops (see page 3079)

10.1.3.5.57 TCustomGrid.SetEditText Method

Provides the interface for a method that updates the text associated with a cell.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); dynamic;
```

C++

```
__fastcall SetEditText(Longint ACol, Longint ARow, const UnicodeString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. The SetEditText method for TCustomGrid (see page 3027) does nothing. Descendants of TCustomGrid (see page 3027) override this method to update an internal representation of the cell's value, or to generate an OnSetEditText event.

See Also

GetEditText (see page 3046)

10.1.3.5.58 TCustomGrid.ShowEditor Method

Displays the inplace editor when the grid enters edit mode.

Pascal

```
procedure ShowEditor;
```

C++

```
__fastcall ShowEditor();
```

Description

Grid controls call ShowEditor when the grid enters edit mode. ShowEditor displays the window of the inplace editor over the currently selected cell. If the inplace editor does not yet exist, ShowEditor creates it.

See Also

CreateEditor (see page 3040)

EditorMode (see page 3066)

HideEditor (see page 3048)

InplaceEditor (see page 3071)

10.1.3.5.59 TCustomGrid.ShowEditorChar Method

Displays the inplace editor, with its text set to the specified character.

Pascal

```
procedure ShowEditorChar(Ch: Char);
```

C++

```
__fastcall ShowEditorChar(Char Ch);
```

Description

Grid controls call ShowEditorChar when the user types a character into a cell and the editor is not already showing. ShowEditorChar displays the window of the inplace editor over the currently selected cell, and sets its text property to the character Ch. If the inplace editor does not yet exist, ShowEditorChar creates it.

See Also

CreateEditor (see page 3040)

EditorMode (see page 3066)

HideEditor (see page 3048)

InplaceEditor (see page 3071)

ShowEditor (see page 3056)

10.1.3.5.60 TCustomGrid.SizeChanged Method

Responds to changes in the size of the grid.

Pascal

```
procedure SizeChanged(OldColCount: Longint; OldRowCount: Longint); dynamic;
```

C++

```
__fastcall SizeChanged(Longint OldColCount, Longint OldRowCount);
```

Description

SizeChanged provides the interface for a method that responds to changes in the number of rows or columns in the grid. The SizeChanged method of TCustomGrid (see page 3027) does nothing. Descendants of TCustomGrid (see page 3027) override this method to make internal adjustments or generate events.

See Also

ColCount (see page 3062)

RowCount (see page 3075)

10.1.3.5.61 TCustomGrid.Sizing Method

Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.

Pascal

```
function Sizing(X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean Sizing(int X, int Y);
```

Description

Grid controls call Sizing internally with the coordinates of a mouse click to determine whether the user is about to resize a row or column.

See Also

MouseDown (see page 3051)

10.1.3.5.62 TCustomGrid.TimedScroll Method

Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.

Pascal

```
procedure TimedScroll(Direction: TGridScrollDirection); dynamic;
```

C++

```
__fastcall TimedScroll(TGridScrollDirection Direction);
```

Description

TimedScroll is called periodically as the user continues to drag with the mouse outside of the scrollable cells in the grid. TimedScroll does nothing unless the user is moving a row or column, resizing a row or column, or selecting a multiple cell region using the mouse. TimedScroll scrolls the cells in the grid to allow the operation to continue into the portion of the grid that was hidden due to scrolling.

See Also

LeftCol (see page 3072)

MouseMove (see page 3052)

TopRow (see page 3080)

VisibleColCount (see page 3082)

VisibleRowCount (see page 3082)

10.1.3.5.63 TCustomGrid.TopLeftChanged Method

Updates the position of the inplace editor when the grid scrolls.

Pascal

```
procedure TopLeftChanged; dynamic;
```

C++

```
__fastcall TopLeftChanged();
```

Description

Grid controls call TopLeftChanged internally, whenever the LeftCol (see page 3072) or TopRow (see page 3080) property changes.

Descendants of TCustomGrid (see page 3027) override this method to make additional adjustments or to generate an OnTopLeftChanged event.

See Also

InplaceEditor (see page 3071)

LeftCol (see page 3072)

TopRow (see page 3080)

10.1.3.5.64 TCustomGrid.UpdateDesigner Method

Notifies the designer when rows and columns of the grid are moved or resized.

Pascal

```
procedure UpdateDesigner;
```

C++

```
__fastcall UpdateDesigner();
```

Description

At design time, the designer automatically keeps track of property changes so that it can prompt the user to save any changes. Moving or resizing the rows and columns of the grid using the mouse bypasses this system. When such changes occur, UpdateDesigner notifies the designer that the properties of the grid have changed and that the changes should be saved.

See Also

MouseUp (see page 3052)

10.1.3.6 TCustomGrid Properties

The properties of the TCustomGrid class are listed here.

Protected Properties

	Name	Description
	BorderStyle (see page 3060)	Determines whether a single line border is drawn around the grid.
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	Color (see page 3063)	Specifies the background color of the control.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Options (see page 3072)	Specifies various display and behavioral properties of the grid.
	ParentColor (see page 3073)	Determines where a control looks for its color information.

	Row (see page 3074)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	ScrollBars (see page 3077)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Published Properties

	Name	Description
	TabStop (see page 3079)	Determines if the user can tab to a control.

10.1.3.6.1 TCustomGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

10.1.3.6.2 TCustomGrid.Col Property

Specifies the index of the column that contains the selected cell.

Pascal

```
property Col: Longint;
```

C++

```
__property Longint Col;
```

Description

Use Col at runtime to determine the current column in the grid. Setting Col moves focus to the cell in the current row that is in the new column. The first column has an index of 0, the second column an index of 1, and so on.

The selected cell in the grid can be located by reading the Col property and the Row (see page 3074) property to obtain the indexes of its column and row. When focus moves to the grid, the selected cell gets input focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
*/
void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
```

```
  Label1->Caption = "From OnClick: The cursor is in column " +
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(index) +
      " has the focus.";
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
}
procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Row (see page 3074)

10.1.3.6.3 TCustomGrid.ColCount Property

Specifies the number of columns in the grid.

Pascal

```
property ColCount: Longint;
```

C++

```
__property Longint ColCount;
```

Description

Read ColCount to determine the number entries in the ColWidths (see page 3063) array. Set ColCount to add or delete columns at the righthand side of the grid. The value of ColCount includes any fixed columns at the left of the grid as well as the scrollable columns in the body of the grid.

C++ Examples:

```
/*
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I, J, K;

  K = 0;

  for (I = 0; I < StringGrid1->ColCount; I++)
    for (J = 0; J < StringGrid1->RowCount; J++)
      StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColWidths (see page 3063)

RowCount (see page 3075)

VisibleColCount (see page 3082)

10.1.3.6.4 TCustomGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

10.1.3.6.5 TCustomGrid.ColWidths Property

Indicates the width (in pixels) of all the columns in the grid.

Pascal

```
property ColWidths [Index: Longint]: Integer;
```

C++

```
__property int ColWidths[Longint Index];
```

Description

Read ColWidths to determine the width of the cells in the column specified by the Index parameter. An index of 0 refers to the first column. Set ColWidths at runtime to change the width of an individual column.

If the width of a column has not been set explicitly by resizing with the mouse, or by using the ColWidths property, its width is DefaultColWidth (see page 3064).

Use the ColCount (see page 3062) property to determine the number of entries in the ColWidths array.

See Also

ColCount (see page 3062)

DefaultColWidth (see page 3064)

RowHeights (see page 3076)

10.1.3.6.6 TCustomGrid.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths (see page 3063) property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount (see page 3062) property, they are created with a width of DefaultColWidth.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
```

```
    int index = ARow * DrawGrid1->ColCount + ACol;
    DrawGrid1->Canvas->Brush->Color = clBackground;
    DrawGrid1->Canvas->FillRect(Rect);
    ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
    if (State.Contains(gdFocused))
    {
      DrawGrid1->Canvas->DrawFocusRect(Rect);
    }
  }

  void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
  &CanSelect)

  {
    CanSelect = True;
  }
```

Delphi Examples:

```
  {
  The following code uses the bitmaps in an image list
  component to draw the contents of each cell in a draw grid.
  This example includes a draw grid on a form. When the
  application runs and the form is created, the width of the
  lines on the draw grid changes if the default column width of
  the grid is over 90 pixels wide.  This example requires a
  populated image list, or can just display an empty grid.
  Set the drawgrid DefaultColWidth and GridLineWidth statically
  to values the cause the button click to do something.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if DrawGrid1.DefaultColWidth > 90 then
      DrawGrid1.GridLineWidth := 2
    else
      DrawGrid1.GridLineWidth := 1;
  end;

  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

ColCount (see page 3062)

ColWidths (see page 3063)

DefaultRowHeight (see page 3066)

10.1.3.6.7 TCustomGrid.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell (see page 3043) method. For TCustomGrid (see page 3027), DrawCell (see page 3043) is an abstract method. Descendants of TCustomGrid (see page 3027) use DrawCell (see page 3043) to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 3043)

OnDrawCell (see page 3134)

10.1.3.6.8 TCustomGrid.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights (see page 3076) property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount (see page 3075) property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth (see page 3064)

RowCount (see page 3075)

RowHeights (see page 3076)

10.1.3.6.9 TCustomGrid.EditorMode Property

Determines whether the current cell can be edited.

Pascal

```
property EditorMode: Boolean;
```

C++

```
__property Boolean EditorMode;
```

Description

Set EditorMode to true, at runtime, to put the grid in edit mode. When EditorMode is true, the user can edit cells in the grid. When the user presses F2, EditorMode is set to true. When the user presses Enter, the value of EditorMode is toggled.

Note: EditorMode has no effect if the Options (see page 3072) property does not include goEditing or if the Options (see page 3072) property includes goAlwaysShowEditor.

See Also

HideEditor (see page 3048)

Options (see page 3072)

ShowEditor (see page 3056)

10.1.3.6.10 TCustomGrid.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color (see page 3063) property.

See Also

Color

FixedCols (see page 3067)

FixedRows (see page 3068)

10.1.3.6.11 TCustomGrid.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids (see page 2997) must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount (see page 3062) - 1.

See Also

ColCount (see page 3062)

DefaultDrawing (see page 3066)

FixedColor (see page 3067)

FixedRows (see page 3068)

10.1.3.6.12 TCustomGrid.FixedRows Property

Specifies the number of rows on the top of the grid that cannot be scrolled.

Pascal

```
property FixedRows: Integer;
```

C++

```
__property int FixedRows;
```

Description

Set FixedRows to create or get rid of nonscrolling rows. Nonscrolling rows appear at the top of the grid, and are always visible, even when the user scrolls the other rows in the grid. Use nonscrolling rows for displaying column titles or column numbers.

Nonscrolling rows are visually distinct from the scrollable rows in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids (see page 2997) must include at least one scrolling row. Do not set FixedRows to a value greater than RowCount (see page 3075) - 1.

See Also

DefaultDrawing (see page 3066)

FixedColor (see page 3067)

FixedCols (see page 3067)

RowCount (see page 3075)

10.1.3.6.13 TCustomGrid.GridHeight Property

Specifies the height of the grid in pixels.

Pascal

```
property GridHeight: Integer;
```

C++

```
__property int GridHeight;
```

Description

Read GridHeight to determine the size of the grid. If GridHeight is less than the value of ClientHeight, all of the rows of the grid appear in the control with an empty region below the grid. If the underlying grid is too tall to appear in the control, GridHeight is the same as ClientHeight, and the user must scroll to see the entire contents of the grid.

See Also

ClientHeight

GridWidth (see page 3070)

10.1.3.6.14 TCustomGrid.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options (see page 3072) property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}
```

```
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols (see page 3067)

FixedRows (see page 3068)

Options (see page 3072)

10.1.3.6.15 TCustomGrid.GridWidth Property

Specifies the width of the grid in pixels.

Pascal

```
property GridWidth: Integer;
```

C++

```
__property int GridWidth;
```

Description

Read GridWidth to determine the size of the grid. If GridWidth is less than the value of ClientWidth, all of the columns of the grid appear in the control with an empty region to the right of the grid. If the underlying grid is too wide to appear in the control, GridWidth is the same as ClientWidth, and the user must scroll to see the entire contents of the grid.

See Also

ClientWidth

GridHeight (see page 3068)

10.1.3.6.16 TCustomGrid.HitTest Property

Indicates the current position of the mouse cursor.

Pascal

```
property HitTest: TPoint;
```

C++

```
__property TPoint HitTest;
```

Description

Read HitTest to determine the position of the mouse when it is over the grid or when the grid has captured the mouse. HitTest may represent a position that is not within the client area of the grid control if the user is moving or resizing the rows or columns, or if the user is selecting a range of cells using the mouse. HitTest is the position of the mouse cursor in the coordinate system where (0,0) is the upper left corner of the grid control's client area.

See Also

ClientOrigin

MouseCoord (see page 3051)

10.1.3.6.17 TCustomGrid.InplaceEditor Property

Designates the edit control the grid uses to allow users to edit the contents of the selected cell.

Pascal

```
property InplaceEditor: TInplaceEdit;
```

C++

```
__property TInplaceEdit InplaceEditor;
```

Description

Read InplaceEditor to gain direct access to the edit control that sits on top of the selected cell when the grid is in edit mode. The inplace editor is created the first time the grid is put in edit mode. It is not destroyed until the grid is destroyed. The grid moves the editor when the selected cell changes. The grid hides the editor's window when the grid goes out of edit mode.

See Also

CreateEditor (see page 3040)

EditorMode (see page 3066)

HideEditor (see page 3048)

ShowEditor (see page 3056)

10.1.3.6.18 TCustomGrid.LeftCol Property

Specifies the index of the first visible scrollable column in the grid.

Pascal

```
property LeftCol: Longint;
```

C++

```
__property Longint LeftCol;
```

Description

Read LeftCol to determine the index of the first column in the scrollable region that is visible. Set LeftCol to scroll the columns in the grid so that the column with index LeftCol is the first column after the fixed columns.

See Also

FixedCols (see page 3067)

ScrollData (see page 3055)

VisibleColCount (see page 3082)

10.1.3.6.19 TCustomGrid.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
```

```
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColWidths (see page 3011)

EditorMode (see page 3066)

GridLineWidth (see page 3069)

RowHeights (see page 3076)

Selection (see page 3077)

TabStops (see page 3079)

10.1.3.6.20 TCustomGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

10.1.3.6.21 TCustomGrid.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col (see page 3060) property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

C++ Examples:

```
/*
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items = StringGrid1->Rows[StringGrid1->Row];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{

  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
```

```
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";

}
```

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

Col (see page 3060)

10.1.3.6.22 TCustomGrid.RowCount Property

Specifies the number of rows in the grid.

Pascal

```
property RowCount: Longint;
```

C++

```
__property Longint RowCount;
```

Description

Read RowCount to determine the number entries in the RowHeights (see page 3076) array. Set RowCount to add or delete rows at the bottom of the grid. The value of RowCount includes any fixed rows at the top of the grid as well as the scrollable rows in the body of the grid.

C++ Examples:

```
/*
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I, J, K;

  K = 0;

  for (I = 0; I < StringGrid1->ColCount; I++)
    for (J = 0; J < StringGrid1->RowCount; J++)
      StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 3062)

RowHeights (see page 3076)

VisibleRowCount (see page 3082)

10.1.3.6.23 TCustomGrid.RowHeights Property

Gives the height (in pixels) of all rows in the grid.

Pascal

```
property RowHeights [Index: Longint]: Integer;
```

C++

```
__property int RowHeights[Longint Index];
```

Description

Read RowHeights to determine the height of the cells in the row specified by the Index parameter. An index of 0 refers to the first row. Set RowWidths at runtime to change the height of an individual row.

If the height of a row has not been set explicitly by resizing with the mouse, or by using the RowHeights property, it's height is DefaultRowHeight (see page 3066).

Use the RowCount (see page 3075) property to determine the number of entries in the RowHeights array.

See Also

ColWidths (see page 3063)

DefaultRowHeight (see page 3066)

RowCount (see page 3075)

10.1.3.6.24 TCustomGrid.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight

ClientWidth

10.1.3.6.25 TCustomGrid.Selection Property

Indicates the boundaries of the current selection.

Pascal

```
property Selection: TGridRect;
```

C++

```
__property TGridRect Selection;
```

Description

Set Selection to select a range of cells in the grid. Selection can either represent the first column, first row, last column and last row in the grid, or the row and column coordinates of the upper left and bottom right cells in the selected region.

Selection can only indicate more than one cell when Options (see page 3072) includes goRangeSelect.

C++ Examples:

```
/*
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TGridRect myRect;
  myRect.Left = 3;
```

```
  myRect.Top = 1;
  myRect.Right = 2;
  myRect.Bottom = 4;
  DrawGrid1->Selection = myRect;
  DrawGrid1->Repaint();
}

void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
  Label1->Caption = "From OnClick: The cursor is in column " +
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = (ARow * DrawGrid1->ColCount + ACol) % (DrawGrid1->ColCount + 1);
  if (State.Contains(gdSelected))
    DrawGrid1->Canvas->Brush->Color = clYellow;
  else
    DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(ARow * DrawGrid1->ColCount + ACol) +
      " has the focus.";
  }
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  myRect: TGridRect;
begin
  myRect.Left := 2;
  myRect.Top := 1;
  myRect.Right := 3;
  myRect.Bottom := 4;
  DrawGrid1.Selection := myRect;
  DrawGrid1.Repaint;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;
```

```
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(ARow * DrawGrid1.ColCount + ACol) +
      ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Col (see page 3060)

Options (see page 3072)

Row (see page 3074)

10.1.3.6.26 TCustomGrid.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

10.1.3.6.27 TCustomGrid.TabStops Property

Indicates whether the user can tab to specified columns in the grid.

Pascal

```
property TabStops [Index: Longint]: Boolean;
```

C++

```
__property Boolean TabStops[Longint Index];
```

Description

Set TabStops to false to remove the column identified by Index from the tab order. The first column in the grid is identified by an Index of 0.

Note: When the Options (see page 3072) property does not include goTabs, the user can't tab to any column, even if TabStops is true.

See Also

Options (see page 3072)

10.1.3.6.28 TCustomGrid.TopRow Property

Specifies the index of the first visible scrollable row in the grid.

Pascal

```
property TopRow: Longint;
```

C++

```
__property Longint TopRow;
```

Description

Read TopRow to determine the index of the first row in the scrollable region that is visible. Set TopRow to scroll the rows in the grid so that the row with index TopRow is the first row after the fixed rows.

C++ Examples:

```
*/
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DrawGrid1->TopRow = DrawGrid1->RowCount - 1;
}

void __fastcall TForm1::DrawGrid1Click(TObject *Sender)
{
  Label1->Caption = "From OnClick: The cursor is in column " +
                    IntToStr(DrawGrid1->Col + 1) +
                    ", row " +
                    IntToStr(DrawGrid1->Row + 1);
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = (ARow * DrawGrid1->ColCount + ACol) % (DrawGrid1->ColCount + 1);
  if (State.Contains(gdSelected))
    DrawGrid1->Canvas->Brush->Color = clYellow;
  else
    DrawGrid1->Canvas->Brush->Color = clBackground;
```

```
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label2->Caption = "From OnDrawCell: Cell " + IntToStr(ARow * DrawGrid1->ColCount + ACol) +
      " has the focus.";
  }
  if ((ARow >=  DrawGrid1->FixedRows) && (ACol >=  DrawGrid1->FixedCols))
    ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)
{
  CanSelect = True;
}
```

Delphi Examples:

```
{
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DrawGrid1.TopRow := DrawGrid1.RowCount - 1;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(ARow * DrawGrid1.ColCount + ACol) + '
has the focus.';
  end;
  if ((ARow >=  DrawGrid1.FixedRows) AND (ACol >=  DrawGrid1.FixedCols)) then
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedRows (see page 3068)

ScrollData (see page 3055)

VisibleRowCount (see page 3082)

10.1.3.6.29 TCustomGrid.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount: Integer;
```

C++

```
__property int VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols (see page 3067) property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount (see page 3062)

FixedCols (see page 3067)

VisibleRowCount (see page 3082)

10.1.3.6.30 TCustomGrid.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows (see page 3068) property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows (see page 3068)

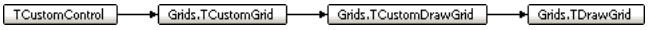
RowCount (see page 3075)

VisibleColCount (see page 3082)

10.1.4 TDrawGrid Class

TDrawGrid represents a grid control that displays information in column and row format.

Class Hierarchy



Pascal

```
TDrawGrid = class(TCustomDrawGrid);
```

C++

```
class TDrawGrid : public TCustomDrawGrid;
```

File

Grids

Description

Add a TDrawGrid object to a form to present arbitrary information in a tabular format. TDrawGrid provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TDrawGrid introduces several events and methods to allow applications to customize the appearance and behavior of the grid.

If the cells of the grid contain text strings that represent objects, consider using a TStringGrid (see page 3180) object instead. If the grid displays field values from the records in a dataset, use TDBGrid instead.

See Also

TDBGrid

TStringGrid (see page 3180)

Draw Grids

Grids

TGridOption (see page 3201)

10.1.4.1 TDrawGrid Members

The following tables list the members exposed by TDrawGrid.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.

	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
V	ColumnMoved (see page 3004)	Generates an OnColumnMoved (see page 3019) event.
	ColumnMoved (see page 3039)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
V	CreateEditor (see page 3040)	Creates the inplace editor control.
V	CreateParams (see page 3041)	Initializes the window-creation parameter record.
V	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.
V	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
V	DeleteRow (see page 3042)	Deletes a specified from the grid.
V	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
V	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
V	DrawCell (see page 3004)	Generates an OnDrawCell (see page 3020) event.
A	DrawCell (see page 3043)	Provides the prototype for a method that draws the contents of a cell in the grid.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
V	GetEditMask (see page 3004)	Generates an OnGetEditMask (see page 3021) event.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 3005)	Generates an OnGetEditText (see page 3022) event.
	GetEditText (see page 3046)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
V	KeyDown (see page 3050)	Provides special processing when the user presses a key.
V	KeyPress (see page 3050)	Provides special processing when the user presses a key.
V	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.

V	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
V	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
V	RowMoved (see page 3005)	Generates an OnRowMoved (see page 3023) event.
	RowMoved (see page 3055)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 3006)	Determines whether a particular cell in the grid can be selected.
V	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
V	SetEditText (see page 3006)	Generates an OnSetEditText (see page 3026) event.
	SetEditText (see page 3056)	Provides the interface for a method that updates the text associated with a cell.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
V	TopLeftChanged (see page 3007)	Generates an OnTopLeftChanged (see page 3027) event.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 3074)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.

Public Methods

	Name	Description
	CellRect (see page 3003)	Returns the screen coordinates of a cell in the grid.
V	Create (see page 3030)	Creates an instance of TCustomGrid (see page 3027).
V	Destroy (see page 3031)	Destroys an instance of TCustomGrid (see page 3027).
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 3005)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

Public Properties

	Name	Description
	Canvas (see page 3007)	Specifies the TCanvas object that presents a drawing surface for the control.

	Col (see page 3009)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 3011)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 3011)	Determines whether the current cell can be edited.
	GridHeight (see page 3012)	Specifies the height of the grid in pixels.
	GridWidth (see page 3012)	Specifies the width of the grid in pixels.
	LeftCol (see page 3012)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 3013)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 3014)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 3015)	Indicates the boundaries of the current selection.
	TabStops (see page 3017)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3017)	Specifies the index of the first visible scrollable row in the grid.

Published Events

	Name	Description
	OnClick (see page 3125)	Occurs when the user clicks the control.
	OnColumnMoved (see page 3127)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 3127)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3128)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 3128)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3131)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 3134)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 3135)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3135)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 3136)	Occurs when a control receives the input focus.
	OnExit (see page 3139)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 3142)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3142)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 3142)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 3146)	Occurs when a key is pressed.
	OnKeyUp (see page 3147)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 3149)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 3150)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 3152)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 3152)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 3153)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 3155)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 3157)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 3157)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 3158)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3158)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3159)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 3159)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3160)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 3160)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Published Properties

	Name	Description
	Align (see page 3088)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3092)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 3092)	Specifies which edges of the control are beveled.
	BevelInner (see page 3093)	Specifies the cut of the inner bevel.
	BevelKind (see page 3093)	Specifies the control's bevel style.
	BevelOuter (see page 3094)	Specifies the cut of the outer bevel.
	BevelWidth (see page 3094)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 3095)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 3095)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 3095)	Specifies the number of columns in the grid.
	Color (see page 3096)	Specifies the background color of the control.
	Constraints (see page 3097)	Specifies the size constraints for the control.
	Ctl3D (see page 3098)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 3098)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3100)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3100)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DoubleBuffered (see page 3101)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 3102)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3103)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3103)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 3105)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 3105)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3106)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3106)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 3107)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 3108)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Options (see page 3109)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 3111)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 3111)	Determines where a control looks for its color information.
	ParentCtl3D (see page 3112)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 3112)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 3112)	Determines where a control looks for its font information.
	ParentShowHint (see page 3113)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3113)	Identifies the pop-up menu associated with the control.
	RowCount (see page 3118)	Specifies the number of rows in the grid.
	ScrollBars (see page 3119)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 3120)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 3121)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 3079)	Determines if the user can tab to a control.
	Visible (see page 3122)	Determines whether the component appears onscreen.
	VisibleColCount (see page 3123)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 3124)	Indicates the number of scrollable rows visible in the grid.

10.1.4.2 TDrawGrid Properties

The properties of the TDrawGrid class are listed here.

Published Properties

	Name	Description
	Align (see page 3088)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3092)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 3092)	Specifies which edges of the control are beveled.
	BevelInner (see page 3093)	Specifies the cut of the inner bevel.
	BevelKind (see page 3093)	Specifies the control's bevel style.
	BevelOuter (see page 3094)	Specifies the cut of the outer bevel.
	BevelWidth (see page 3094)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 3095)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 3095)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 3095)	Specifies the number of columns in the grid.
	Color (see page 3096)	Specifies the background color of the control.
	Constraints (see page 3097)	Specifies the size constraints for the control.
	Ctl3D (see page 3098)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 3098)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3100)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3100)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DoubleBuffered (see page 3101)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 3102)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3103)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3103)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 3105)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 3105)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3106)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3106)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 3107)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 3108)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Options (see page 3109)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 3111)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 3111)	Determines where a control looks for its color information.
	ParentCtl3D (see page 3112)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 3112)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 3112)	Determines where a control looks for its font information.
	ParentShowHint (see page 3113)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3113)	Identifies the pop-up menu associated with the control.
	RowCount (see page 3118)	Specifies the number of rows in the grid.
	ScrollBars (see page 3119)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 3120)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 3121)	Indicates the position of the control in its parent's tab order.
	Visible (see page 3122)	Determines whether the component appears onscreen.
	VisibleColCount (see page 3123)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 3124)	Indicates the number of scrollable rows visible in the grid.

10.1.4.2.1 TDrawGrid.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
```

```
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
```

```
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

10.1.4.2.2 TDrawGrid.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

10.1.4.2.3 TDrawGrid.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

10.1.4.2.4 TDrawGrid.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

10.1.4.2.5 TDrawGrid.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

10.1.4.2.6 TDrawGrid.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

10.1.4.2.7 TDrawGrid.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

10.1.4.2.8 TDrawGrid.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

10.1.4.2.9 TDrawGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

10.1.4.2.10 TDrawGrid.ColCount Property

Specifies the number of columns in the grid.

Pascal

```
property ColCount: Longint;
```

C++

```
__property Longint ColCount;
```

Description

Read ColCount to determine the number entries in the ColWidths array. Set ColCount to add or delete columns at the righthand side of the grid. The value of ColCount includes any fixed columns at the left of the grid as well as the scrollable columns in the body of the grid.

C++ Examples:

```
/*
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I, J, K;

  K = 0;

  for (I = 0; I < StringGrid1->ColCount; I++)
    for (J = 0; J < StringGrid1->RowCount; J++)
      StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColWidths (see page 3063)

RowCount (see page 3075)

VisibleColCount (see page 3082)

10.1.4.2.11 TDrawGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

10.1.4.2.12 TDrawGrid.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

10.1.4.2.13 TDrawGrid.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

10.1.4.2.14 TDrawGrid.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount property, they are created with a width of DefaultColWidth.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
```

```
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;
```

```
procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColCount (see page 3062)

ColWidths (see page 3063)

DefaultRowHeight (see page 3066)

10.1.4.2.15 TDrawGrid.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 3043)

OnDrawCell (see page 3134)

10.1.4.2.16 TDrawGrid.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth (see page 3064)

RowCount (see page 3075)

RowHeights (see page 3076)

10.1.4.2.17 TDrawGrid.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
```

```
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

10.1.4.2.18 TDrawGrid.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

10.1.4.2.19 TDrawGrid.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

10.1.4.2.20 TDrawGrid.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
```

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

10.1.4.2.21 TDrawGrid.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

10.1.4.2.22 TDrawGrid.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color

FixedCols (see page 3067)

FixedRows (see page 3068)

10.1.4.2.23 TDrawGrid.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount - 1.

See Also

ColCount (see page 3062)

DefaultDrawing (see page 3066)

FixedColor (see page 3067)

FixedRows (see page 3068)

10.1.4.2.24 TDrawGrid.FixedRows Property

Specifies the number of rows on the top of the grid that cannot be scrolled.

Pascal

```
property FixedRows: Integer;
```

C++

```
__property int FixedRows;
```

Description

Set FixedRows to create or get rid of nonscrolling rows. Nonscrolling rows appear at the top of the grid, and are always visible, even when the user scrolls the other rows in the grid. Use nonscrolling rows for displaying column titles or column numbers.

Nonscrolling rows are visually distinct from the scrollable rows in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling row. Do not set FixedRows to a value greater than RowCount - 1.

See Also

DefaultDrawing (see page 3066)

FixedColor (see page 3067)

FixedCols (see page 3067)

RowCount (see page 3075)

10.1.4.2.25 TDrawGrid.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

10.1.4.2.26 TDrawGrid.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
```

```
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols (see page 3067)

FixedRows (see page 3068)

Options (see page 3072)

10.1.4.2.27 TDrawGrid.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColWidths (see page 3011)

EditorMode (see page 3066)

GridLineWidth (see page 3069)

RowHeights (see page 3076)

Selection (see page 3077)

TabStops (see page 3079)

10.1.4.2.28 TDrawGrid.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

10.1.4.2.29 TDrawGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

10.1.4.2.30 TDrawGrid.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

10.1.4.2.31 TDrawGrid.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

10.1.4.2.32 TDrawGrid.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own

Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

10.1.4.2.33 TDrawGrid.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

10.1.4.2.34 TDrawGrid.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
```

```
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
    else
      Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

10.1.4.2.35 TDrawGrid.RowCount Property

Specifies the number of rows in the grid.

Pascal

```
property RowCount: Longint;
```

C++

```
__property Longint RowCount;
```

Description

Read RowCount to determine the number entries in the RowHeights array. Set RowCount to add or delete rows at the bottom of the grid. The value of RowCount includes any fixed rows at the top of the grid as well as the scrollable rows in the body of the grid.

C++ Examples:

```
/*
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I, J, K;

  K = 0;

  for (I = 0; I < StringGrid1->ColCount; I++)
    for (J = 0; J < StringGrid1->RowCount; J++)
      StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 3062)

RowHeights (see page 3076)

VisibleRowCount (see page 3082)

10.1.4.2.36 TDrawGrid.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight

ClientWidth

10.1.4.2.37 TDrawGrid.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
```

```
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

10.1.4.2.38 TDrawGrid.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a

TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

10.1.4.2.39 TDrawGrid.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

10.1.4.2.40 TDrawGrid.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount: Integer;
```

C++

```
__property int VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount (see page 3062)

FixedCols (see page 3067)

VisibleRowCount (see page 3082)

10.1.4.2.41 TDrawGrid.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows (see page 3068)

RowCount (see page 3075)

VisibleColCount (see page 3082)

10.1.4.3 TDrawGrid Events

The events of the TDrawGrid class are listed here.

Published Events

	Name	Description
	OnClick (see page 3125)	Occurs when the user clicks the control.
	OnColumnMoved (see page 3127)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 3127)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3128)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 3128)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3131)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 3134)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 3135)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3135)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 3136)	Occurs when a control receives the input focus.

	OnExit (see page 3139)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 3142)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3142)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 3142)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 3146)	Occurs when a key is pressed.
	OnKeyUp (see page 3147)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 3149)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 3150)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 3152)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 3152)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 3153)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 3155)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 3157)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 3157)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 3158)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3158)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3159)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 3159)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3160)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 3160)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

10.1.4.3.1 TDrawGrid.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.

- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

10.1.4.3.2 TDrawGrid.OnColumnMoved Event

Occurs immediately after the position of a column changes.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Use OnColumnMoved to perform special processing when the position of a grid column changes. Columns can be moved programmatically or by user manipulation. The FromIndex parameter is the old index of the column, and the ToIndex parameter is the new index. OnColumnMoved does not occur unless the Options (see page 3109) property includes goColMoving.

See Also

OnRowMoved (see page 3158)

Options (see page 3072)

10.1.4.3.3 TDrawGrid.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

10.1.4.3.4 TDrawGrid.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

10.1.4.3.5 TDrawGrid.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
```

```
}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
```

```
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

10.1.4.3.6 TDrawGrid.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
```

```
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

10.1.4.3.7 TDrawGrid.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell: TDrawCellEvent;
```

C++

```
__property TDrawCellEvent OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas (see page 3007) property.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing (see page 3100) property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing (see page 3100) property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

See Also

Canvas

DefaultDrawing (see page 3066)

10.1.4.3.8 TDrawGrid.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

10.1.4.3.9 TDrawGrid.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
```

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

10.1.4.3.10 TDrawGrid.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

10.1.4.3.11 TDrawGrid.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));
```

```
    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

10.1.4.3.12 TDrawGrid.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the specified cell. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options (see page 3109) property includes goEditing.

See Also

Options (see page 3072)

EditMask (see page 4604)

10.1.4.3.13 TDrawGrid.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the specified cell. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell (see page 3134) event handler.

OnGetEditText does not occur unless the Options (see page 3109) property includes goEditing.

See Also

OnDrawCell (see page 3134)

OnSetEditText (see page 3159)

10.1.4.3.14 TDrawGrid.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
```

```
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
```

```
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

10.1.4.3.15 TDrawGrid.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

10.1.4.3.16 TDrawGrid.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
```

```
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

10.1.4.3.17 TDrawGrid.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events.

This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

10.1.4.3.18 TDrawGrid.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
```

```
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

10.1.4.3.19 TDrawGrid.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

10.1.4.3.20 TDrawGrid.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

10.1.4.3.21 TDrawGrid.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

10.1.4.3.22 TDrawGrid.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
    StatusBar1->Panels->Items[1]->Text = "";
    StatusBar1->Panels->Items[2]->Text = "";
    StatusBar1->Panels->Items[3]->Text = "";
  }

  // Use this code as the OnMouseMove event handler of the form:
  void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
  {
    if (Shift.Contains(ssLeft)) // make sure button is down
    {
      if (Y > StartY)
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
  {
  The following example requires a form with a four-paneled
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

10.1.4.3.23 TDrawGrid.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

10.1.4.3.24 TDrawGrid.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

10.1.4.3.25 TDrawGrid.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. The FromIndex parameter is the old index of the row, and the ToIndex parameter is the new index. OnRowMoved does not occur unless the Options (see page 3109) property includes goRowMoving.

See Also

OnColumnMoved (see page 3127)

Options (see page 3072)

10.1.4.3.26 TDrawGrid.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell: TSelectCellEvent;
```

C++

```
__property TSelectCellEvent OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected.

See Also

Col (see page 3060)

Row (see page 3074)

TabStops (see page 3079)

10.1.4.3.27 TDrawGrid.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText: TSetEditEvent;
```

C++

```
__property TSetEditEvent OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell (see page 3134) event handler. OnSetEditText occurs every time the user changes the text.

OnSetEditText does not occur unless the Options (see page 3109) property includes goEditing.

See Also

OnDrawCell (see page 3134)

OnGetEditText (see page 3142)

Options (see page 3072)

10.1.4.3.28 TDrawGrid.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

10.1.4.3.29 TDrawGrid.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

10.1.4.3.30 TDrawGrid.OnTopLeftChanged Event

Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol (see page 3072)

TopRow (see page 3080)

10.1.5 TInplaceEdit Class

TInplaceEdit provides text editing capabilities for grid controls.

Class Hierarchy

Pascal

```
TInplaceEdit = class(TCustomMaskEdit);
```

C++

```
class TInplaceEdit : public TCustomMaskEdit;
```

File

Grids

Description

TInplaceEdit is a helper object tailored to work with grid controls. It provides a multiline editing window that appears in the appropriate cell of a grid control. TInplaceEdit provides services so that the grid can influence the response of the control to user actions, and can move or resize the editing window as cells are moved or resized.

The TInplaceEdit window has no border, so it does not stand out as a separate control. TInplaceEdit is not a data-aware control, but can work with a data-aware grid such as TDBGrid, allowing the grid to fetch or post the text, and allowing the grid to impose a mask to limit the text of the edit control to data that is valid.

See Also

TCustomGrid (see page 3027)

TDBEdit

TDBGrid

10.1.5.1 TInplaceEdit Members

The following tables list the members exposed by TInplaceEdit.

Protected Methods

	Name	Description
	BoundsChanged (see page 3163)	Indents the text area of the window slightly after the grid repositions the edit control.
	CheckCursor (see page 4595)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	CreateParams (see page 3164)	Initializes a window-creation parameter record passed in the Params parameter.
	DblClick (see page 3164)	Informs the grid when the user double-clicks in the edit window.
	DoMouseWheel (see page 3165)	Delegates mouse wheel messages to the associated grid.
	EditCanModify (see page 3165)	Determines whether the user can edit the Text.
	GetFirstEditChar (see page 4596)	Returns the index in the EditMask (see page 4604) of the first nonliteral character.
	GetLastEditChar (see page 4597)	Returns the index in the EditMask (see page 4604) of the last nonliteral character.
	GetMaxChars (see page 4597)	Returns the length of the EditText (see page 4606).

	GetNextEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetPriorEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 4599)	Returns the endpoints of the current selection in the EditText (see page 4606).
V	KeyDown (see page 3167)	Responds when the user presses a key.
V	KeyPress (see page 3167)	Responds when the user types a key.
V	KeyUp (see page 3168)	Responds when the user releases a key.
	ReformatText (see page 4601)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
V	Reset (see page 4601)	Returns the EditText (see page 4606) to its value at the time when the edit control last lost focus.
	SetCursor (see page 4602)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetSel (see page 4602)	Selects character range.
V	UpdateContents (see page 3169)	Sets the EditMask and Text properties to values specified by the grid.
V	Validate (see page 4603)	Validates a specified string against the current mask.
V	ValidateError (see page 4603)	Beeps and raises an EDBEditError (see page 4592) exception.
V	WndProc (see page 3170)	Supplements the inherited method to add special processing to some Windows messages.

Protected Properties

	Name	Description
	EditMask (see page 4604)	Specifies the mask that represents what text is valid for the masked edit control.
R	Grid (see page 3171)	Specifies the grid object that owns and controls this TInplaceEdit (see page 3161).
	MaskState (see page 4607)	Indicates state information about how the mask is being applied.
	MaxLength (see page 4607)	Specifies the maximum number of characters that can appear in the edit control.

Public Methods

	Name	Description
V	Clear (see page 4595)	Sets the Text (see page 4608) property to an empty string.
V	Create (see page 3162)	Creates an instance of TInplaceEdit (see page 3161).
	Deselect (see page 3165)	Removes the selection from the edit window.
	GetTextLen (see page 4599)	Returns the length of the underlying text.
	Hide (see page 3166)	Hides the edit control window and moves input focus to the grid.
	Invalidate (see page 3166)	Invalidates the edit window and the corresponding portion of the grid window.
	Move (see page 3168)	Changes the position of the edit window and repaints the contents.
	PosEqual (see page 3168)	Indicates whether the edit window has the size and position indicated by Rect.
	SetFocus (see page 3169)	Gives the editor window input focus if it is visible.
	UpdateLoc (see page 3170)	Changes the position of the edit window but does not repaint the contents.
V	ValidateEdit (see page 4603)	Validates the EditText (see page 4606) against the current mask.
	Visible (see page 3170)	This is Visible, a member of class TInplaceEdit.

Public Properties

	Name	Description
	EditText (see page 4606)	Specifies the text that appears in the window of the masked edit control.
R	IsMasked (see page 4606)	Indicates whether a mask exists for the edit control.
	Text (see page 4608)	Denotes the underlying text for the masked edit control before the mask has been applied.

10.1.5.2 TInplaceEdit.Create Constructor

Creates an instance of TInplaceEdit (see page 3161).

Pascal

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

C++

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

Description

Do not create instances of TInplaceEdit (see page 3161). These are created automatically by the grid that uses the editor.

AOwner specifies the Owner for the editor. This is the component that is responsible for freeing the TInplaceEdit (see page 3161) instance.

Create overrides the default constructor to make sure that the control has no border to provide a visual indication that the control exists.

See Also

Hide (see page 3166)

BoundsChanged (see page 3163)

Owner

10.1.5.3 TInplaceEdit Methods

The methods of the TInplaceEdit class are listed here.

Protected Methods

	Name	Description
	BoundsChanged (see page 3163)	Indents the text area of the window slightly after the grid repositions the edit control.
	CreateParams (see page 3164)	Initializes a window-creation parameter record passed in the Params parameter.
	DblClick (see page 3164)	Informs the grid when the user double-clicks in the edit window.
	DoMouseWheel (see page 3165)	Delegates mouse wheel messages to the associated grid.
	EditCanModify (see page 3165)	Determines whether the user can edit the Text.
	KeyDown (see page 3167)	Responds when the user presses a key.
	KeyPress (see page 3167)	Responds when the user types a key.
	KeyUp (see page 3168)	Responds when the user releases a key.
	UpdateContents (see page 3169)	Sets the EditMask and Text properties to values specified by the grid.
	WndProc (see page 3170)	Supplements the inherited method to add special processing to some Windows messages.

Public Methods

	Name	Description
	Deselect (see page 3165)	Removes the selection from the edit window.
	Hide (see page 3166)	Hides the edit control window and moves input focus to the grid.
	Invalidate (see page 3166)	Invalidates the edit window and the corresponding portion of the grid window.
	Move (see page 3168)	Changes the position of the edit window and repaints the contents.
	PosEqual (see page 3168)	Indicates whether the edit window has the size and position indicated by Rect.
	SetFocus (see page 3169)	Gives the editor window input focus if it is visible.
	UpdateLoc (see page 3170)	Changes the position of the edit window but does not repaint the contents.
	Visible (see page 3170)	This is Visible, a member of class TInplaceEdit.

10.1.5.3.1 TInplaceEdit.BoundsChanged Method

Indents the text area of the window slightly after the grid repositions the edit control.

Pascal

```
procedure BoundsChanged; virtual;
```

C++

```
virtual __fastcall BoundsChanged();
```

Description

BoundsChanged reinitializes the editing window after the control has been moved or resized. It creates a slight indentation in the text formatting area so that the contents of the editor do not appear to crowd the edges of the cell.

BoundsChanged is exposed as a protected method so that derived objects can add further enhancements when the editing window changes size or location.

See Also

Move (see page 3168)

UpdateLoc (see page 3170)

PosEqual (see page 3168)

10.1.5.3.2 TInplaceEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. TInplaceEdit (see page 3161) overrides the inherited method to specify a multiline edit control.

See Also

CreateWnd

CreateWindowHandle

10.1.5.3.3 TInplaceEdit.DblClick Method

Informs the grid when the user double-clicks in the edit window.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick is called when the user double-clicks in the edit window. DblClick simply passes the event on to the grid. The grid must interpret and respond to the double-click. If the edit window needs to respond to the double-click, the grid must trigger the response by calling a method or setting a property of the TInplaceEdit (see page 3161).

See Also

Grid (see page 3171)

10.1.5.3.4 TInplaceEdit.Deselect Method

Removes the selection from the edit window.

Pascal

```
procedure Deselect;
```

C++

```
__fastcall Deselect();
```

Description

Use Deselect to cause all text in the edit window to be unselected. Deselect simply changes the selection, it does not change the actual text.

See Also

ClearSelection

SelectAll

KeyPress (see page 3167)

10.1.5.3.5 TInplaceEdit.DoMouseWheel Method

Delegates mouse wheel messages to the associated grid.

Pascal

```
function DoMouseWheel(Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint): Boolean;
override;
```

C++

```
virtual __fastcall Boolean DoMouseWheel(TShiftState Shift, int WheelDelta, TPoint MousePos);
```

Description

TInplaceEdit (see page 3161) overrides the inherited DoMouseWheel method so that it can pass any mouse wheel messages on to the grid specified by the Grid (see page 3171) property. This causes the grid's OnMouseWheel event handler to execute even though the inplace edit control has focus.

See Also

OnMouseWheel

Grid (see page 3171)

10.1.5.3.6 TInplaceEdit.EditCanModify Method

Determines whether the user can edit the Text.

Pascal

```
function EditCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean EditCanModify();
```

Description

EditCanModify is called automatically when the user tries to edit the value of Text. EditCanModify calls the CanEditModify method of the grid to determine whether the current cell of the grid should be edited. When EditCanModify returns true, the user is permitted to edit the text. When EditCanModify returns false, the user is prevented from editing the text.

See Also

Grid (see page 3171)

CanEditModify (see page 3037)

Text

10.1.5.3.7 TInplaceEdit.Hide Method

Hides the edit control window and moves input focus to the grid.

Pascal

```
procedure Hide;
```

C++

```
__fastcall Hide();
```

Description

Hide makes the edit control window invisible without destroying it. Hide also moves the input focus from the editor window to the grid, so that the invisible window does not swallow keyboard events.

The grid calls Hide to hide the edit window when changing out of edit mode. It also calls Hide when scrolling the cell that is being edited out of the grid window.

See Also

Grid (see page 3171)

10.1.5.3.8 TInplaceEdit.Invalidate Method

Invalidates the edit window and the corresponding portion of the grid window.

Pascal

```
procedure Invalidate; reintroduce;
```

C++

```
__fastcall Invalidate();
```

Description

Invalidate ensures that the portion of the grid window under the edit control is invalidated at the same time as the edit window. This is important when the edit window is hidden or destroyed so that the grid window can repaint the cell that was covered by the edit window.

See Also

Grid (see page 3171)

Hide (see page 3166)

10.1.5.3.9 TInplaceEdit.KeyDown Method

Responds when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When the user presses a key, KeyDown checks whether the edit control or grid should process the KeyDown event. The Enter key is always passed on to the grid. The F2 key calls Deselect (see page 3165) to remove the current selection. Navigation keystrokes are passed on to the grid when the movement cannot be accommodated within the text window. For example, if the user types Left when the cursor is already at the left edge of the edit window, the keystroke is passed to the grid which moves focus to the previous cell. If the cursor is in the middle of a line, the keystroke is processed by the edit window, which moves left a single character.

KeyDown calls the OnKeyDown event handler for the grid, if there is one, before calling the inherited KeyDown for the edit control.

See Also

Grid (see page 3171)

KeyPress (see page 3167)

KeyUp (see page 3168)

KeyDown (see page 4599)

10.1.5.3.10 TInplaceEdit.KeyPress Method

Responds when the user types a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress allows the grid to handle the keystroke, and only processes keystrokes that the grid does not fully handle. If the grid is not in edit mode after handling the keystroke, KeyPress throws away the keystroke. Keystroke processes the Return Key and calls the inherited method to handle all other unprocessed keystrokes.

Pressing the Return key causes KeyPress to select all the text, unless it is already selected, in which case it causes KeyPress to deselect all the text.

See Also

Grid (see page 3171)

SelectAll

Deselect (see page 3165)

KeyPress (see page 4600)

KeyDown (see page 3167)

KeyUp (see page 3168)

10.1.5.3.11 TInplaceEdit.KeyUp Method

Responds when the user releases a key.

Pascal

```
procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyUp(Word Key, TShiftState Shift);
```

Description

The inplace edit control does not process the KeyUp event at all. It simply passes this event on to the grid, which may or may not process the event.

See Also

Grid (see page 3171)

KeyDown (see page 3167)

KeyPress (see page 3167)

10.1.5.3.12 TInplaceEdit.Move Method

Changes the position of the edit window and repaints the contents.

Pascal

```
procedure Move(const Loc: TRect);
```

C++

```
__fastcall Move(const TRect Loc);
```

Description

The grid calls Move to move the edit window to the indicated rectangle when the corresponding cell moves or resizes. The position of Loc is relative to the grid's client area.

See Also

Grid (see page 3171)

UpdateLoc (see page 3170)

10.1.5.3.13 TInplaceEdit.PosEqual Method

Indicates whether the edit window has the size and position indicated by Rect.

Pascal

```
function PosEqual(const Rect: TRect): Boolean;
```

C++

```
__fastcall Boolean PosEqual(const TRect Rect);
```

Description

Use PosEqual to determine if the edit window occupies a particular rectangle, where Rect is given in global coordinates. PosEqual can be used to determine whether the inplace edit control represents a particular cell in the grid, by checking the bounding rectangle of the cell.

See Also

BoundsRect

TCustomGrid (see page 3027)

10.1.5.3.14 TInplaceEdit.SetFocus Method

Gives the editor window input focus if it is visible.

Pascal

```
procedure SetFocus; reintroduce;
```

C++

```
__fastcall SetFocus();
```

Description

Use SetFocus to cause keyboard events to go to the editor window. Focus will not move to the window if it is hidden by a call to the Hide (see page 3166) method.

See Also

KeyPress (see page 3167)

KeyDown (see page 3167)

KeyUp (see page 3168)

Hide (see page 3166)

10.1.5.3.15 TInplaceEdit.UpdateContents Method

Sets the EditMask and Text properties to values specified by the grid.

Pascal

```
procedure UpdateContents; virtual;
```

C++

```
virtual __fastcall UpdateContents();
```

Description

UpdateContents calls the GetEditMask and GetEditText methods of the grid, and sets the EditMask and Text properties accordingly. UpdateContents allows the grid to bring up the inplace edit control with the contents of a particular cell, and to impose a mask to restrict the editor to valid input for the particular cell.

See Also

Grid (see page 3171)

EditMask (see page 4604)

Text

GetEditMask (see page 3045)

GetEditText (see page 3046)

10.1.5.3.16 TInplaceEdit.UpdateLoc Method

Changes the position of the edit window but does not repaint the contents.

Pascal

```
procedure UpdateLoc(const Loc: TRect);
```

C++

```
__fastcall UpdateLoc(const TRect Loc);
```

Description

The grid calls UpdateLoc to move the edit window when the grid is moved. Unlike the Move (see page 3168) method, UpdateLoc does not invalidate the edit window to cause a repaint.

See Also

Grid (see page 3171)

Move (see page 3168)

Invalidate (see page 3166)

10.1.5.3.17 TInplaceEdit.Visible Method

Pascal

```
function Visible: Boolean;
```

C++

```
__fastcall Boolean Visible();
```

Description

This is Visible, a member of class TInplaceEdit.

10.1.5.3.18 TInplaceEdit.WndProc Method

Supplements the inherited method to add special processing to some Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method of a control that receives Windows messages.

WndProc for TInplaceEdit (see page 3161) passes focus events on to the grid. It calls the inherited method for all other Windows messages. WndProc converts mouse clicks to double clicks when the grid receives the first click within the double-click time period.

See Also

Grid (see page 3171)

WndProc

10.1.5.4 TInplaceEdit Properties

The properties of the TInplaceEdit class are listed here.

Protected Properties

	Name	Description
	Grid (see page 3171)	Specifies the grid object that owns and controls this TInplaceEdit (see page 3161).

10.1.5.4.1 TInplaceEdit.Grid Property

Specifies the grid object that owns and controls this TInplaceEdit (see page 3161).

Pascal

```
property Grid: TCustomGrid;
```

C++

```
__property TCustomGrid Grid;
```

Description

Read Grid to obtain the grid object for which TInplaceEdit (see page 3161) provides editing services. TInplaceEdit (see page 3161) uses the Grid property so that it can receive information about the text to be edited and so that it can pass on relevant events.

See Also

UpdateContents (see page 3169)

UpdateLoc (see page 3170)

10.1.6 TInplaceEditList Class

Class Hierarchy

Pascal

```
TInplaceEditList = class(TInPlaceEdit);
```

C++

```
class TInplaceEditList : public TInPlaceEdit;
```

File

Grids

Description

This is class Grids.TInplaceEditList.

10.1.6.1 TInplaceEditList Members

The following tables list the members exposed by TInplaceEditList.

Protected Methods

	Name	Description
	BoundsChanged (see page 3173)	This is BoundsChanged, a member of class TInplaceEditList.
	ButtonRect (see page 3173)	This is ButtonRect, a member of class TInplaceEditList.
	CloseUp (see page 3174)	This is CloseUp, a member of class TInplaceEditList.
	DblClick (see page 3174)	This is DblClick, a member of class TInplaceEditList.
	DoDropDownKeys (see page 3174)	This is DoDropDownKeys, a member of class TInplaceEditList.
	DoEditButtonClick (see page 3174)	This is DoEditButtonClick, a member of class TInplaceEditList.
	DoGetPickListItems (see page 3174)	This is DoGetPickListItems, a member of class TInplaceEditList.
	DropDown (see page 3175)	This is DropDown, a member of class TInplaceEditList.
	KeyDown (see page 3175)	This is KeyDown, a member of class TInplaceEditList.
	ListMouseUp (see page 3175)	This is ListMouseUp, a member of class TInplaceEditList.
	MouseDown (see page 3175)	This is MouseDown, a member of class TInplaceEditList.
	MouseMove (see page 3176)	This is MouseMove, a member of class TInplaceEditList.
	MouseUp (see page 3176)	This is MouseUp, a member of class TInplaceEditList.
	OverButton (see page 3176)	This is OverButton, a member of class TInplaceEditList.
	PaintWindow (see page 3176)	This is PaintWindow, a member of class TInplaceEditList.
	StopTracking (see page 3177)	This is StopTracking, a member of class TInplaceEditList.
	TrackButton (see page 3177)	This is TrackButton, a member of class TInplaceEditList.
	UpdateContents (see page 3177)	This is UpdateContents, a member of class TInplaceEditList.
	WndProc (see page 3177)	This is WndProc, a member of class TInplaceEditList.

Public Events

	Name	Description
	OnEditButtonClick (see page 3180)	This is OnEditButtonClick, a member of class TInplaceEditList.
	OnGetPickListitems (see page 3180)	This is OnGetPickListitems, a member of class TInplaceEditList.

Public Methods

	Name	Description
	Create (see page 3172)	This is Create, a member of class TInplaceEditList.
	RestoreContents (see page 3176)	This is RestoreContents, a member of class TInplaceEditList.

Public Properties

	Name	Description
	ActiveList (see page 3178)	This is ActiveList, a member of class TInplaceEditList.
	ButtonWidth (see page 3178)	This is ButtonWidth, a member of class TInplaceEditList.
	DropDownRows (see page 3178)	This is DropDownRows, a member of class TInplaceEditList.
	EditStyle (see page 3178)	This is EditStyle, a member of class TInplaceEditList.
	ListVisible (see page 3179)	This is ListVisible, a member of class TInplaceEditList.
	PickList (see page 3179)	This is PickList, a member of class TInplaceEditList.
	PickListLoaded (see page 3179)	This is PickListLoaded, a member of class TInplaceEditList.
	Pressed (see page 3179)	This is Pressed, a member of class TInplaceEditList.

10.1.6.2 TInplaceEditList.Create Constructor

Pascal

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

C++

```
virtual __fastcall TInplaceEditList(TComponent * Owner);
```

Description

This is Create, a member of class TInplaceEditList.

10.1.6.3 TInplaceEditList Methods

The methods of the TInplaceEditList class are listed here.

Protected Methods

	Name	Description
	BoundsChanged (see page 3173)	This is BoundsChanged, a member of class TInplaceEditList.
	ButtonRect (see page 3173)	This is ButtonRect, a member of class TInplaceEditList.
	CloseUp (see page 3174)	This is CloseUp, a member of class TInplaceEditList.
	DblClick (see page 3174)	This is DblClick, a member of class TInplaceEditList.
	DoDropDownKeys (see page 3174)	This is DoDropDownKeys, a member of class TInplaceEditList.
	DoEditButtonClick (see page 3174)	This is DoEditButtonClick, a member of class TInplaceEditList.
	DoGetPickListItems (see page 3174)	This is DoGetPickListItems, a member of class TInplaceEditList.
	DropDown (see page 3175)	This is DropDown, a member of class TInplaceEditList.
	KeyDown (see page 3175)	This is KeyDown, a member of class TInplaceEditList.
	ListMouseUp (see page 3175)	This is ListMouseUp, a member of class TInplaceEditList.
	MouseDown (see page 3175)	This is MouseDown, a member of class TInplaceEditList.
	MouseMove (see page 3176)	This is MouseMove, a member of class TInplaceEditList.
	MouseUp (see page 3176)	This is MouseUp, a member of class TInplaceEditList.
	OverButton (see page 3176)	This is OverButton, a member of class TInplaceEditList.
	PaintWindow (see page 3176)	This is PaintWindow, a member of class TInplaceEditList.
	StopTracking (see page 3177)	This is StopTracking, a member of class TInplaceEditList.
	TrackButton (see page 3177)	This is TrackButton, a member of class TInplaceEditList.
	UpdateContents (see page 3177)	This is UpdateContents, a member of class TInplaceEditList.
	WndProc (see page 3177)	This is WndProc, a member of class TInplaceEditList.

Public Methods

	Name	Description
	RestoreContents (see page 3176)	This is RestoreContents, a member of class TInplaceEditList.

10.1.6.3.1 TInplaceEditList.BoundsChanged Method

Pascal

```
procedure BoundsChanged; override;
```

C++

```
virtual __fastcall BoundsChanged();
```

Description

This is BoundsChanged, a member of class TInplaceEditList.

10.1.6.3.2 TInplaceEditList.ButtonRect Method

Pascal

```
function ButtonRect: TRect;
```

C++

```
__fastcall TRect ButtonRect();
```

Description

This is ButtonRect, a member of class TInplaceEditList.

10.1.6.3.3 TInplaceEditList.CloseUp Method

Pascal

```
procedure CloseUp(Accept: Boolean); dynamic;
```

C++

```
__fastcall CloseUp(Boolean Accept);
```

Description

This is CloseUp, a member of class TInplaceEditList.

10.1.6.3.4 TInplaceEditList.DblClick Method

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

This is DblClick, a member of class TInplaceEditList.

10.1.6.3.5 TInplaceEditList.DoDropDownKeys Method

Pascal

```
procedure DoDropDownKeys(var Key: Word; Shift: TShiftState); virtual;
```

C++

```
virtual __fastcall DoDropDownKeys(Word Key, TShiftState Shift);
```

Description

This is DoDropDownKeys, a member of class TInplaceEditList.

10.1.6.3.6 TInplaceEditList.DoEditButtonClick Method

Pascal

```
procedure DoEditButtonClick; virtual;
```

C++

```
virtual __fastcall DoEditButtonClick();
```

Description

This is DoEditButtonClick, a member of class TInplaceEditList.

10.1.6.3.7 TInplaceEditList.DoGetPickListItems Method

Pascal

```
procedure DoGetPickListItems; dynamic;
```

C++

```
__fastcall DoGetPickListItems();
```

Description

This is DoGetPickListItems, a member of class TInplaceEditList.

10.1.6.3.8 TInplaceEditList.DropDown Method

Pascal

```
procedure DropDown; dynamic;
```

C++

```
__fastcall DropDown();
```

Description

This is DropDown, a member of class TInplaceEditList.

10.1.6.3.9 TInplaceEditList.KeyDown Method

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

This is KeyDown, a member of class TInplaceEditList.

10.1.6.3.10 TInplaceEditList.ListMouseUp Method

Pascal

```
procedure ListMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X: Integer;
Y: Integer);
```

C++

```
__fastcall ListMouseUp(TObject * Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is ListMouseUp, a member of class TInplaceEditList.

10.1.6.3.11 TInplaceEditList.MouseDown Method

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseDown, a member of class TInplaceEditList.

10.1.6.3.12 TInplaceEditList.MouseMove Method

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

This is MouseMove, a member of class TInplaceEditList.

10.1.6.3.13 TInplaceEditList.MouseUp Method

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseUp, a member of class TInplaceEditList.

10.1.6.3.14 TInplaceEditList.OverButton Method

Pascal

```
function OverButton(const P: TPoint): Boolean;
```

C++

```
__fastcall Boolean OverButton(const TPoint P);
```

Description

This is OverButton, a member of class TInplaceEditList.

10.1.6.3.15 TInplaceEditList.PaintWindow Method

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

This is PaintWindow, a member of class TInplaceEditList.

10.1.6.3.16 TInplaceEditList.RestoreContents Method

Pascal

```
procedure RestoreContents;
```

C++

```
__fastcall RestoreContents();
```

Description

This is RestoreContents, a member of class TInplaceEditList.

10.1.6.3.17 TInplaceEditList.StopTracking Method

Pascal

```
procedure StopTracking;
```

C++

```
__fastcall StopTracking();
```

Description

This is StopTracking, a member of class TInplaceEditList.

10.1.6.3.18 TInplaceEditList.TrackButton Method

Pascal

```
procedure TrackButton(X: Integer; Y: Integer);
```

C++

```
__fastcall TrackButton(int X, int Y);
```

Description

This is TrackButton, a member of class TInplaceEditList.

10.1.6.3.19 TInplaceEditList.UpdateContents Method

Pascal

```
procedure UpdateContents; override;
```

C++

```
virtual __fastcall UpdateContents();
```

Description

This is UpdateContents, a member of class TInplaceEditList.

10.1.6.3.20 TInplaceEditList.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TInplaceEditList.

10.1.6.4 TInplaceEditList Properties

The properties of the TInplaceEditList class are listed here.

Public Properties

	Name	Description
	ActiveList (see page 3178)	This is ActiveList, a member of class TInplaceEditList.
	ButtonWidth (see page 3178)	This is ButtonWidth, a member of class TInplaceEditList.
	DropDownRows (see page 3178)	This is DropDownRows, a member of class TInplaceEditList.
R	EditStyle (see page 3178)	This is EditStyle, a member of class TInplaceEditList.
	ListVisible (see page 3179)	This is ListVisible, a member of class TInplaceEditList.
R	PickList (see page 3179)	This is PickList, a member of class TInplaceEditList.
	PickListLoaded (see page 3179)	This is PickListLoaded, a member of class TInplaceEditList.
R	Pressed (see page 3179)	This is Pressed, a member of class TInplaceEditList.

10.1.6.4.1 TInplaceEditList.ActiveList Property

Pascal

```
property ActiveList: TWinControl;
```

C++

```
__property TWinControl * ActiveList;
```

Description

This is ActiveList, a member of class TInplaceEditList.

10.1.6.4.2 TInplaceEditList.ButtonWidth Property

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

This is ButtonWidth, a member of class TInplaceEditList.

10.1.6.4.3 TInplaceEditList.DropDownRows Property

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

This is DropDownRows, a member of class TInplaceEditList.

10.1.6.4.4 TInplaceEditList.EditStyle Property

Pascal

```
property EditStyle: TEditStyle;
```

C++

```
__property TEditStyle EditStyle;
```

Description

This is EditStyle, a member of class TInplaceEditList.

10.1.6.4.5 TInplaceEditList.ListVisible Property

Pascal

```
property ListVisible: Boolean;
```

C++

```
__property Boolean ListVisible;
```

Description

This is ListVisible, a member of class TInplaceEditList.

10.1.6.4.6 TInplaceEditList.PickList Property

Pascal

```
property PickList: TCustomListbox;
```

C++

```
__property TCustomListbox PickList;
```

Description

This is PickList, a member of class TInplaceEditList.

10.1.6.4.7 TInplaceEditList.PickListLoaded Property

Pascal

```
property PickListLoaded: Boolean;
```

C++

```
__property Boolean PickListLoaded;
```

Description

This is PickListLoaded, a member of class TInplaceEditList.

10.1.6.4.8 TInplaceEditList.Pressed Property

Pascal

```
property Pressed: Boolean;
```

C++

```
__property Boolean Pressed;
```

Description

This is Pressed, a member of class TInplaceEditList.

10.1.6.5 TInplaceEditList Events

The events of the TInplaceEditList class are listed here.

Public Events

	Name	Description
	OnEditButtonClick (see page 3180)	This is OnEditButtonClick, a member of class TInplaceEditList.
	OnGetPickListitems (see page 3180)	This is OnGetPickListitems, a member of class TInplaceEditList.

10.1.6.5.1 TInplaceEditList.OnEditButtonClick Event

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

This is OnEditButtonClick, a member of class TInplaceEditList.

10.1.6.5.2 TInplaceEditList.OnGetPickListitems Event

Pascal

```
property OnGetPickListitems: TOnGetPickListItems;
```

C++

```
__property TOnGetPickListItems OnGetPickListitems;
```

Description

This is OnGetPickListitems, a member of class TInplaceEditList.

10.1.7 TStringGrid Class

TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.

Class Hierarchy

Pascal

```
TStringGrid = class(TDrawGrid);
```

C++

```
class TStringGrid : public TDrawGrid;
```

File

Grids

Description

Add a TStringGrid object to a form to present textual data in a tabular format. TStringGrid provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TStringGrid introduces the ability to associate an object with each string in the grid. These objects can encapsulate any information or behavior represented by the strings that are presented to the user.

If the strings to be presented in a grid represent field values from the records in a dataset, use TDBGrid instead.

See Also

TDBGrid

TValueListEditor

String Grids

Grids

10.1.7.1 TStringGrid Members

The following tables list the members exposed by TStringGrid.

Protected Events

	Name	Description
	OnColumnMoved (see page 3019)	Occurs immediately after the position of a column changes.
	OnDrawCell (see page 3020)	Occurs when a cell in the grid needs to be drawn.
	OnGetEditMask (see page 3021)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3022)	Occurs when the in-place editor requests the value of a cell.
	OnRowMoved (see page 3023)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3025)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3026)	Occurs when the user edits the value of a cell.
	OnTopLeftChanged (see page 3027)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
V	ColumnMoved (see page 3186)	Generates an OnColumnMoved event.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.

	CreateEditor (see page 3040)	Creates the inplace editor control.
	CreateParams (see page 3041)	Initializes the window-creation parameter record.
	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.
	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
	DeleteRow (see page 3042)	Deletes a specified from the grid.
	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
	DrawCell (see page 3187)	Generates an OnDrawCell event.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3004)	Generates an OnGetEditMask (see page 3021) event.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 3187)	Generates an OnGetEditText event.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	KeyDown (see page 3050)	Provides special processing when the user presses a key.
	KeyPress (see page 3050)	Provides special processing when the user presses a key.
	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3187)	Generates an OnRowMoved event.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 3006)	Determines whether a particular cell in the grid can be selected.
	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 3188)	Generates an OnSetEditText event.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3007)	Generates an OnTopLeftChanged (see page 3027) event.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.

	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	BorderStyle (see page 3060)	Determines whether a single line border is drawn around the grid.
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	Color (see page 3063)	Specifies the background color of the control.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Options (see page 3072)	Specifies various display and behavioral properties of the grid.
	ParentColor (see page 3073)	Determines where a control looks for its color information.
	Row (see page 3074)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	ScrollBars (see page 3077)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Public Methods

	Name	Description
	CellRect (see page 3003)	Returns the screen coordinates of a cell in the grid.
V	Create (see page 3185)	Creates an instance of TStringGrid (see page 3180).
V	Destroy (see page 3186)	Destroys an instance of TStringGrid (see page 3180).
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 3005)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

Public Properties

	Name	Description
	Canvas (see page 3007)	Specifies the TCanvas object that presents a drawing surface for the control.
	Cells (see page 3188)	Lists the strings for each cell in the grid.
	Col (see page 3009)	Specifies the index of the column that contains the selected cell.
	Cols (see page 3189)	Lists strings with their associated objects for each column.
	ColWidths (see page 3011)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 3011)	Determines whether the current cell can be edited.
	GridHeight (see page 3012)	Specifies the height of the grid in pixels.
	GridWidth (see page 3012)	Specifies the width of the grid in pixels.

	LeftCol (see page 3012)	Specifies the index of the first visible scrollable column in the grid.
	Objects (see page 3190)	Lists the objects for each cell in the grid.
	Row (see page 3013)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 3014)	Gives the height (in pixels) of all rows in the grid.
	Rows (see page 3190)	Lists the strings and their associated objects for each row.
	Selection (see page 3015)	Indicates the boundaries of the current selection.
	TabStops (see page 3017)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3017)	Specifies the index of the first visible scrollable row in the grid.

Published Events

	Name	Description
	OnClick (see page 3125)	Occurs when the user clicks the control.
	OnColumnMoved (see page 3127)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 3127)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3128)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 3128)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3131)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 3134)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 3135)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3135)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 3136)	Occurs when a control receives the input focus.
	OnExit (see page 3139)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 3142)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 3142)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 3142)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 3146)	Occurs when a key is pressed.
	OnKeyUp (see page 3147)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 3149)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 3150)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 3152)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 3152)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 3153)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 3155)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 3157)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 3157)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 3158)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 3158)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 3159)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 3159)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3160)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 3160)	Occurs immediately after the TopRow (see page 3017) property or the LeftCol (see page 3012) property changes.

Published Properties

	Name	Description
	Align (see page 3088)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3092)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 3092)	Specifies which edges of the control are beveled.

	BevelInner (see page 3093)	Specifies the cut of the inner bevel.
	BevelKind (see page 3093)	Specifies the control's bevel style.
	BevelOuter (see page 3094)	Specifies the cut of the outer bevel.
	BevelWidth (see page 3094)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 3095)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 3095)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 3095)	Specifies the number of columns in the grid.
	Color (see page 3096)	Specifies the background color of the control.
	Constraints (see page 3097)	Specifies the size constraints for the control.
	Ctl3D (see page 3098)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 3098)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3100)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3100)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DoubleBuffered (see page 3101)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 3102)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3103)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3103)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 3105)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 3105)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3106)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3106)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 3107)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 3108)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Options (see page 3109)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 3111)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 3111)	Determines where a control looks for its color information.
	ParentCtl3D (see page 3112)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 3112)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 3112)	Determines where a control looks for its font information.
	ParentShowHint (see page 3113)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3113)	Identifies the pop-up menu associated with the control.
	RowCount (see page 3118)	Specifies the number of rows in the grid.
	ScrollBars (see page 3119)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 3120)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 3121)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 3079)	Determines if the user can tab to a control.
	Visible (see page 3122)	Determines whether the component appears onscreen.
	VisibleColCount (see page 3123)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 3124)	Indicates the number of scrollable rows visible in the grid.

10.1.7.2 TStringGrid.Create Constructor

Creates an instance of TStringGrid (see page 3180).

Pascal

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

C++

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

Description

Call Create to create an instance of TStringGrid (see page 3180) at runtime. For string grids placed on forms at design time, Create is called automatically.

AOwner is another component, typically a form, that becomes the grid's Owner. The Owner is responsible for freeing the grid.

See Also

Create (see page 3030)

10.1.7.3 TStringGrid.Destroy Destructor

Destroys an instance of TStringGrid (see page 3180).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStringGrid();
```

Description

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

Destroy frees the helper objects used to manage the array of strings and their associated objects.

See Also

System::TObject::Free

10.1.7.4 TStringGrid Methods

The methods of the TStringGrid class are listed here.

Protected Methods

	Name	Description
	ColumnMoved (see page 3186)	Generates an OnColumnMoved event.
	DrawCell (see page 3187)	Generates an OnDrawCell event.
	GetEditText (see page 3187)	Generates an OnGetEditText event.
	RowMoved (see page 3187)	Generates an OnRowMoved event.
	SetEditText (see page 3188)	Generates an OnSetEditText event.

10.1.7.4.1 TStringGrid.ColumnMoved Method

Generates an OnColumnMoved event.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. It overrides the inherited method to generate an

OnColumnMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnColumnMoved event.

See Also

OnColumnMoved (see page 3019)

10.1.7.4.2 TStringGrid.DrawCell Method

Generates an OnDrawCell event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell. Descendants of TCustomDrawGrid can override this method to draw the contents of the cell or to block the OnDrawCell event.

See Also

DefaultDrawing (see page 3066)

FixedCols (see page 3067)

FixedRows (see page 3068)

OnDrawCell (see page 3020)

10.1.7.4.3 TStringGrid.GetEditText Method

Generates an OnGetEditText event.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns the Value parameter set by the OnGetEditText event handler. If there is no OnGetEditText event handler, GetEditText returns an empty string.

See Also

OnGetEditText (see page 3022)

10.1.7.4.4 TStringGrid.RowMoved Method

Generates an OnRowMoved event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnRowMoved event.

See Also

OnRowMoved (see page 3023)

10.1.7.4.5 TStringGrid.SetEditText Method

Generates an OnSetEditText event.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(Longint ACol, Longint ARow, const UnicodeString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. SetEditText overrides the inherited parent class's method to generate an OnSetEditText event. Descendants of TCustomDrawGrid can override this method to update an internal representation of the cell's value, or to block the OnSetEditText event.

See Also

OnSetEditText (see page 3026)

10.1.7.5 TStringGrid Properties

The properties of the TStringGrid class are listed here.

Public Properties

	Name	Description
	Cells (see page 3188)	Lists the strings for each cell in the grid.
	Cols (see page 3189)	Lists strings with their associated objects for each column.
	Objects (see page 3190)	Lists the objects for each cell in the grid.
	Rows (see page 3190)	Lists the strings and their associated objects for each row.

10.1.7.5.1 TStringGrid.Cells Property

Lists the strings for each cell in the grid.

Pascal

```
property Cells [ACol, ARow: Integer]: string;
```

C++

```
__property UnicodeString Cells[int ACol, ARow];
```

Description

Use Cells to access the string within a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate

of the cell. The first row is row zero, and the first column is column zero.

The ColCount (see page 3095) and RowCount (see page 3118) property values define the size of the array of strings.

To access the objects associated with the strings in the Cells array, use the Objects (see page 3190) property.

C++ Examples:

```
/*
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I, J, K;

  K = 0;

  for (I = 0; I < StringGrid1->ColCount; I++)
    for (J = 0; J < StringGrid1->RowCount; J++)
      StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 3062)

Objects (see page 3190)

RowCount (see page 3075)

10.1.7.5.2 TStringGrid.Cols Property

Lists strings with their associated objects for each column.

Pascal

```
property Cols [Index: Integer]: TStrings;
```

C++

```
__property TStrings Cols[int Index];
```

Description

Use Cols to access all the strings for a single column, along with their associated objects. The number of strings and associated

objects is always equal to the value of the ColCount (see page 3095) property, the number of columns in the grid. The Index parameter is the number of the column, where the value of the first column in the grid is zero.

To get all the strings and objects for a row, rather than for a column, use the Rows (see page 3190) property.

See Also

ColCount (see page 3062)

Objects (see page 3190)

Rows (see page 3190)

10.1.7.5.3 TStringGrid.Objects Property

Lists the objects for each cell in the grid.

Pascal

```
property Objects [ACol, ARow: Integer]: TObject;
```

C++

```
__property TObject * Objects[int ACol, ARow];
```

Description

Use the Objects property to access the object associated with a particular cell. The ColCount (see page 3095) and RowCount (see page 3118) values define the size of the array of objects. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell, where the first column is column 0, and the first row is row 0.

Note: The string grid does not own the objects in the Objects array. Objects added to the Objects array still exist even if the string grid is destroyed. They must be explicitly destroyed by the application.

See Also

Cells (see page 3188)

ColCount (see page 3062)

RowCount (see page 3075)

10.1.7.5.4 TStringGrid.Rows Property

Lists the strings and their associated objects for each row.

Pascal

```
property Rows [Index: Integer]: TStrings;
```

C++

```
__property TStrings Rows[int Index];
```

Description

Use Rows to access all the strings for a single row, along with their associated objects. The number of strings and associated objects is always equal to the value of the RowCount (see page 3118) property, the number of rows in the grid. The Index parameter is the number of the row, where the value of the first row in the grid is zero.

To get all the strings and objects for a column, rather than for a row, use the Cols (see page 3189) property.

C++ Examples:

```
/*
This example displays a string grid, a list box, and a
```

```
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items = StringGrid1->Rows[StringGrid1->Row];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{

  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";

}
```

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

ColCount (see page 3062)

Cols (see page 3189)

Objects (see page 3190)

10.1.8 TStringGridStrings Class

TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

Class Hierarchy

Pascal

```
TStringGridStrings = class(TStrings);
```

C++

```
class TStringGridStrings : public TStrings;
```

File

Grids

Description

Do not use TStringGridStrings directly in an application. It is tailored to work with a TStringGrid (see page 3180) object, and is created by the string grid to represent the strings (and associated objects) in a row or column of the grid.

See Also

TStringGrid (see page 3180)

TStringList

10.1.8.1 TStringGridStrings Members

The following tables list the members exposed by TStringGridStrings.

Protected Methods

	Name	Description
	Get (see page 3195)	Returns a string given its index.
	GetCount (see page 3195)	Returns the number of strings in the list
	GetObject (see page 3196)	Returns the object associated with the string at a specified index.
	Put (see page 3197)	Changes the value of the string with a specified index.
	PutObject (see page 3197)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 3197)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 3193)	Replaces the first empty string with a specified string.
	Assign (see page 3194)	Copies the list of strings and objects from another TStrings object.
	Clear (see page 3194)	Replaces all strings in the list with empty strings.
	Create (see page 3193)	Creates an instance of TStringGridStrings (see page 3192).

	Delete (see page 3195)	Prevents strings from being deleted.
	Insert (see page 3196)	Prevents strings from being inserted.

10.1.8.2 TStringGridStrings.Create Constructor

Creates an instance of TStringGridStrings (see page 3192).

Pascal

```
constructor Create(AGrid: TStringGrid; AIndex: Longint);
```

C++

```
__fastcall TStringGridStrings(TStringGrid AGrid, Longint AIndex);
```

Description

Do not create instances of TStringGridStrings (see page 3192). String grids create TStringGridStrings (see page 3192) objects for every row and column, to allow quick access of the strings and objects associated with the cells in the grid.

The AGrid parameter specifies the string grid that will use the TStringGridStrings (see page 3192) object. The AIndex parameter specifies the index of the row or column associated with the TStringGridStrings (see page 3192) object. If AIndex is greater than 0, the TStringGridStrings (see page 3192) object represents the row indexed by AIndex. If AIndex is less than 0, the TStringGridStrings (see page 3192) object represents the column indexed by the absolute value of AIndex. If AIndex is 0, the TStringGridStrings (see page 3192) object represents an empty row or column.

See Also

TStringGrid (see page 3180)

10.1.8.3 TStringGridStrings Methods

The methods of the TStringGridStrings class are listed here.

Protected Methods

	Name	Description
	Get (see page 3195)	Returns a string given its index.
	GetCount (see page 3195)	Returns the number of strings in the list
	GetObject (see page 3196)	Returns the object associated with the string at a specified index.
	Put (see page 3197)	Changes the value of the string with a specified index.
	PutObject (see page 3197)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 3197)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 3193)	Replaces the first empty string with a specified string.
	Assign (see page 3194)	Copies the list of strings and objects from another TStrings object.
	Clear (see page 3194)	Replaces all strings in the list with empty strings.
	Delete (see page 3195)	Prevents strings from being deleted.
	Insert (see page 3196)	Prevents strings from being inserted.

10.1.8.3.1 TStringGridStrings.Add Method

Replaces the first empty string with a specified string.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const UnicodeString S);
```

Description

Call Add to add a string to the list. Add searches the list for an empty string and replaces it with S. It then returns the index where it added S to the list. If there are no empty strings in the list, Add does not add S to the list and returns –1.

See Also

AddStrings

Append

Count

IndexOf

Insert (see page 3196)

10.1.8.3.2 TStringGridStrings.Assign Method

Copies the list of strings and objects from another TStrings object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the strings and objects from a source object of type TStrings. If the source object has more strings than the number of cells in the row or column represented by the TStringGridStrings (see page 3192) object, Assign only copies the first Count strings and objects.

If the Source parameter is not a TStrings object, Assign calls the inherited method.

See Also

Count

Objects

Strings

10.1.8.3.3 TStringGridStrings.Clear Method

Replaces all strings in the list with empty strings.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to remove all strings from the list, including any references to associated objects. Clear replaces every string with an

empty string and every object reference with nil (Delphi) or NULL (C++).

Note that Clear does not free the objects that are associated with strings in the list.

See Also

Delete (see page 3195)

Objects

Strings

10.1.8.3.4 TStringGridStrings.Delete Method

Prevents strings from being deleted.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Because TStringGridStrings (see page 3192) objects represent the rows or columns of a string grid, the number of entries in the list must correspond exactly to the number of cells in the corresponding row or column. Rather than allow entries to be deleted, destroying this correspondence, Delete raises an EInvalidGridOperation (see page 2999) exception.

See Also

EInvalidGridOperation (see page 2999)

Clear (see page 3194)

10.1.8.3.5 TStringGridStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property.

In TStrings Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings

10.1.8.3.6 TStringGridStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TStrings GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count

10.1.8.3.7 TStringGridStrings.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property.

Index is the index of the string with which the object is associated.

In TStrings, GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects

10.1.8.3.8 TStringGridStrings.Insert Method

Prevents strings from being inserted.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S);
```

Description

Because TStringGridStrings (see page 3192) objects represent the rows or columns of a string grid, the number of entries in the list must correspond exactly to the number of cells in the corresponding row or column. Rather than allow entries to be inserted, destroying this correspondence, Insert raises an EInvalidGridOperation (see page 2999) exception.

See Also

EInvalidGridOperation (see page 2999)

Add (see page 3193)

10.1.8.3.9 TStringGridStrings.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings

Objects

10.1.8.3.10 TStringGridStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property.

As implemented in TStrings, PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects

10.1.8.3.11 TStringGridStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the

TStrings object is not already in the middle of an update, TStrings calls SetUpdateState internally, with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, TStrings calls SetUpdateState internally, with Updating set to false.

As implemented in TStrings, SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate

EndUpdate

10.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEditStyle (see page 3199)	TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.
	TGridOption (see page 3201)	TGridOption and TGridOptions (see page 3207) customize the appearance and behavior of a grid control.
	TGridState (see page 3203)	TGridState indicates any user actions that are currently changing the layout of the grid.

Records

	Name	Description
	TGridAxisDrawInfo (see page 3200)	TGridAxisDrawInfo and TGridDrawInfo (see page 3201) represent information that is useful when rendering a grid.
	TGridCoord (see page 3200)	TGridCoord specifies a cell in a grid control.
	TGridDrawInfo (see page 3201)	This is record Grids.TGridDrawInfo.
	TGridRect (see page 3203)	TGridRect specifies a set of cells that occupy a rectangular region in a grid control.

10.2.1 Grids.TEditStyle Enumeration

TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.

Pascal

```
TEditStyle = (
  esSimple,
  esEllipsis,
  esPickList
);
```

C++

```
enum TEditStyle {
  esSimple,
  esEllipsis,
  esPickList
};
```

File

Grids

Description

TEditStyle describes the buttons, if any, that can appear in the cell of a grid. It can be any of the following values:

Value	Meaning
esSimple	The cell acts as a simple edit control where users can type a new value.
esEllipsis	The cell contains an ellipsis button that users can click to launch a dialog.
esPickList	The cell contains a drop-down arrow button that users can click to display a pick list of possible values.

10.2.2 Grids.TGridAxisDrawInfo Record

TGridAxisDrawInfo and TGridDrawInfo (see page 3201) represent information that is useful when rendering a grid.

Pascal

```
TGridAxisDrawInfo = record
  EffectiveLineWidth: Integer;
  FixedBoundary: Integer;
  GridBoundary: Integer;
  GridExtent: Integer;
  LastFullVisibleCell: Longint;
  FullVisBoundary: Integer;
  FixedCellCount: Integer;
  FirstGridCell: Integer;
  GridCellCount: Integer;
  GetExtent: TGetExtentsFunc;
end;
```

C++

```
struct TGridAxisDrawInfo {
  int EffectiveLineWidth;
  int FixedBoundary;
  int GridBoundary;
  int GridExtent;
  Longint LastFullVisibleCell;
  int FullVisBoundary;
  int FixedCellCount;
  int FirstGridCell;
  int GridCellCount;
  TGetExtentsFunc GetExtent;
};
```

File

Grids

Description

TGridDrawInfo (see page 3201) encapsulates information that reflects the layout of the grid axes. Each axis is described by a TGridAxisDrawInfo value, which encodes information about the size of the axis and the type and number of cells along that axis.

10.2.3 Grids.TGridCoord Record

TGridCoord specifies a cell in a grid control.

Pascal

```
TGridCoord = record
  X: Longint;
  Y: Longint;
end;
```

C++

```
struct TGridCoord {
  Longint X;
  Longint Y;
};
```

File

Grids

Description

TGridCoord indicates a specific cell in a grid.

X indicates the column of the grid, where 0 specifies the first column, 1 specifies the second column, and so on.

Y indicates the row of the grid, where 0 specifies the first row, 1 specifies the second row, and so on.

10.2.4 Grids.TGridDrawInfo Record

Pascal

```
TGridDrawInfo = record
  Horz: TGridAxisDrawInfo;
  Vert: TGridAxisDrawInfo;
end;
```

C++

```
struct TGridDrawInfo {
  TGridAxisDrawInfo Horz;
  TGridAxisDrawInfo Vert;
};
```

File

Grids

Description

This is record Grids.TGridDrawInfo.

10.2.5 Grids.TGridOption Enumeration

TGridOption and TGridOptions (see page 3207) customize the appearance and behavior of a grid control.

Pascal

```
TGridOption = (
  goFixedVertLine,
  goFixedHorzLine,
  goVertLine,
  goHorzLine,
  goRangeSelect,
  goDrawFocusSelected,
  goRowSizing,
  goColSizing,
  goRowMoving,
  goColMoving,
  goEditing,
  goTabs,
  goRowSelect,
  goAlwaysShowEditor,
  goThumbTracking
);
```

C++

```
enum TGridOption {
  goFixedVertLine,
  goFixedHorzLine,
  goVertLine,
  goHorzLine,
  goRangeSelect,
  goDrawFocusSelected,
  goRowSizing,
  goColSizing,
  goRowMoving,
  goColMoving,
  goEditing,
  goTabs,
  goRowSelect,
  goAlwaysShowEditor,
  goThumbTracking
};
```

File

Grids

Description

TGridOptions (see page 3207) is a set of TGridOption values. The following table lists the possible TGridOption values:

Value	Meaning
goFixedVertLine	Vertical lines are drawn to separate the fixed (nonscrolling) columns in the grid.
goFixedHorzLine	Horizontal lines are drawn to separate the fixed (nonscrolling) rows in the grid.
goVertLine	Vertical lines are drawn to separate the scrollable columns in the grid.
goHorzLine	Horizontal lines are drawn to separate the scrollable rows in the grid.
goRangeSelect	Users can select ranges of cells at one time. goRangeSelect is ignored if Options includes goEditing.
goDrawFocusSelected	Cells with input focus are drawn with a special highlight color, just like selected cells without input focus. If goDrawFocusSelected is not included, the cell with input focus is distinguished by a focus rectangle, not by a special background color.
goRowSizing	Scrollable rows can be individually resized.
goColSizing	Scrollable columns can be individually resized.
goRowMoving	Scrollable rows can be moved using the mouse.
goColMoving	Scrollable columns can be moved using the mouse.
goEditing	Users can edit the contents of cells. When goEditing is included in Options, goRangeSelect has no effect.
goTabs	Users can navigate through the cells in the grid using Tab and Shift+Tab.
goRowSelect	Entire rows are selected rather than individual cells. If goRowSelect is included in Options, goAlwaysShowEditor has no effect.
goAlwaysShowEditor	The grid is locked into edit mode. The user does not need to use Enter or F2 to turn on EditorMode. If Options does not include goEditing, goAlwaysShowEditor has no effect. If Options includes goRowSelect, goAlwaysShowEditor has no effect.
goThumbTracking	The grid image updates while the user is dragging the thumb of the scroll bar. If goThumbTracking is not included, the image does not update until the user releases the thumb in a new position.

10.2.6 Grids.TGridRect Record

TGridRect specifies a set of cells that occupy a rectangular region in a grid control.

Pascal

```
TGridRect = record
  case Integer of
    0: (Bottom: Longint;
        Left: Longint;
        Right: Longint;
        Top: Longint;);
    1: (BottomRight: TGridCoord;
        TopLeft: TGridCoord;);
end;
```

C++

```
struct TGridRect {
  Longint Bottom;
  Longint Left;
  Longint Right;
  Longint Top;
  TGridCoord BottomRight;
  TGridCoord TopLeft;
};
```

File

Grids

Description

TGridRect indicates a rectangular set of cells from the grid in one of two ways:

Using the first method, TopLeft indicates the coordinates of the top-left cell in the set and BottomRight indicates the coordinates of the bottom-right cell.

Using the second method, Left indicates the first column in the set and Right the last column in the set, where 0 is the first column of the grid, 1 is the second column, and so on. Top indicates the first row in the set and Bottom the last row, where 0 is the first row of the grid, 1 the second column, and so on.

10.2.7 Grids.TGridState Enumeration

TGridState indicates any user actions that are currently changing the layout of the grid.

Pascal

```
TGridState = (
  gsNormal,
  gsSelecting,
  gsRowSizing,
  gsColSizing,
  gsRowMoving,
  gsColMoving
);
```

C++

```
enum TGridState {
```

```
    gsNormal,
    gsSelecting,
    gsRowSizing,
    gsColSizing,
    gsRowMoving,
    gsColMoving
  };
```

File

Grids

Description

TGridState represents state information about how the grid layout is changing. The following table lists the possible values:

Value	Meaning
gsNormal	The grid layout is not changing.
gsSelecting	The user is selecting a cell or row.
gsRowSizing	The user is changing the height of a row.
gsColSizing	The user is changing the width of a column.
gsRowMoving	The user is dragging a row to a new position.
gsColMoving	The user is dragging a column to a new position.

10.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDrawCellEvent (see page 3205)	Function type for OnDrawCell event.
TGetEditEvent (see page 3206)	TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.
TGetExtentsFunc (see page 3206)	This is type Grids.TGetExtentsFunc.
TGridDrawState (see page 3206)	TGridDrawState represents information that is useful when rendering the cell of a grid.
TGridMovement (see page 3207)	This is type Grids.TGridMovement.
TGridOptions (see page 3207)	TGridOptions and TGridOption (see page 3201) customize the appearance and behavior of a grid control.
TGridScrollDirection (see page 3209)	TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.
TMovedEvent (see page 3210)	This is type Grids.TMovedEvent.
TOnGetPickListItems (see page 3210)	This is type Grids.TOnGetPickListItems.
TSelectCellEvent (see page 3210)	This is type Grids.TSelectCellEvent.
TSetEditEvent (see page 3211)	TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

10.3.1 Grids.TDrawCellEvent Type

Function type for OnDrawCell event.

Pascal

```
TDrawCellEvent = procedure (Sender: TObject; ACol, ARow: Longint; Rect: TRect; State:
TGridDrawState) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; Rect: TRect; State: TGridDrawState) ( TDrawCellEvent)();
```

File

Grids

Description

TDrawCellEvent is the function type for a OnDrawCell event. Its parameters are described in this table.

Parameter	Meaning
Sender	Grid in which cell is being drawn.
ACol, ARow	Column and row indexes of the cell being drawn.
Rect	Location of cell on canvas.
State	TGridDrawState (see page 3206) that indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

See Also

OnDrawCell (see page 3020)

TGridDrawState (see page 3206)

10.3.2 Grids.TGetEditEvent Type

TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.

Pascal

```
TGetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Value: string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; var Value: string) ( TGetEditEvent)();
```

File

Grids

Description

Sender is the grid whose cell is being edited.

ACol and ARow indicate the column and row indexes of the cell that is being edited.

Value is the string requested by the inplace editor for that cell. The event handler supplies this string.

10.3.3 Grids.TGetExtentsFunc Type

Pascal

```
TGetExtentsFunc = function (Index: Longint): Integer of object;
```

C++

```
(Index: Longint): Integer ( TGetExtentsFunc)();
```

File

Grids

Description

This is type Grids.TGetExtentsFunc.

10.3.4 Grids.TGridDrawState Type

TGridDrawState represents information that is useful when rendering the cell of a grid.

Pascal

```
TGridDrawState = set of (gdSelected, gdFocused, gdFixed);
```

C++

```
(gdSelected, gdFocused, gdFixed) TGridDrawState;
```

File

Grids

Description

TGridDrawState represents state information about the cell of a grid that is being rendered. It is a set that includes 0 or more of the following:

Value	Meaning
gdSelected	The cell is currently selected.
gdFocused	The cell has input focus.
gdFixed	The cell is in the fixed region of the grid.

10.3.5 Grids.TGridMovement Type

Pascal

```
TGridMovement = gsRowMoving..gsColMoving;
```

C++

```
gsRowMoving..gsColMoving TGridMovement;
```

File

Grids

Description

This is type Grids.TGridMovement.

10.3.6 Grids.TGridOptions Type

TGridOptions and TGridOption (see page 3201) customize the appearance and behavior of a grid control.

Pascal

```
TGridOptions = set of TGridOption;
```

C++

```
TGridOption TGridOptions;
```

File

Grids

Description

TGridOptions is a set of TGridOption (see page 3201) values. The following table lists the possible TGridOption (see page 3201) values:

Value	Meaning
goFixedVertLine	Vertical lines are drawn to separate the fixed (nonscrolling) columns in the grid.
goFixedHorzLine	Horizontal lines are drawn to separate the fixed (nonscrolling) rows in the grid.
goVertLine	Vertical lines are drawn to separate the scrollable columns in the grid.
goHorzLine	Horizontal lines are drawn to separate the scrollable rows in the grid.
goRangeSelect	Users can select ranges of cells at one time. goRangeSelect is ignored if Options includes goEditing.
goDrawFocusSelected	Cells with input focus are drawn with a special highlight color, just like selected cells without input focus. If goDrawFocusSelected is not included, the cell with input focus is distinguished by a focus rectangle, not by a special background color.
goRowSizing	Scrollable rows can be individually resized.
goColSizing	Scrollable columns can be individually resized.
goRowMoving	Scrollable rows can be moved using the mouse.
goColMoving	Scrollable columns can be moved using the mouse.
goEditing	Users can edit the contents of cells. When goEditing is included in Options, goRangeSelect has no effect.
goTabs	Users can navigate through the cells in the grid using Tab and Shift+Tab.
goRowSelect	Entire rows are selected rather than individual cells. If goRowSelect is included in Options, goAlwaysShowEditor has no effect.
goAlwaysShowEditor	The grid is locked into edit mode. The user does not need to use Enter or F2 to turn on EditorMode. If Options does not include goEditing, goAlwaysShowEditor has no effect. If Options includes goRowSelect, goAlwaysShowEditor has no effect.
goThumbTracking	The grid image updates while the user is dragging the thumb of the scroll bar. If goThumbTracking is not included, the image does not update until the user releases the thumb in a new position.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
```

```
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

10.3.7 Grids.TGridScrollDirection Type

TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.

Pascal

```
TGridScrollDirection = set of (sdLeft, sdRight, sdUp, sdDown);
```

C++

```
(sdLeft, sdRight, sdUp, sdDown) TGridScrollDirection;
```

File

Grids

Description

TGridScrollDirection is a set of one or two values that indicate the direction in which to scroll a grid. The set contains at least one value and at most one horizontal and one vertical value. The following table lists the possible values:

Value	Meaning
sdLeft	Scroll cells horizontally to the left.
sdRight	Scroll cells horizontally to the right.
sdUp	Scroll cells vertically up.
sdDown	Scroll cells vertically down.

See Also

TCustomGrid

10.3.8 Grids.TMovedEvent Type

Pascal

```
TMovedEvent = procedure (Sender: TObject; FromIndex, ToIndex: Longint) of object;
```

C++

```
(Sender: TObject; FromIndex, ToIndex: Longint) ( TMovedEvent)();
```

File

Grids

Description

This is type Grids.TMovedEvent.

10.3.9 Grids.TOnGetPickListItems Type

Pascal

```
TOnGetPickListItems = procedure (ACol, ARow: Integer; Items: TStrings) of object;
```

C++

```
(ACol, ARow: Integer; Items: TStrings) ( TOnGetPickListItems)();
```

File

Grids

Description

This is type Grids.TOnGetPickListItems.

10.3.10 Grids.TSelectCellEvent Type

Pascal

```
TSelectCellEvent = procedure (Sender: TObject; ACol, ARow: Longint; var CanSelect: Boolean) of
object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; var CanSelect: Boolean) ( TSelectCellEvent)();
```

File

Grids

Description

This is type Grids.TSelectCellEvent.

10.3.11 Grids.TSetEditEvent Type

TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

Pascal

```
TSetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; const Value: string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; const Value: string) ( TSetEditEvent)();
```

File

Grids

Description

Sender is the grid whose cell was just edited.

ACol and ARow indicate the column and row indexes of the cell that was edited.

Value is the string to assign to the cell.

10.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxCustomExtents (see page 3212)	This is constant Grids.MaxCustomExtents.
MaxShortInt (see page 3212)	This is constant Grids.MaxShortInt.

10.4.1 Grids.MaxCustomExtents Constant

Pascal

```
MaxCustomExtents = MaxListSize;
```

C++

```
MaxCustomExtents = MaxListSize;
```

File

Grids

Description

This is constant Grids.MaxCustomExtents.

10.4.2 Grids.MaxShortInt Constant

Pascal

```
MaxShortInt = High(ShortInt);
```

C++

```
MaxShortInt = High(ShortInt);
```

File

Grids

Description

This is constant Grids.MaxShortInt.

11 HelpIntfs Namespace

This is namespace HelpIntfs.

Topics

Name	Description
Classes (see page 3214)	The following table lists classes in this documentation.
Interfaces (see page 3215)	The following table lists interfaces in this documentation.
Functions (see page 3242)	The following table lists functions in this documentation.

Classes

	Name	Description
	EHelpSystemException (see page 3214)	EHelpSystemException is the class for errors raised by the RTL Help System.

Functions

	Name	Description
	GetHelpSystem (see page 3242)	Returns Help System object.
	GetHelpSystem (see page 3242)	Returns Help System object.
	GetHelpSystem (see page 3242)	Returns Help System object.
	RegisterViewer (see page 3242)	Register Help Viewer to receive requests.

Interfaces

	Name	Description
	ICustomHelpViewer (see page 3215)	Help Viewer access for Help System..
	IExtendedHelpViewer (see page 3219)	Help Viewer with context help support.
	IExtendedHelpViewer2 (see page 3222)	Help Viewer with ability to open Index and Search panes, as well as open Topics with Anchors.
	IFilteredHelpViewer (see page 3224)	Help Viewer with ability to Set and Get help filters.
	IHelpManager (see page 3226)	IHelpManager provides access to the Help System for a Help Viewer.
	IHelpSelector (see page 3228)	Interface for Help Selector object.
	IHelpSelector2 (see page 3230)	This is class HelpIntfs.IHelpSelector2.
	IHelpSystem (see page 3231)	Application Help System that supports multiple Help Viewers.
	IHelpSystem2 (see page 3234)	Application Help System that supports multiple Help Viewers.
	IHelpSystem3 (see page 3235)	Application Help System that supports multiple Help Viewers.
	IHelpSystemFlags (see page 3238)	This is class HelpIntfs.IHelpSystemFlags.
	ISpecialWinHelpViewer (see page 3239)	Help Viewer that supports special Windows Help requests.

11.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EHelpSystemException (see page 3214)	EHelpSystemException is the class for errors raised by the RTL Help System.

11.1.1 EHelpSystemException Class

EHelpSystemException is the class for errors raised by the RTL Help System.

Class Hierarchy

Pascal

```
EHelpSystemException = class(Exception);
```

C++

```
class EHelpSystemException : public Exception;
```

File

HelpIntfs

Description

The Run-Time Library Help System raises EHelpSystemException when an error occurs while registering or invoking a help viewer.

11.2 Interfaces

The following table lists interfaces in this documentation.

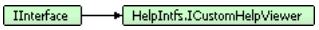
Interfaces

	Name	Description
	ICustomHelpViewer (see page 3215)	Help Viewer access for Help System..
	IExtendedHelpViewer (see page 3219)	Help Viewer with context help support.
	IExtendedHelpViewer2 (see page 3222)	Help Viewer with ability to open Index and Search panes, as well as open Topics with Anchors.
	IFilteredHelpViewer (see page 3224)	Help Viewer with ability to Set and Get help filters.
	IHelpManager (see page 3226)	IHelpManager provides access to the Help System for a Help Viewer.
	IHelpSelector (see page 3228)	Interface for Help Selector object.
	IHelpSelector2 (see page 3230)	This is class HelpIntfs.IHelpSelector2.
	IHelpSystem (see page 3231)	Application Help System that supports multiple Help Viewers.
	IHelpSystem2 (see page 3234)	Application Help System that supports multiple Help Viewers.
	IHelpSystem3 (see page 3235)	Application Help System that supports multiple Help Viewers.
	IHelpSystemFlags (see page 3238)	This is class HelpIntfs.IHelpSystemFlags.
	ISpecialWinHelpViewer (see page 3239)	Help Viewer that supports special Windows Help requests.

11.2.1 ICustomHelpViewer Interface

Help Viewer access for Help System..

Class Hierarchy



Pascal

```
ICustomHelpViewer = interface(IInterface);
```

C++

```
__interface ICustomHelpViewer : public IInterface;
```

File

HelpIntfs

Description

A Help Viewer is an object which handles application Help requests passed to it by the Help System on behalf of an application. A Help Viewer is typically a wrapper for an external application, such as a help engine or web browser.

The Help Viewer class must implement the ICustomHelpViewer interface, or one of its descendant interfaces, IExtendedHelpViewer (see page 3219) and ISpecialWinHelpViewer (see page 3239). The Help System uses methods from these interfaces to pass Help requests to the Help Viewer.

To use a Help Viewer, an application must call the global RegisterViewer (see page 3242) function.

Note: ISpecialWinHelpViewer (see page 3239) is only provided with Windows products. Do not use this type in cross-platform applications.

See Also

RegisterViewer (see page 3242)

Communicating with the Help Manager

Help System Interfaces

Implementing ICustomHelpViewer

11.2.1.1 ICustomHelpViewer Members

The following tables list the members exposed by ICustomHelpViewer.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.

11.2.1.2 ICustomHelpViewer Methods

The methods of the ICustomHelpViewer class are listed here.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.

11.2.1.2.1 ICustomHelpViewer.CanShowTableOfContents Method

Query support for Table of Contents.

Pascal

```
function CanShowTableOfContents: Boolean;
```

C++

```
__fastcall Boolean CanShowTableOfContents();
```

Description

The Help System uses CanShowTableOfContents to determine which Help Viewers know how to display a Table of Contents.

CanShowTableOfContents should return true if the Viewer is ready for a subsequent call to ShowTableOfContents (see page 3218). The Help System should not call ShowTableOfContents (see page 3218) for a Viewer after CanShowTableOfContents returns false; if this occurs, the Viewer should raise an EHelpSystemException (see page 3214).

See Also

ShowTableOfContents (see page 3233)

ShowTableOfContents (see page 3218)

EHelpSystemException (see page 3214)

11.2.1.1.2.2 ICustomHelpViewer.GetHelpStrings Method

Return Help Strings that match a keyword.

Pascal

```
function GetHelpStrings(const HelpString: string): TStringList;
```

C++

```
__fastcall TStringList GetHelpStrings(const UnicodeString HelpString);
```

Description

The Help System calls GetHelpStrings to obtain a list of Help Strings for topics that match a keyword-based Help request. GetHelpStrings should expect the keyword in the parameter and return the Help Strings in a TStringList object.

A Help String is an identifying string associated with a specific topic. The Help System requires that each of the topics provided by a given Viewer have a different Help String. The Help System does not require that two Viewers never use the same Help String, but an application should still avoid doing this. One strategy is to include the Viewer Name in the Help String.

The Help System uses Help Strings to provide the Help Selector with a merged list of Help topics that match a request. Typically, Help Strings are human-readable topic descriptions, and the Help Selector simply displays them to the user so the user can choose which topic will be displayed.

See Also

ShowHelp (see page 3233)

UnderstandsKeyword (see page 3219)

ShowHelp (see page 3218)

GetViewerName (see page 3217)

11.2.1.1.2.3 ICustomHelpViewer.GetViewerName Method

Return string identifying Help Viewer.

Pascal

```
function GetViewerName: string;
```

C++

```
__fastcall UnicodeString GetViewerName();
```

Description

The Help System calls GetViewerName to obtain a string identifying the Help Viewer. Every Help Viewer within an application must return a unique Viewer Name.

11.2.1.2.4 ICustomHelpViewer.NotifyID Method

Save Viewer's numeric ID.

Pascal

```
procedure NotifyID(const ViewerID: Integer);
```

C++

```
__fastcall NotifyID(const int ViewerID);
```

Description

The Help System calls NotifyID to pass the application a unique numeric ID associated with a Viewer. The application uses this ID to identify a specific viewer in calls to Release (IHelpManager (see page 3226)).

See Also

Release (see page 3227)

11.2.1.2.5 ICustomHelpViewer.ShowHelp Method

Show a specific Help topic.

Pascal

```
procedure ShowHelp(const HelpString: string);
```

C++

```
__fastcall ShowHelp(const UnicodeString HelpString);
```

Description

The Help System calls ShowHelp when it needs the Help Viewer to display the Help Topic identified by HelpString. The HelpString parameter is a value previously provided by the Help Viewer in response to a GetHelpStrings (see page 3217) call.

11.2.1.2.6 ICustomHelpViewer.ShowTableOfContents Method

Show a Table of Contents.

Pascal

```
procedure ShowTableOfContents;
```

C++

```
__fastcall ShowTableOfContents();
```

Description

The Help System calls ShowTableOfContents when it needs the Help Viewer to display a Table of Contents. The Help System does not call ShowTableOfContents for a Viewer unless that Viewer has previously returned true for a call to CanShowTableOfContents (see page 3216).

See Also

CanShowTableOfContents (see page 3216)

11.2.1.2.7 ICustomHelpViewer.ShutDown Method

Permanent Viewer shut down.

Pascal

```
procedure ShutDown;
```

C++

```
__fastcall ShutDown();
```

Description

The Help System calls ShutDown when it is notified that the application is shutting down. A Viewer should respond to this call by terminating any external applications used to display Help topics. After a call to ShutDown, a Viewer can assume that no more Help requests will arrive.

11.2.1.2.8 ICustomHelpViewer.SoftShutDown Method

Temporary Viewer shut down.

Pascal

```
procedure SoftShutDown;
```

C++

```
__fastcall SoftShutDown();
```

Description

The Help System calls SoftShutDown when it is notified that the application want to close all open Help Topics. A Viewer should respond to this call by terminating any external applications used to display Help topics, but should be prepared to restart them as needed.

11.2.1.2.9 ICustomHelpViewer.UnderstandsKeyword Method

Would respond to a keyword-based Help request.

Pascal

```
function UnderstandsKeyword(const HelpString: string): Integer;
```

C++

```
__fastcall int UnderstandsKeyword(const UnicodeString HelpString);
```

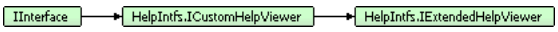
Description

The Help System calls UnderstandsKeyword to determine to determine if the Help Viewer can provide topics for a keyword-based Help request based on HelpString. UnderstandsKeyword should return the number of Help topics that would match such a request.

11.2.2 IExtendedHelpViewer Interface

Help Viewer with context help support.

Class Hierarchy



Pascal

```
IExtendedHelpViewer = interface(ICustomHelpViewer);
```

C++

```
__interface IExtendedHelpViewer : public ICustomHelpViewer;
```

File

HelpIntfs

Description

IExtendedHelpViewer extends ICustomHelpViewer (see page 3215) by adding support for Topic IDs and Context IDs.

A topic ID is a short string that uniquely identifies each topic provided by a Help Viewer. A topic ID provides a simple way for an application to display a specific Help topic without searching for it.

A context ID is an integer value that uniquely identifies each Help topic. Context IDs support context-sensitive Help by providing a value that associates forms and controls with their help topics. A form or control can be associated a topic simply by setting its HelpContext property.

See Also

RegisterViewer (see page 3242)

Help System Interfaces

11.2.2.1 IExtendedHelpViewer Members

The following tables list the members exposed by IExtendedHelpViewer.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayTopic (see page 3221)	Use Topic ID to display topic.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.2.2 IExtendedHelpViewer Methods

The methods of the IExtendedHelpViewer class are listed here.

Public Methods

	Name	Description
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayTopic (see page 3221)	Use Topic ID to display topic.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.2.2.1 IExtendedHelpViewer.DisplayHelpByContext Method

Use Context ID to display topic.

Pascal

```
procedure DisplayHelpByContext(const ContextID: Integer; const HelpFileName: string);
```

C++

```
__fastcall DisplayHelpByContext(const int ContextID, const UnicodeString HelpFileName);
```

Description

The RTL Help System calls DisplayHelpByContext to have the Viewer handle a context-based Help request. ContextID is the topic's Context ID. HelpFileName is the name of the file containing the topic. Note that some Help engines require the file name to be fully qualified.

See Also

UnderstandsTopic (see page 3222)

11.2.2.2.2 IExtendedHelpViewer.DisplayTopic Method

Use Topic ID to display topic.

Pascal

```
procedure DisplayTopic(const Topic: string);
```

C++

```
__fastcall DisplayTopic(const UnicodeString Topic);
```

Description

The RTL Help System calls DisplayTopic to have the Viewer handle a topic-based Help request. Topic is the Topic ID, a short string that must be unique for all the topics maintained by a particular Viewer.

See Also

UnderstandsTopic (see page 3222)

11.2.2.2.3 IExtendedHelpViewer.UnderstandsContext Method

Queries the Help Viewer if it understands the given Context ID.

Pascal

```
function UnderstandsContext(const ContextID: Integer; const HelpFileName: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsContext(const int ContextID, const UnicodeString HelpFileName);
```

Description

UnderstandsContext should return true if the Viewer is able to display the topic identified by ContextID in the file HelpFileName. When the RTL Help System receives a Context-based Help request, it calls UnderstandsContext for each Viewer that implements it until the method returns true. (Viewers are polled in the order they were registered.) The Help System than calls that Viewer's DisplayHelpByContext (see page 3221) method.

See Also

DisplayHelpByContext (see page 3221)

11.2.2.2.4 IExtendedHelpViewer.UnderstandsTopic Method

Queries the Help Viewer if it understands the given Topic.

Pascal

```
function UnderstandsTopic(const Topic: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsTopic(const UnicodeString Topic);
```

Description

UnderstandsTopic should return true if the Viewer is able to display the topic identified by the Topic parameter. When the RTL Help System receives a Topic-based Help request, it calls UnderstandsTopic for each Viewer that implements it until the method returns true. (Viewers are polled in the order they were registered.) The Help System than calls that Viewer's DisplayTopic (see page 3221) method.

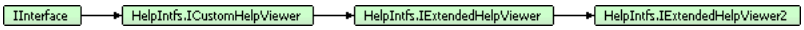
See Also

DisplayTopic (see page 3221)

11.2.3 IExtendedHelpViewer2 Interface

Help Viewer with ability to open Index and Search panes, as well as open Topics with Anchors.

Class Hierarchy



Pascal

```
IExtendedHelpViewer2 = interface(IExtendedHelpViewer);
```

C++

```
__interface IExtendedHelpViewer2 : public IExtendedHelpViewer;
```

File

HelpIntfs

Description

IExtendedHelpViewer2 extends IExtendedHelpViewer (see page 3219) by adding support for opening the Index and Search pane of the Help Viewer. It also allows you to specify an Anchor when opening up a specific topic.

See Also

RegisterViewer (see page 3242)

Help System Interfaces

11.2.3.1 IExtendedHelpViewer2 Members

The following tables list the members exposed by IExtendedHelpViewer2.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayIndex (see page 3223)	Opens Index pane of Help Viewer.
	DisplaySearch (see page 3223)	Opens Search pane of Help Viewer.
	DisplayTopic (see page 3224)	Use Topic ID to display topic, specifying page anchor.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.3.2 IExtendedHelpViewer2 Methods

The methods of the IExtendedHelpViewer2 class are listed here.

Public Methods

	Name	Description
	DisplayIndex (see page 3223)	Opens Index pane of Help Viewer.
	DisplaySearch (see page 3223)	Opens Search pane of Help Viewer.
	DisplayTopic (see page 3224)	Use Topic ID to display topic, specifying page anchor.

11.2.3.2.1 IExtendedHelpViewer2.DisplayIndex Method

Opens Index pane of Help Viewer.

Pascal

```
procedure DisplayIndex(const Topic: string);
```

C++

```
__fastcall DisplayIndex(const UnicodeString Topic);
```

Description

The RTL Help System calls DisplayIndex to have the Viewer open its Index Pane. If the Help Viewer has the ability to search for the given topic parameter, it executes the Index search; otherwise it is ignored.

See Also

DisplaySearch (see page 3223)

11.2.3.2.2 IExtendedHelpViewer2.DisplaySearch Method

Opens Search pane of Help Viewer.

Pascal

```
procedure DisplaySearch(const Topic: string);
```

C++

```
__fastcall DisplaySearch(const UnicodeString Topic);
```

Description

The RTL Help System calls DisplaySearch to have the Viewer open its Search Pane. If the Help Viewer has the ability to search for the given topic parameter, it executes the search; otherwise it is ignored.

See Also

DisplayIndex (see page 3223)

11.2.3.2.3 IExtendedHelpViewer2.DisplayTopic Method

Use Topic ID to display topic, specifying page anchor.

Pascal

```
procedure DisplayTopic(const Topic: string; const Anchor: string); overload;
```

C++

```
__fastcall DisplayTopic(const UnicodeString Topic, const UnicodeString Anchor);
```

Description

The RTL Help System calls DisplayTopic to have the Viewer handle a topic-based Help request and can specify a page anchor. If the page does not have the page anchor, the page is opened anyway.

Topic is the Topic ID, a short string that must be unique for all topics maintained by a particular Viewer.

Anchor is a tag that allows you to jump to a specific place in a page.

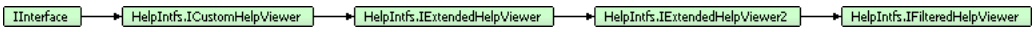
See Also

UnderstandsTopic (see page 3222)

11.2.4 IFilteredHelpViewer Interface

Help Viewer with ability to Set and Get help filters.

Class Hierarchy



Pascal

```
IFilteredHelpViewer = interface(IExtendedHelpViewer2);
```

C++

```
__interface IFilteredHelpViewer : public IExtendedHelpViewer2;
```

File

HelpIntfs

Description

IFilteredHelpViewer extends IExtendedHelpViewer2 (see page 3222) by adding support for getting and setting help filters.

See Also

RegisterViewer (see page 3242)

Help System Interfaces

11.2.4.1 IFilteredHelpViewer Members

The following tables list the members exposed by IFilteredHelpViewer.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayIndex (see page 3223)	Opens Index pane of Help Viewer.
	DisplaySearch (see page 3223)	Opens Search pane of Help Viewer.
	DisplayTopic (see page 3221)	Use Topic ID to display topic.
	DisplayTopic (see page 3224)	Use Topic ID to display topic, specifying page anchor.
	GetFilter (see page 3225)	Returns current help filter.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	SetFilter (see page 3226)	Sets current help filter.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.4.2 IFilteredHelpViewer Methods

The methods of the IFilteredHelpViewer class are listed here.

Public Methods

	Name	Description
	GetFilter (see page 3225)	Returns current help filter.
	SetFilter (see page 3226)	Sets current help filter.

11.2.4.2.1 IFilteredHelpViewer.GetFilter Method

Returns current help filter.

Pascal

```
function GetFilter: string;
```

C++

```
__fastcall UnicodeString GetFilter();
```

Description

GetFilter queries the Help Viewer for the currently applied Help Filter.

See Also

SetFilter (see page 3226)

11.2.4.2.2 IFilteredHelpViewer.SetFilter Method

Sets current help filter.

Pascal

```
procedure SetFilter(const Filter: string);
```

C++

```
__fastcall SetFilter(const UnicodeString Filter);
```

Description

SetFilter sets the given help filter. This is useful when trying to narrow down help searches.

See Also

GetFilter (see page 3225)

11.2.5 IHelpManager Interface

IHelpManager provides access to the Help System for a Help Viewer.

Class Hierarchy

HelpIntfs.IHelpManager

Pascal

```
IHelpManager = interface;
```

C++

```
__interface IHelpManager;
```

File

HelpIntfs

Description

IHelpManager defines methods that a Help Viewer uses to interact with the Help System. A Viewer accesses these methods using the object that the global RegisterViewer (see page 3242) function returns.

See Also

RegisterViewer (see page 3242)

Help System Interfaces

11.2.5.1 IHelpManager Members

The following tables list the members exposed by IHelpManager.

Public Methods

	Name	Description
	GetHandle (see page 3227)	Get application window handle.
	GetHelpFile (see page 3227)	Get Help File name.
	Release (see page 3227)	Notify Help System of Help Viewer shutdown.

11.2.5.2 IHelpManager Methods

The methods of the IHelpManager class are listed here.

Public Methods

	Name	Description
	GetHandle (see page 3227)	Get application window handle.
	GetHelpFile (see page 3227)	Get Help File name.
	Release (see page 3227)	Notify Help System of Help Viewer shutdown.

11.2.5.2.1 IHelpManager.GetHandle Method

Get application window handle.

Pascal

```
function GetHandle: LongInt;
```

C++

```
__fastcall LongInt GetHandle();
```

Description

GetHandle returns the main window handle of the application responsible for the current Help request.

11.2.5.2.2 IHelpManager.GetHelpFile Method

Get Help File name.

Pascal

```
function GetHelpFile: string;
```

C++

```
__fastcall UnicodeString GetHelpFile();
```

Description

GetHelpFile returns the name of the Help File associated with the current Help request.

11.2.5.2.3 IHelpManager.Release Method

Notify Help System of Help Viewer shutdown.

Pascal

```
procedure Release(const ViewerID: Integer);
```

C++

```
__fastcall Release(const int ViewerID);
```

Description

Release disconnects the Help Viewer from the RTL Help System. The ViewerID parameter is the value the Help System previously passed to the Viewer by calling NotifyID.

A Help Viewer should call Release when it is about to shut down, unless the Viewer is shutting down in response to a call to ShutDown.

See Also

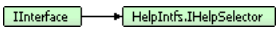
NotifyID (see page 3218)

ShutDown (see page 3218)

11.2.6 IHelpSelector Interface

Interface for Help Selector object.

Class Hierarchy



Pascal

```
IHelpSelector = interface(IInterface);
```

C++

```
__interface IHelpSelector : public IInterface;
```

File

HelpIntfs

Description

When an application uses more than one Help Viewer, there may be multiple responses to a Help request. By default, the Help System polls the Viewers in the order they were registered, and gives the request to the first viewer that claims to be able to handle it.

An application can optionally provide a Help Selector, which determines the Viewer for keyword-based Help requests and for requests to display a Table of Contents. To define a Help Selector, create an object that implements IHelpSelector and pass it to the Help System, using AssignHelpSelector.

The Help Selector cannot affect context-based or topic-based Help requests, which always go to the first Viewer that claims to be able to handle the request.

See Also

RegisterViewer (see page 3242)

IHelpSystem (see page 3231)

Help System Interfaces

11.2.6.1 IHelpSelector Members

The following tables list the members exposed by IHelpSelector.

Public Methods

	Name	Description
	SelectKeyword (see page 3229)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 3229)	Choose Viewer to display Table of Contents.

11.2.6.2 IHelpSelector Methods

The methods of the IHelpSelector class are listed here.

Public Methods

	Name	Description
	SelectKeyword (see page 3229)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 3229)	Choose Viewer to display Table of Contents.

11.2.6.2.1 IHelpSelector.SelectKeyword Method

Keyword-based Help support for multiple Viewers.

Pascal

```
function SelectKeyword(Keywords: TStrings): Integer;
```

C++

```
__fastcall int SelectKeyword(TStrings Keywords);
```

Description

The Help System calls SelectKeyword when more than one Help Viewer has topics that match a keyword-based Help request. The Keywords parameter contains a merged list of Help Strings, obtained by calling GetHelpStrings for each Viewer. SelectKeyword indicates a specific topic by returning the index of a specific Help String. The Help System then passes this Help String to the ShowHelp method of the appropriate Viewer. If the return value is negative, the Help System assumes that the Help request has been aborted and does nothing.

Typically, SelectKeyword displays a dialog box so the user can choose one of the Help Strings, and returns the index of the user's choice, or a negative value if the user cancels the dialog. This duplicates a feature of most Viewers, but allows the user to see all matching topics in a single list.

Note: SelectKeyword must not modify or rearrange the Help Strings list.

See Also

UnderstandsKeyword (see page 3219)

GetHelpStrings (see page 3217)

ShowHelp (see page 3218)

11.2.6.2.2 IHelpSelector.TableOfContents Method

Choose Viewer to display Table of Contents.

Pascal

```
function TableOfContents(Contents: TStrings): Integer;
```

C++

```
__fastcall int TableOfContents(TStrings Contents);
```

Description

The Help System calls TableOfContents when more than one Help Viewer responds to a request to show a Help Table of Contents. The Help System passes a list of Viewer names in the Contents parameter. TableOfContents should return the index of the chosen Viewer's name.

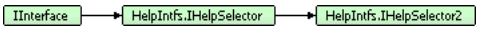
Note: TableOfContents must not modify or rearrange the Viewer names list.

See Also

CanShowTableOfContents (see page 3216)

11.2.7 IHelpSelector2 Interface

Class Hierarchy



Pascal

```
IHelpSelector2 = interface(IHelpSelector);
```

C++

```
__interface IHelpSelector2 : public IHelpSelector;
```

File

HelpIntfs

Description

This is class HelpIntfs.IHelpSelector2.

11.2.7.1 IHelpSelector2 Members

The following tables list the members exposed by IHelpSelector2.

Public Methods

	Name	Description
	SelectContext (see page 3230)	This is SelectContext, a member of class IHelpSelector2.
	SelectKeyword (see page 3229)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 3229)	Choose Viewer to display Table of Contents.

11.2.7.2 IHelpSelector2 Methods

The methods of the IHelpSelector2 class are listed here.

Public Methods

	Name	Description
	SelectContext (see page 3230)	This is SelectContext, a member of class IHelpSelector2.

11.2.7.2.1 IHelpSelector2.SelectContext Method

Pascal

```
function SelectContext(Viewers: TStrings): Integer;
```

C++

```
__fastcall int SelectContext(TStrings Viewers);
```

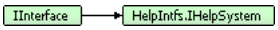
Description

This is SelectContext, a member of class IHelpSelector2.

11.2.8 IHelpSystem Interface

Application Help System that supports multiple Help Viewers.

Class Hierarchy



Pascal

```
IHelpSystem = interface(IInterface);
```

C++

```
__interface IHelpSystem : public IInterface;
```

File

HelpIntfs

Description

The Run-Time Library Help System allows an application to access one or more Help Viewers through a uniform interface, IHelpSystem. To obtain an object that implements IHelpSystem, use the HelpSystem property of the Application global variable, or call the global GetHelpSystem (see page 3242) function.

See Also

Help System Interfaces

11.2.8.1 IHelpSystem Members

The following tables list the members exposed by IHelpSystem.

Public Methods

	Name	Description
	AssignHelpSelector (see page 3232)	Assign a Help Selector object.
	Hook (see page 3232)	Do a WinHelp-style Help request.
	ShowContextHelp (see page 3232)	Do Context-based Help request.
	ShowHelp (see page 3233)	Do Keyword-based Help request.
	ShowTableOfContents (see page 3233)	Show Table of Contents.
	ShowTopicHelp (see page 3233)	Show Help topic.

11.2.8.2 IHelpSystem Methods

The methods of the IHelpSystem class are listed here.

Public Methods

	Name	Description
	AssignHelpSelector (see page 3232)	Assign a Help Selector object.
	Hook (see page 3232)	Do a WinHelp-style Help request.

	ShowContextHelp (see page 3232)	Do Context-based Help request.
	ShowHelp (see page 3233)	Do Keyword-based Help request.
	ShowTableOfContents (see page 3233)	Show Table of Contents.
	ShowTopicHelp (see page 3233)	Show Help topic.

11.2.8.2.1 IHelpSystem.AssignHelpSelector Method

Assign a Help Selector object.

Pascal

```
procedure AssignHelpSelector(const Selector: IHelpSelector);
```

C++

```
__fastcall AssignHelpSelector(const IHelpSelector Selector);
```

Description

Call AssignHelpSelector to provide a Help Selector object. Use of a Help Selector allows an application to define the way a Help Viewer is chosen to handle keyword-based Help requests and requests to display a Table of Contents.

11.2.8.2.2 IHelpSystem.Hook Method

Do a WinHelp-style Help request.

Pascal

```
function Hook(Handle: Longint; const HelpFile: string; Comand: Word; Data: Longint): Boolean;
```

C++

```
__fastcall Boolean Hook(Longint Handle, const UnicodeString HelpFile, Word Comand, Longint
Data);
```

Description

Hook accepts Help requests that uses the same conventions as the Microsoft Windows WinHelp API. Hook should only be called for requests that cannot be translated into calls to ShowContextHelp (see page 3232), ShowHelp (see page 3233), ShowTableOfContents (see page 3233), or ShowTopicHelp (see page 3233).

Note: If the Help System is unable to decipher the Help request, it looks for a Viewer that implements ISpecialWinHelpViewer (see page 3239). If the application has registered such a viewer, the request succeeds if the Viewer is able to handle it. Otherwise the request fails.

11.2.8.2.3 IHelpSystem.ShowContextHelp Method

Do Context-based Help request.

Pascal

```
procedure ShowContextHelp(const ContextID: Longint; const HelpFileName: string);
```

C++

```
__fastcall ShowContextHelp(const Longint ContextID, const UnicodeString HelpFileName);
```

Description

Call ShowContextHelp to display a single Help topic, identified by ContextID and HelpFileName. For context-based Help requests to work, the application must have previously registered at least one Help Viewer that implements IExtendedHelpViewer (see page 3219). If no such Viewer is found, an EHelpSystemException (see page 3214) is raised.

11.2.8.2.4 IHelpSystem.ShowHelp Method

Do Keyword-based Help request.

Pascal

```
procedure ShowHelp(const HelpKeyword: string; const HelpFileName: string);
```

C++

```
__fastcall ShowHelp(const UnicodeString HelpKeyword, const UnicodeString HelpFileName);
```

Description

Call ShowHelp to submit a keyword-based Help request. By default, the RTL Help System gives the request to the first Help Viewer that claims to be able to handle it. Typically, the Viewer passes the Help request to an external Help engine, which displays a list of matching topics (if more than one topic matches) or goes directly to a specific topic (if a single topic matches).

To support keyword-based Help requests using multiple Help Viewers, assign a Help Selector, using the AssignHelpSelector (see page 3232) method.

11.2.8.2.5 IHelpSystem.ShowTableOfContents Method

Show Table of Contents.

Pascal

```
procedure ShowTableOfContents;
```

C++

```
__fastcall ShowTableOfContents();
```

Description

Call ShowTableOfContents to request that the Help System display a Table of Contents.

Note: Tapplication.HelpShowTableOfContents is the preferred method of opening the Table of Contents.

11.2.8.2.6 IHelpSystem.ShowTopicHelp Method

Show Help topic.

Pascal

```
procedure ShowTopicHelp(const Topic: string; const HelpFileName: string);
```

C++

```
__fastcall ShowTopicHelp(const UnicodeString Topic, const UnicodeString HelpFileName);
```

Description

Call ShowTopicHelp to request that the Help System display a specific Help topic.

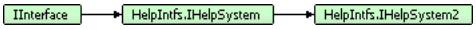
The Topic parameter is a short string identifying the specific topic.

The HelpFileName parameter is the name of the file containing the topic. If this parameter is a zero-length string, the HelpFile property of the global Application variable is used.

11.2.9 IHelpSystem2 Interface

Application Help System that supports multiple Help Viewers.

Class Hierarchy



Pascal

```
IHelpSystem2 = interface(IHelpSystem);
```

C++

```
__interface IHelpSystem2 : public IHelpSystem;
```

File

HelpIntfs

Description

The Run-Time Library Help System allows an application to access one or more Help Viewers through a uniform interface, IHelpSystem2. To obtain an object that implements IHelpSystem2, call the global function GetHelpSystem (see page 3242).

This interface extends IHelpSystem (see page 3231) with the ability to query the Help System if it understands a given keyword.

See Also

Help System Interfaces

GetHelpSystem (see page 3242)

11.2.9.1 IHelpSystem2 Members

The following tables list the members exposed by IHelpSystem2.

Public Methods

	Name	Description
	AssignHelpSelector (see page 3232)	Assign a Help Selector object.
	Hook (see page 3232)	Do a WinHelp-style Help request.
	ShowContextHelp (see page 3232)	Do Context-based Help request.
	ShowHelp (see page 3233)	Do Keyword-based Help request.
	ShowTableOfContents (see page 3233)	Show Table of Contents.
	ShowTopicHelp (see page 3233)	Show Help topic.
	UnderstandsKeyword (see page 3235)	Queries Help system for ability to understand keyword.

11.2.9.2 IHelpSystem2 Methods

The methods of the IHelpSystem2 class are listed here.

Public Methods

	Name	Description
	UnderstandsKeyword (see page 3235)	Queries Help system for ability to understand keyword.

11.2.9.2.1 IHelpSystem2.UnderstandsKeyword Method

Queries Help system for ability to understand keyword.

Pascal

```
function UnderstandsKeyword(const HelpKeyword: string; const HelpFileName: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsKeyword(const UnicodeString HelpKeyword, const UnicodeString
HelpFileName);
```

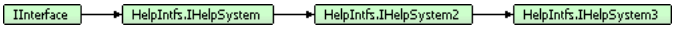
Description

The Help System calls UnderstandsKeyword to determine if the Help Viewer can provide topics for a keyword-based Help request based on the given keyword. UnderstandsKeyword should return true if any registered help viewer understands the given keyword.

11.2.10 IHelpSystem3 Interface

Application Help System that supports multiple Help Viewers.

Class Hierarchy



Pascal

```
IHelpSystem3 = interface(IHelpSystem2);
```

C++

```
__interface IHelpSystem3 : public IHelpSystem2;
```

File

HelpIntfs

Description

The Run-Time Library Help System allows an application to access one or more Help Viewers through a uniform interface, IHelpSystem3. To obtain an object that implements IHelpSystem3, call the global function GetHelpSystem (see page 3242).

This interface extends IHelpSystem2 (see page 3234) with the ability to show the index and search panes. It also allows you to utilize page anchors and get and set help filters.

See Also

Help System Interfaces

GetHelpSystem (see page 3242)

11.2.10.1 IHelpSystem3 Members

The following tables list the members exposed by IHelpSystem3.

Public Methods

	Name	Description
	AssignHelpSelector (see page 3232)	Assign a Help Selector object.
	GetFilter (see page 3236)	Returns current help filter.
	Hook (see page 3232)	Do a WinHelp-style Help request.
	SetFilter (see page 3236)	Sets current help filter.
	ShowContextHelp (see page 3232)	Do Context-based Help request.
	ShowHelp (see page 3233)	Do Keyword-based Help request.
	ShowIndex (see page 3237)	Opens Index pane of Help Viewer.
	ShowSearch (see page 3237)	Opens Search pane of Help Viewer.
	ShowTableOfContents (see page 3233)	Show Table of Contents.
	ShowTopicHelp (see page 3237)	Show Help topic.
	UnderstandsKeyword (see page 3235)	Queries Help system for ability to understand keyword.

11.2.10.2 IHelpSystem3 Methods

The methods of the IHelpSystem3 class are listed here.

Public Methods

	Name	Description
	GetFilter (see page 3236)	Returns current help filter.
	SetFilter (see page 3236)	Sets current help filter.
	ShowIndex (see page 3237)	Opens Index pane of Help Viewer.
	ShowSearch (see page 3237)	Opens Search pane of Help Viewer.
	ShowTopicHelp (see page 3237)	Show Help topic.

11.2.10.2.1 IHelpSystem3.GetFilter Method

Returns current help filter.

Pascal

```
function GetFilter: string;
```

C++

```
__fastcall UnicodeString GetFilter();
```

Description

The RTL Help System calls GetFilter to have the Viewer return its current help filter. This function returns the filter of the first Help Viewer that implements this interface (Help Viewers are found in the order that they are added).

See Also

SetFilter (see page 3236)

11.2.10.2.2 IHelpSystem3.SetFilter Method

Sets current help filter.

Pascal

```
procedure SetFilter(const Filter: string);
```

C++

```
__fastcall SetFilter(const UnicodeString Filter);
```

Description

The RTL Help System calls SetFilter to set the given filter for all Viewers that implement IHelpSystem3 (see page 3235).

See Also

GetFilter (see page 3236)

11.2.10.2.3 IHelpSystem3.ShowIndex Method

Opens Index pane of Help Viewer.

Pascal

```
procedure ShowIndex(const Topic: string; const HelpFileName: string);
```

C++

```
__fastcall ShowIndex(const UnicodeString Topic, const UnicodeString HelpFileName);
```

Description

The RTL Help System calls ShowIndex to have a Viewer open its Index Pane. If the Help Viewer has the ability to search for the given topic parameter, it executes the Index search; otherwise it is ignored. The **HelpFileName** parameter is only used in certain help systems (such as Microsoft HTML Help) and is used only if there is no help file name already known by the Help System.

The help system queries all registered Help Viewers. If only one viewer can display an Index pane, it is shown; otherwise the user is asked to decide which Viewer to display.

See Also

ShowSearch (see page 3237)

11.2.10.2.4 IHelpSystem3.ShowSearch Method

Opens Search pane of Help Viewer.

Pascal

```
procedure ShowSearch(const Topic: string; const HelpFileName: string);
```

C++

```
__fastcall ShowSearch(const UnicodeString Topic, const UnicodeString HelpFileName);
```

Description

The RTL Help System calls ShowSearch to have a Viewer open its Search Pane. If the Help Viewer has the ability to search for the given topic parameter, it executes the search; otherwise it is ignored. The **HelpFileName** parameter is only used in certain help systems (such as Microsoft HTML Help) and is only used if there is no help file name already known by the Help System.

The help system queries all registered Help Viewers. If only one viewer can display an Search pane then it is shown; otherwise the user is asked to decide which Viewer to display.

See Also

ShowIndex (see page 3237)

11.2.10.2.5 IHelpSystem3.ShowTopicHelp Method

Show Help topic.

Pascal

```
procedure ShowTopicHelp(const Topic: string; const Anchor: string; const HelpFileName:
string); overload;
```

C++

```
__fastcall ShowTopicHelp(const UnicodeString Topic, const UnicodeString Anchor, const
UnicodeString HelpFileName);
```

Description

Call ShowTopicHelp to request that the Help System display a specific Help topic.

The **Topic** parameter is a short string identifying the specific topic.

The **Anchor** parameter is a specific place in a page specified while creating the help page. The anchor is ignored if it does not exist in the page.

The **HelpFileName** parameter is the name of the file containing the topic. If this parameter is a zero-length string, the HelpFile property of the global variable Application is used.

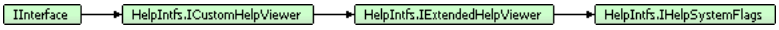
See Also

HelpFile (see page 1974)

Application (see page 2612)

11.2.11 IHelpSystemFlags Interface

Class Hierarchy



Pascal

```
IHelpSystemFlags = interface(IExtendedHelpViewer);
```

C++

```
__interface IHelpSystemFlags : public IExtendedHelpViewer;
```

File

HelpIntfs

Description

This is class HelpIntfs.IHelpSystemFlags.

11.2.11.1 IHelpSystemFlags Members

The following tables list the members exposed by IHelpSystemFlags.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayTopic (see page 3221)	Use Topic ID to display topic.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.

	GetUseDefaultTopic (see page 3239)	This is GetUseDefaultTopic, a member of class IHelpSystemFlags.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	SetUseDefaultTopic (see page 3239)	This is SetUseDefaultTopic, a member of class IHelpSystemFlags.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.11.2 IHelpSystemFlags Methods

The methods of the IHelpSystemFlags class are listed here.

Public Methods

	Name	Description
	GetUseDefaultTopic (see page 3239)	This is GetUseDefaultTopic, a member of class IHelpSystemFlags.
	SetUseDefaultTopic (see page 3239)	This is SetUseDefaultTopic, a member of class IHelpSystemFlags.

11.2.11.2.1 IHelpSystemFlags.GetUseDefaultTopic Method

Pascal

```
function GetUseDefaultTopic: Boolean;
```

C++

```
__fastcall Boolean GetUseDefaultTopic();
```

Description

This is GetUseDefaultTopic, a member of class IHelpSystemFlags.

11.2.11.2.2 IHelpSystemFlags.SetUseDefaultTopic Method

Pascal

```
procedure SetUseDefaultTopic(AValue: Boolean);
```

C++

```
__fastcall SetUseDefaultTopic(Boolean AValue);
```

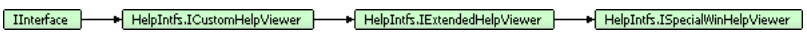
Description

This is SetUseDefaultTopic, a member of class IHelpSystemFlags.

11.2.12 ISpecialWinHelpViewer Interface

Help Viewer that supports special Windows Help requests.

Class Hierarchy



Pascal

```
ISpecialWinHelpViewer = interface(IExtendedHelpViewer);
```

C++

```
__interface ISpecialWinHelpViewer : public IExtendedHelpViewer;
```

File

HelpIntfs

Description

ISpecialWinHelpViewer supports Help requests that cannot be easily generalized, and should only be handled by the Microsoft Windows Help engine or a closely compatible application.

ISpecialWinHelpViewer consists of a single method, CallWinHelp (see page 3241), which should pass the Help request to the Help engine with as little manipulation as possible. An application needs to provide only one Viewer that implements ISpecialWinHelpViewer. If more than one Viewer is registered that implements ISpecialWinHelpViewer, only the first Viewer ever receives calls to CallWinHelp (see page 3241).

See Also

IhelpManager

IhelpSystem

ISpecialWinHelpViewer

RegisterViewer (see page 3242)

Help System Interfaces

11.2.12.1 ISpecialWinHelpViewer Members

The following tables list the members exposed by ISpecialWinHelpViewer.

Public Methods

	Name	Description
	CallWinHelp (see page 3241)	Pass Help request directly to engine.
	CanShowTableOfContents (see page 3216)	Query support for Table of Contents.
	DisplayHelpByContext (see page 3221)	Use Context ID to display topic.
	DisplayTopic (see page 3221)	Use Topic ID to display topic.
	GetHelpStrings (see page 3217)	Return Help Strings that match a keyword.
	GetViewerName (see page 3217)	Return string identifying Help Viewer.
	NotifyID (see page 3218)	Save Viewer's numeric ID.
	ShowHelp (see page 3218)	Show a specific Help topic.
	ShowTableOfContents (see page 3218)	Show a Table of Contents.
	ShutDown (see page 3218)	Permanent Viewer shut down.
	SoftShutDown (see page 3219)	Temporary Viewer shut down.
	UnderstandsContext (see page 3221)	Queries the Help Viewer if it understands the given Context ID.
	UnderstandsKeyword (see page 3219)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 3222)	Queries the Help Viewer if it understands the given Topic.

11.2.12.2 ISpecialWinHelpViewer Methods

The methods of the ISpecialWinHelpViewer class are listed here.

Public Methods

	Name	Description
	CallWinHelp (see page 3241)	Pass Help request directly to engine.

11.2.12.2.1 ISpecialWinHelpViewer.CallWinHelp Method

Pass Help request directly to engine.

Pascal

```
function CallWinHelp(Handle: LongInt; const HelpFile: string; Command: Word; Data: LongInt):
Boolean;
```

C++

```
__fastcall Boolean CallWinHelp(LongInt Handle, const UnicodeString HelpFile, Word Command,
LongInt Data);
```

Description

The RTL Help System calls CallWinHelp when it receives a Windows-style Help request that cannot be generalized as a Topic-based or Context-based request. CallWinHelp should pass the request directly to the Windows winhelp() API or to a functional equivalent.

CallWinHelp should return true if it was able to handle the request.

11.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetHelpSystem (see page 3242)	Returns Help System object.
	GetHelpSystem (see page 3242)	Returns Help System object.
	GetHelpSystem (see page 3242)	Returns Help System object.
	RegisterViewer (see page 3242)	Register Help Viewer to receive requests.

11.3.1 HelpIntfs.GetHelpSystem Function

Returns Help System object.

Pascal

```
function GetHelpSystem(out System: IHelpSystem): Boolean; overload;
function GetHelpSystem(out System: IHelpSystem2): Boolean; overload;
function GetHelpSystem(out System: IHelpSystem3): Boolean; overload;
```

C++

```
Boolean GetHelpSystem(IHelpSystem System);
Boolean GetHelpSystem(IHelpSystem2 System);
Boolean GetHelpSystem(IHelpSystem3 System);
```

File

HelpIntfs

Description

GetHelpSystem returns an object that implements IHelpSystem (see page 3231). An application calls IHelpSystem (see page 3231) methods to interact with the Help System.

GetHelpSystem returns 1 if the Help System object was successfully obtained, 0 otherwise.

See Also

TApplication

11.3.2 HelpIntfs.RegisterViewer Function

Register Help Viewer to receive requests.

Pascal

```
function RegisterViewer(const newViewer: ICustomHelpViewer; out Manager: IHelpManager):
Integer;
```

C++

```
int RegisterViewer(const ICustomHelpViewer newViewer, IHelpManager Manager);
```

File

HelpIntfs

Description

Call RegisterViewer to register a Help Viewer with the Help System. An application can register any number of Viewers. The order in which Help Viewers are registered is significant, because each Help request causes the RTL Help System to poll the Viewers in the order they were registered, and give the Help request to the first viewer that claims to be able to handle it. An application can override this behavior for keyword-based Help requests and for requests to display a Table of Contents by providing a Help Selector object.

Pass the Viewer object in the newViewer parameter.

RegisterViewer returns a Help Manager object in the Manager parameter; the Viewer uses this object to interact with the Help System.

The return value of RegisterViewer is 1 is the Viewer was successfully registered, 0 otherwise.

12 HTMLHelpViewer Namespace

This is namespace HTMLHelpViewer.

Topics

Name	Description
Interfaces (see page 3246)	The following table lists interfaces in this documentation.
Variables (see page 3249)	The following table lists variables in this documentation.

Interfaces

	Name	Description
	IHtmlHelpTester (see page 3246)	This is class HTMLHelpViewer.IHtmlHelpTester.

Variables

Name	Description
HtmlHelpTester (see page 3249)	This is variable HTMLHelpViewer.HtmlHelpTester.
ViewerName (see page 3249)	This is variable HTMLHelpViewer.ViewerName.

12.1 Interfaces

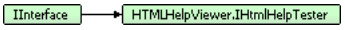
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHtmlHelpTester (see page 3246)	This is class HTMLHelpViewer.IHtmlHelpTester.

12.1.1 IHtmlHelpTester Interface

Class Hierarchy



Pascal

```
IHtmlHelpTester = interface(IInterface);
```

C++

```
__interface IHtmlHelpTester : public IInterface;
```

File

HTMLHelpViewer

Description

This is class HTMLHelpViewer.IHtmlHelpTester.

12.1.1.1 IHtmlHelpTester Members

The following tables list the members exposed by IHtmlHelpTester.

Public Methods

	Name	Description
	CanShowALink (see page 3247)	This is CanShowALink, a member of class IHtmlHelpTester.
	CanShowContext (see page 3247)	This is CanShowContext, a member of class IHtmlHelpTester.
	CanShowTopic (see page 3247)	This is CanShowTopic, a member of class IHtmlHelpTester.
	ConvertWinHelp (see page 3247)	This is ConvertWinHelp, a member of class IHtmlHelpTester.
	GetDefaultHelpFile (see page 3248)	This is GetDefaultHelpFile, a member of class IHtmlHelpTester.
	GetHelpPath (see page 3248)	This is GetHelpPath, a member of class IHtmlHelpTester.
	GetHelpStrings (see page 3248)	This is GetHelpStrings, a member of class IHtmlHelpTester.

12.1.1.2 IHtmlHelpTester Methods

The methods of the IHtmlHelpTester class are listed here.

Public Methods

	Name	Description
	CanShowALink (see page 3247)	This is CanShowALink, a member of class IHtmlHelpTester.
	CanShowContext (see page 3247)	This is CanShowContext, a member of class IHtmlHelpTester.
	CanShowTopic (see page 3247)	This is CanShowTopic, a member of class IHtmlHelpTester.
	ConvertWinHelp (see page 3247)	This is ConvertWinHelp, a member of class IHtmlHelpTester.
	GetDefaultHelpFile (see page 3248)	This is GetDefaultHelpFile, a member of class IHtmlHelpTester.
	GetHelpPath (see page 3248)	This is GetHelpPath, a member of class IHtmlHelpTester.
	GetHelpStrings (see page 3248)	This is GetHelpStrings, a member of class IHtmlHelpTester.

12.1.1.2.1 IHtmlHelpTester.CanShowALink Method

Pascal

```
function CanShowALink(const ALink: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean CanShowALink(const UnicodeString ALink, const UnicodeString FileName);
```

Description

This is CanShowALink, a member of class IHtmlHelpTester.

12.1.1.2.2 IHtmlHelpTester.CanShowContext Method

Pascal

```
function CanShowContext(const Context: Integer; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean CanShowContext(const int Context, const UnicodeString FileName);
```

Description

This is CanShowContext, a member of class IHtmlHelpTester.

12.1.1.2.3 IHtmlHelpTester.CanShowTopic Method

Pascal

```
function CanShowTopic(const Topic: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean CanShowTopic(const UnicodeString Topic, const UnicodeString FileName);
```

Description

This is CanShowTopic, a member of class IHtmlHelpTester.

12.1.1.2.4 IHtmlHelpTester.ConvertWinHelp Method

Pascal

```
function ConvertWinHelp(var Command: Word; var Data: LongInt): Boolean;
```

C++

```
__fastcall Boolean ConvertWinHelp(Word Command, LongInt Data);
```

Description

This is ConvertWinHelp, a member of class IHtmlHelpTester.

12.1.1.2.5 IHtmlHelpTester.GetDefaultHelpFile Method

Pascal

```
function GetDefaultHelpFile: string;
```

C++

```
__fastcall UnicodeString GetDefaultHelpFile();
```

Description

This is GetDefaultHelpFile, a member of class IHtmlHelpTester.

12.1.1.2.6 IHtmlHelpTester.GetHelpPath Method

Pascal

```
function GetHelpPath: string;
```

C++

```
__fastcall UnicodeString GetHelpPath();
```

Description

This is GetHelpPath, a member of class IHtmlHelpTester.

12.1.1.2.7 IHtmlHelpTester.GetHelpStrings Method

Pascal

```
function GetHelpStrings(const ALink: string): TStringList;
```

C++

```
__fastcall TStringList GetHelpStrings(const UnicodeString ALink);
```

Description

This is GetHelpStrings, a member of class IHtmlHelpTester.

12.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
HtmlHelpTester (see page 3249)	This is variable HTMLHelpViewer.HtmlHelpTester.
ViewerName (see page 3249)	This is variable HTMLHelpViewer.ViewerName.

12.2.1 HTMLHelpViewer.HtmlHelpTester Variable

Pascal

```
HtmlHelpTester: IHtmlHelpTester;
```

C++

```
IHtmlHelpTester HtmlHelpTester;
```

File

HTMLHelpViewer

Description

This is variable HTMLHelpViewer.HtmlHelpTester.

12.2.2 HTMLHelpViewer.ViewerName Variable

Pascal

```
ViewerName: string;
```

C++

```
UnicodeString ViewerName;
```

File

HTMLHelpViewer

Description

This is variable HTMLHelpViewer.ViewerName.

13 HTTPApp Namespace

This is namespace HTTPApp.

Topics

Name	Description
Classes (see page 3254)	The following table lists classes in this documentation.
Interfaces (see page 3416)	The following table lists interfaces in this documentation.
Functions (see page 3436)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3444)	The following table lists structs, records, enums in this documentation.
Types (see page 3446)	The following table lists types in this documentation.
Constants (see page 3448)	The following table lists constants in this documentation.

Classes

	Name	Description
	EWebBrokerException (see page 3254)	EWebBrokerException is an exception raised by WebBroker objects.
	TAbstractContentParser (see page 3255)	TAbstractContentParser is the base class for objects that parse the content of an HTTP request.
	TAbstractWebModuleFactory (see page 3259)	TAbstractWebModuleFactory is the base class for all Web module factory objects.
	TAbstractWebModuleList (see page 3266)	TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TAbstractWebPageInfo (see page 3275)	TAbstractWebPageInfo represents the attributes of a Web page module.
	TAbstractWebPageModuleFactory (see page 3285)	TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TAbstractWebRequestFile (see page 3288)	TAbstractWebRequestFile describes an uploaded file from an HTTP request.
	TAbstractWebRequestFiles (see page 3292)	TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 3288) objects.
	TAbstractWebSession (see page 3295)	TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.
	TContentParser (see page 3301)	TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.
	TCookie (see page 3304)	TCookie represents a Cookie header in an HTTP response message.
	TCookieCollection (see page 3309)	TCookieCollection is a collection of TCookie (see page 3304) objects.
	TCustomContentProducer (see page 3313)	TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.
	TCustomWebDispatcher (see page 3318)	TCustomWebDispatcher is the abstract base class for Web dispatcher objects.
	TDefaultWebAppServices (see page 3330)	This is class HTTPApp.TDefaultWebAppServices.
	TWebActionItem (see page 3335)	TWebActionItem allows a Web server application to create and send responses to HTTP request messages.

	TWebActionItems (see page 3344)	TWebActionItems is a collection of TWebActionItem (see page 3335) objects.
	TWebDispatcher (see page 3349)	TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.
	TWebModule (see page 3354)	TWebModule is the automatically generated Web module for a Web server application.
	TWebRequest (see page 3358)	TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.
	TWebResponse (see page 3389)	TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

Constants

Name	Description
MAX_DATETIMES (see page 3448)	This is constant HTTPApp.MAX_DATETIMES.
MAX_INTEGERS (see page 3448)	This is constant HTTPApp.MAX_INTEGERS.
MAX_STRINGS (see page 3449)	This is constant HTTPApp.MAX_STRINGS.
sDateFormat (see page 3449)	This is constant HTTPApp.sDateFormat.

Functions

	Name	Description
	DayOfWeekStr (see page 3436)	Converts a TDateTime into a string representing the day of the week.
	DosPathToUnixPath (see page 3437)	Converts a DOS-compatible path specification to a Unix-compatible path specification.
	EncodingFromContentType (see page 3437)	This is function HTTPApp.EncodingFromContentType.
	EncodingGetBytes (see page 3437)	This is function HTTPApp.EncodingGetBytes.
	EncodingGetString (see page 3438)	This is function HTTPApp.EncodingGetString.
	ExtractHeaderFields (see page 3438)	Parses a multi-valued string into its constituent fields.
	ExtractHeaderFields (see page 3438)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 3439)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 3439)	Parses a multi-valued string into its constituent fields.
	HTMLDecode (see page 3439)	This is function HTTPApp.HTMLDecode.
	HTMLEncode (see page 3440)	This is function HTTPApp.HTMLEncode.
	HTTPDecode (see page 3440)	Decodes a string that includes HTTP escape characters.
	HTTPEncode (see page 3441)	Converts a string into a form that contains no values that are illegal in an HTTP message header.
	MonthStr (see page 3441)	Converts a TDateTime into a string representing the month.
	ParseDate (see page 3442)	Converts a date value from an HTTP message header into a TDateTime.
	RegisterContentParser (see page 3442)	This is function HTTPApp.RegisterContentParser.
	StatusString (see page 3442)	Returns the explanatory string associated with one of the predefined HTTP response status codes.
	UnixPathToDosPath (see page 3443)	Converts a Unix-compatible path specification to a Dos-compatible path specification.

Interfaces

	Name	Description
	IGetAppDispatcher (see page 3416)	This is class HTTPApp.IGetAppDispatcher.
	IGetWebAppServices (see page 3417)	This is class HTTPApp.IGetWebAppServices.
	IGetWebRequestHandler (see page 3418)	This is class HTTPApp.IGetWebRequestHandler.
	IMultiModuleSupport (see page 3419)	This is class HTTPApp.IMultiModuleSupport.
	IProduceContent (see page 3420)	IProduceContent is the interface for generating the content of a Web page.
	IProduceContentFrom (see page 3422)	This is class HTTPApp.IProduceContentFrom.
	ISetAppDispatcher (see page 3423)	This is class HTTPApp.ISetAppDispatcher.
	IWebAppServices (see page 3424)	IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.
	IWebDispatch (see page 3427)	
	IWebDispatchActions (see page 3430)	IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.
	IWebDispatcherAccess (see page 3431)	IWebDispatcherAccess provides access to the current HTTP request and response object.

	IWebExceptionHandler (see page 3433)	This is class HTTPApp.IWebExceptionHandler.
	IWebRequestHandler (see page 3434)	IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

Structs, Records, Enums

	Name	Description
	TMethodType (see page 3444)	This is record HTTPApp.TMethodType.
	TWebModuleCacheMode (see page 3444)	This is record HTTPApp.TWebModuleCacheMode.
	TWebModuleCreateMode (see page 3445)	This is record HTTPApp.TWebModuleCreateMode.

Types

Name	Description
TContentParserClass (see page 3446)	This is type HTTPApp.TContentParserClass.
THTTPMethodEvent (see page 3446)	This is type HTTPApp.THTTPMethodEvent.
TModuleAddedProc (see page 3447)	This is type HTTPApp.TModuleAddedProc.
TWebExceptionEvent (see page 3447)	This is type HTTPApp.TWebExceptionEvent.
TWebModuleContext (see page 3447)	This is type HTTPApp.TWebModuleContext.

13.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWebBrokerException (see page 3254)	EWebBrokerException is an exception raised by WebBroker objects.
	TAbstractContentParser (see page 3255)	TAbstractContentParser is the base class for objects that parse the content of an HTTP request.
	TAbstractWebModuleFactory (see page 3259)	TAbstractWebModuleFactory is the base class for all Web module factory objects.
	TAbstractWebModuleList (see page 3266)	TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TAbstractWebPageInfo (see page 3275)	TAbstractWebPageInfo represents the attributes of a Web page module.
	TAbstractWebPageModuleFactory (see page 3285)	TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TAbstractWebRequestFile (see page 3288)	TAbstractWebRequestFile describes an uploaded file from an HTTP request.
	TAbstractWebRequestFiles (see page 3292)	TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 3288) objects.
	TAbstractWebSession (see page 3295)	TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.
	TContentParser (see page 3301)	TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.
	TCookie (see page 3304)	TCookie represents a Cookie header in an HTTP response message.
	TCookieCollection (see page 3309)	TCookieCollection is a collection of TCookie (see page 3304) objects.
	TCustomContentProducer (see page 3313)	TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.
	TCustomWebDispatcher (see page 3318)	TCustomWebDispatcher is the abstract base class for Web dispatcher objects.
	TDefaultWebAppServices (see page 3330)	This is class HTTPApp.TDefaultWebAppServices.
	TWebActionItem (see page 3335)	TWebActionItem allows a Web server application to create and send responses to HTTP request messages.
	TWebActionItems (see page 3344)	TWebActionItems is a collection of TWebActionItem (see page 3335) objects.
	TWebDispatcher (see page 3349)	TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.
	TWebModule (see page 3354)	TWebModule is the automatically generated Web module for a Web server application.
	TWebRequest (see page 3358)	TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.
	TWebResponse (see page 3389)	TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

13.1.1 EWebBrokerException Class

EWebBrokerException is an exception raised by WebBroker objects.

Class Hierarchy

Pascal

```
EWebBrokerException = class(Exception);
```

C++

```
class EWebBrokerException : public Exception;
```

File

HTTPApp

Description

EWebBrokerException is raised when a WebBroker object encounters a problem. It functions identically to its parent class, TException.

See Also

Exception

13.1.2 TAbstractContentParser Class

TAbstractContentParser is the base class for objects that parse the content of an HTTP request.

Class Hierarchy

Pascal

```
TAbstractContentParser = class(TObject);
```

C++

```
class TAbstractContentParser : public TObject;
```

File

HTTPApp

Description

Web request objects use a descendant of TAbstractContentParser to retrieve information about the named values in the content of the HTTP request they represent. TAbstractContentParser defines two protected methods, GetContentFields (see page 3257) and GetFiles (see page 3258), which the Web request uses to obtain values for its ContentFields and Files properties, respectively.

TAbstractContentParser should not be instantiated. The GetContentFields (see page 3257) and GetFiles (see page 3258) methods are abstract or, in C++ terminology, pure virtual, which means that TAbstractContentParser does not supply an implementation. Each TAbstractContentParser descendant overrides these methods to support its own method of extracting named value strings and uploaded files from an HTTP request. The TAbstractContentParser descendant also overrides the public CanParse (see page 3257) method to check whether the request message contains information encoded in a way that the content parser understands.

Each TAbstractContentParser descendant registers itself with the Web server application. Then, when a Web request object needs to parse its content to set the ContentFields and Files properties, it searches through the registered content parsers to locate the most recently registered parser that can parse the current content.

See Also

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TContentParser (see page 3301)

TMultipartContentParser (see page 6706)

13.1.2.1 TAbstractContentParser Members

The following tables list the members exposed by TAbstractContentParser.

Protected Methods

	Name	Description
	GetContentFields (see page 3257)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 3258)	Returns a set of uploaded files that the parser extracts from the current Web request.

Protected Properties

	Name	Description
	WebRequest (see page 3258)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
	CanParse (see page 3257)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	Create (see page 3256)	Creates and initializes a new TAbstractContentParser (see page 3255) object.

13.1.2.2 TAbstractContentParser.Create Constructor

Creates and initializes a new TAbstractContentParser (see page 3255) object.

Pascal

```
constructor Create(AWebRequest: TWebRequest); virtual;
```

C++

```
virtual __fastcall TAbstractContentParser(TWebRequest AWebRequest);
```

Description

Do not create instances of TAbstractContentParser (see page 3255). This class and its descendants are intended only as helper objects to the Web request. The Web request creates its own instance of a TAbstractContentParser (see page 3255) descendant that can parse its content.

AWebRequest is the Web request object that creates this content parser to extract information from its content. This becomes the value of the WebRequest (see page 3258) property.

See Also

WebRequest (see page 3258)

13.1.2.3 TAbstractContentParser Methods

The methods of the TAbstractContentParser class are listed here.

Protected Methods

	Name	Description
	GetContentFields (see page 3257)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 3258)	Returns a set of uploaded files that the parser extracts from the current Web request.

Public Methods

	Name	Description
	CanParse (see page 3257)	Indicates whether a specified request message contains information encoded in a way that this parser understands.

13.1.2.3.1 TAbstractContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a class method (Delphi) or static method (C++), which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered TAbstractContentParser (see page 3255) descendant until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TAbstractContentParser (see page 3255), CanParse always returns false. Descendant classes that implement the GetContentFields (see page 3257) and GetFiles (see page 3258) methods override CanParse to indicate when a Web request has a format they can parse.

See Also

GetContentFields (see page 3257)

GetFiles (see page 3258)

13.1.2.3.2 TAbstractContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; virtual; abstract;
```

C++

```
virtual __fastcall TStrings GetContentFields() = 0;
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TAbstractContentParser (see page 3255), GetContentFields is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract name/value pairs from the current request and add them to the returned string list.

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 3375)

WebRequest (see page 3258)

ExtractContentFields (see page 3363)

13.1.2.3.3 TAbstractContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles() = 0;
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TAbstractContentParser (see page 3255), GetFiles is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract uploaded files from the current request and add them to the Web request files object.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

Files (see page 3379)

WebRequest (see page 3258)

TAbstractWebRequestFile (see page 3288)

13.1.2.4 TAbstractContentParser Properties

The properties of the TAbstractContentParser class are listed here.

Protected Properties

	Name	Description
	WebRequest (see page 3258)	Indicates the Web request object that uses this parser to parse its content.

13.1.2.4.1 TAbstractContentParser.WebRequest Property

Indicates the Web request object that uses this parser to parse its content.

Pascal

```
property WebRequest: TWebRequest;
```

C++

```
__property TWebRequest WebRequest;
```

Description

Content parsers use the protected WebRequest property internally to access the content they parse. The value of WebRequest is a Web request object whose properties represent the headers and content of the current HTTP request message.

See Also

Content (see page 3373)

13.1.3 TAbstractWebModuleFactory Class

TAbstractWebModuleFactory is the base class for all Web module factory objects.

Class Hierarchy

Pascal

```
TAbstractWebModuleFactory = class;
```

C++

```
class TAbstractWebModuleFactory;
```

File

HTTPApp

Description

All WebSnap modules are registered with the WebSnap application using a factory object. In Delphi, the factory object is typically created in the initialization section of the module source file. The factory object is responsible for creating Web module instance. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instance for reuse with subsequent request messages.

Do not create instances of TAbstractWebModuleFactory. Most of its methods are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. Instead, create instances of a TAbstractWebModuleFactory descendant such as TWebDataModuleFactory, TWebAppDataModuleFactory, TWebPageModuleFactory, or TWebAppPageModuleFactory. Each of these classes includes an implementation of the abstract or pure virtual methods, and represents group of Web module factories. These groups are based on whether the Web module factory generates the single application Web module (which contains the components to dispatch HTTP request messages), and it generates a Web module that creates a Web page.

See Also

TWebAppPageModuleFactory

TWebAppDataModuleFactory

TWebPageModuleFactory

TWebRequestHandler

TWebDataModuleFactory

13.1.3.1 TAbstractWebModuleFactory Members

The following tables list the members exposed by TAbstractWebModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 3260)	Returns the value of the CacheMode (see page 3264) property.
	GetComponentClass (see page 3261)	Returns the value of the ComponentClass (see page 3264) property.
	GetCreateMode (see page 3261)	Returns the value of the CreateMode (see page 3265) property.
	GetIsAppModule (see page 3261)	Returns the value of the IsAppModule (see page 3265) property.
	GetModuleName (see page 3262)	Returns the value of the ModuleName (see page 3265) property.
	GetWebPageInfo (see page 3263)	Returns the value of the WebPageInfo (see page 3266) property.

Public Methods

	Name	Description
	GetModule (see page 3262)	Creates an instance of the associated Web module.
	PreventDestruction (see page 3263)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
	CacheMode (see page 3264)	Indicates whether the factory can cache Web modules for reuse.
	ComponentClass (see page 3264)	Specifies the class type of the Web module that this factory creates.
	CreateMode (see page 3265)	Indicates when the factory creates instances of its Web module.
	IsAppModule (see page 3265)	Indicates whether the corresponding Web module is the main Web application module.
	ModuleName (see page 3265)	Specifies the name of the Web module class that this factory instantiates.
	WebPageInfo (see page 3266)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

13.1.3.2 TAbstractWebModuleFactory Methods

The methods of the TAbstractWebModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 3260)	Returns the value of the CacheMode (see page 3264) property.
	GetComponentClass (see page 3261)	Returns the value of the ComponentClass (see page 3264) property.
	GetCreateMode (see page 3261)	Returns the value of the CreateMode (see page 3265) property.
	GetIsAppModule (see page 3261)	Returns the value of the IsAppModule (see page 3265) property.
	GetModuleName (see page 3262)	Returns the value of the ModuleName (see page 3265) property.
	GetWebPageInfo (see page 3263)	Returns the value of the WebPageInfo (see page 3266) property.

Public Methods

	Name	Description
	GetModule (see page 3262)	Creates an instance of the associated Web module.
	PreventDestruction (see page 3263)	Prevents the WebSnap application from freeing the Web module instance.

13.1.3.2.1 TAbstractWebModuleFactory.GetCacheMode Method

Returns the value of the CacheMode (see page 3264) property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; virtual; abstract;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode() = 0;
```

Description

GetCacheMode is the protected read implementation of the CacheMode (see page 3264) property.

In TAbstractWebModuleFactory (see page 3259), GetCacheMode is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the CacheMode (see page 3264) property.

See Also

CacheMode (see page 3264)

13.1.3.2.2 TAbstractWebModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass (see page 3264) property.

Pascal

```
function GetComponentClass: TComponentClass; virtual; abstract;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass() = 0;
```

Description

GetComponentClass is the protected read implementation of the ComponentClass (see page 3264) property.

In TAbstractWebModuleFactory (see page 3259), GetComponentClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ComponentClass (see page 3264) property.

See Also

ComponentClass (see page 3264)

13.1.3.2.3 TAbstractWebModuleFactory.GetCreateMode Method

Returns the value of the CreateMode (see page 3265) property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; virtual; abstract;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode() = 0;
```

Description

GetCreateMode is the protected read implementation of the CreateMode (see page 3265) property.

In TAbstractWebModuleFactory (see page 3259), GetCreateMode is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the CreateMode (see page 3265) property.

See Also

CreateMode (see page 3265)

13.1.3.2.4 TAbstractWebModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule (see page 3265) property.

Pascal

```
function GetIsAppModule: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetIsAppModule() = 0;
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule (see page 3265) property.

In TAbstractWebModuleFactory (see page 3259), GetIsAppModule is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the IsAppModule (see page 3265) property.

See Also

IsAppModule (see page 3265)

13.1.3.2.5 TAbstractWebModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * GetModule() = 0;
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. This method should only be called once for each request message that requires the use of the Web module.

GetModule returns the Web module instance.

In TAbstractWebModuleFactory (see page 3259), GetModule is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation that creates a new Web module instance.

See Also

ComponentClass (see page 3264)

CreateMode (see page 3265)

CacheMode (see page 3264)

13.1.3.2.6 TAbstractWebModuleFactory.GetModuleName Method

Returns the value of the ModuleName (see page 3265) property.

Pascal

```
function GetModuleName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetModuleName() = 0;
```

Description

GetModuleName is the protected read implementation of the ModuleName (see page 3265) property.

In TAbstractWebModuleFactory (see page 3259), GetModuleName is abstract pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ModuleName (see page 3265) property.

See Also

ModuleName (see page 3265)

13.1.3.2.7 TAbstractWebModuleFactory.GetWebPageInfo Method

Returns the value of the WebPageInfo (see page 3266) property.

Pascal

```
function GetWebPageInfo: TAbstractWebPageInfo; virtual;
```

C++

```
virtual __fastcall TAbstractWebPageInfo GetWebPageInfo();
```

Description

GetWebPageInfo is the protected read implementation of the WebPageInfo (see page 3266) property.

In TAbstractWebModuleFactory (see page 3259), GetWebPageInfo always returns nil (Delphi) or NULL (C++). Descendant classes that represent the factories for Web page modules override this method to return the Web page information object that describes the module's Web page.

See Also

WebPageInfo (see page 3266)

13.1.3.2.8 TAbstractWebModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; virtual; abstract;
```

C++

```
virtual __fastcall PreventDestruction() = 0;
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TAbstractWebModuleFactory (see page 3259), PreventDestruction is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation that prevents the Web module from being freed.

See Also

CacheMode (see page 3264)

13.1.3.3 TAbstractWebModuleFactory Properties

The properties of the TAbstractWebModuleFactory class are listed here.

Public Properties

	Name	Description
	CacheMode (see page 3264)	Indicates whether the factory can cache Web modules for reuse.
	ComponentClass (see page 3264)	Specifies the class type of the Web module that this factory creates.
	CreateMode (see page 3265)	Indicates when the factory creates instances of its Web module.
	IsAppModule (see page 3265)	Indicates whether the corresponding Web module is the main Web application module.
	ModuleName (see page 3265)	Specifies the name of the Web module class that this factory instantiates.
	WebPageInfo (see page 3266)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

13.1.3.3.1 TAbstractWebModuleFactory.CacheMode Property

Indicates whether the factory can cache Web modules for reuse.

Pascal

```
property CacheMode: TWebModuleCacheMode;
```

C++

```
__property TWebModuleCacheMode CacheMode;
```

Description

The Web module factory creates an instance of its Web module to handle a specific HTTP request message. After the WebSnap application has finished with that request, it can free the Web module instance, or cache it so that the next time it needs to create an instance of its Web module, it can simply reuse the cached instance.

When CacheMode is caCache, the WebSnap application caches Web module instances for reuse. This can improve performance.

When CacheMode is caDestroy, the WebSnap application frees Web module instances when they are no longer needed for the current request message. This is the safest approach if changes to the Web module or any of the components it contains that are made while processing a request message can interfere with the correct handling of subsequent request messages.

See Also

CreateMode (see page 3265)

PreventDestruction (see page 3263)

13.1.3.3.2 TAbstractWebModuleFactory.ComponentClass Property

Specifies the class type of the Web module that this factory creates.

Pascal

```
property ComponentClass: TComponentClass;
```

C++

```
__property TComponentClass ComponentClass;
```

Description

ComponentClass is the class type of the Web module that the factory object creates. This is a descendant of TWebDataModule, TWebAppDataModule, TWebPageModule, or TWebAppPageModule. The WebSnap application wizard or Web module wizard generates the class declaration for the specific Web module class associated with a Web module factory.

See Also

ModuleName (see page 3265)

WebPageInfo (see page 3266)

IsAppModule (see page 3265)

13.1.3.3.3 TAbstractWebModuleFactory.CreateMode Property

Indicates when the factory creates instances of its Web module.

Pascal

```
property CreateMode: TWebModuleCreateMode;
```

C++

```
__property TWebModuleCreateMode CreateMode;
```

Description

Read CreateMode to determine whether the factory creates its Web module for every request message (crAlways), or when the WebSnap application specifically needs the Web module for the current request (crOnDemand).

Note: When the Web module factory creates the Web application module, CreateMode is always crAlways. The Web application module is always required because it contains the components that dispatch HTTP messages. For Web module factories that create any other Web modules in the application, CreateMode reflects a flag passed to the constructor.

See Also

CacheMode (see page 3264)

IsAppModule (see page 3265)

13.1.3.3.4 TAbstractWebModuleFactory.IsAppModule Property

Indicates whether the corresponding Web module is the main Web application module.

Pascal

```
property IsAppModule: Boolean;
```

C++

```
__property Boolean IsAppModule;
```

Description

WebSnap applications read IsAppModule to determine whether a Web module factory creates the single application module. WebSnap applications include one and only one application module, which holds the components responsible for dispatching HTTP requests. The Web application module is an instance of TWebAppPageModule or TWebAppDataModule.

See Also

TWebAppPageModule

TWebAppDataModule

ComponentClass (see page 3264)

TPageDispatcher

13.1.3.3.5 TAbstractWebModuleFactory.ModuleName Property

Specifies the name of the Web module class that this factory instantiates.

Pascal

```
property ModuleName: string;
```

C++

```
__property UnicodeString ModuleName;
```

Description

ModuleName is the name of the Web module class that the factory object instantiates. That is, this is the name of the class for which the ComponentClass (see page 3264) property is a class reference.

See Also

ComponentClass (see page 3264)

WebPageInfo (see page 3266)

IsAppModule (see page 3265)

FindModuleName (see page 3270)

13.1.3.3.6 TAbstractWebModuleFactory.WebPageInfo Property

Describes the attributes of the Web page that the corresponding Web module generates, if any.

Pascal

```
property WebPageInfo: TAbstractWebPageInfo;
```

C++

```
__property TAbstractWebPageInfo WebPageInfo;
```

Description

When the Web module that the factory object creates can generate a Web page, WebPageInfo describes the attributes of the Web page, such as its name, title, access rights, and so on. This information is stored in the Web module factory so that it is available even when there is no current instance of the Web module.

When the Web module that the factory object creates does not generate a Web page, WebPageInfo is nil (Delphi) or NULL (C++).

See Also

ComponentClass (see page 3264)

13.1.4 TAbstractWebModuleList Class

TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.

Class Hierarchy

Pascal

```
TAbstractWebModuleList = class(TObject);
```

C++

```
class TAbstractWebModuleList : public TObject;
```

File

HTTPApp

Description

TAbstractWebModuleList is the base class for TWebModuleList, which lists every Web module that has been instantiated to serve the current request. It introduces properties and methods for accessing the Web modules in the application, as well as for the Web module factory objects that create those Web module instances. All the methods in TAbstractWebModuleList are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. TWebModuleList, a TAbstractWebModuleList descendant, overrides those abstract or pure virtual methods to provide a default implementation.

When a WebSnap application handles multiple requests concurrently, it includes multiple web modules lists, each on a separate thread and each associated with a different request.

The Web context, which can be accessed using the global WebContext function, provides access to the TAbstractWebModuleList object that is associated with the current request. Use the Web context's WebModules property to obtain a reference to the Web module list.

See Also

TWebModuleList

TWebModuleFactoryList

TAbstractWebModuleFactory (see page 3259)

WebContext

TAbstractWebContext

13.1.4.1 TAbstractWebModuleList Members

The following tables list the members exposed by TAbstractWebModuleList.

Protected Methods

	Name	Description
	GetFactory (see page 3270)	This is GetFactory, a member of class TAbstractWebModuleList.
	GetFactoryCount (see page 3270)	
	GetItem (see page 3271)	This is GetItem, a member of class TAbstractWebModuleList.
	GetItemCount (see page 3271)	
	GetOnModuleAdded (see page 3271)	
	SetOnModuleAdded (see page 3271)	

Public Events

	Name	Description
	OnModuleAdded (see page 3274)	Occurs when a new Web module is added to the Items (see page 3273) property array.

Public Methods

	Name	Description
	AddModuleClass (see page 3268)	Creates a new Web module, given its class, and adds it to the Items (see page 3273) property array.
	AddModuleName (see page 3269)	Creates a new Web module, given its name, and adds it to the Items (see page 3273) property array.
	FindModuleClass (see page 3269)	Locates a Web module given its class reference.
	FindModuleName (see page 3270)	Locates a Web module given its name.

Public Properties

	Name	Description
R	Factory (see page 3272)	Lists the Web module factories for all the Web modules available in the application.
R	FactoryCount (see page 3272)	Indicates the number of Web module factories listed in the Factory (see page 3272) property array.
R	ItemCount (see page 3273)	Indicates the number of Web modules that have been instantiated in the current thread.
R	Items (see page 3273)	Lists the instantiated Web modules in the current thread.

13.1.4.2 TAbstractWebModuleList Methods

The methods of the TAbstractWebModuleList class are listed here.

Protected Methods

	Name	Description
A	GetFactory (see page 3270)	This is GetFactory, a member of class TAbstractWebModuleList.
A	GetFactoryCount (see page 3270)	
A	GetItem (see page 3271)	This is GetItem, a member of class TAbstractWebModuleList.
A	GetItemCount (see page 3271)	
A	GetOnModuleAdded (see page 3271)	
A	SetOnModuleAdded (see page 3271)	

Public Methods

	Name	Description
A	AddModuleClass (see page 3268)	Creates a new Web module, given its class, and adds it to the Items (see page 3273) property array.
A	AddModuleName (see page 3269)	Creates a new Web module, given its name, and adds it to the Items (see page 3273) property array.
A	FindModuleClass (see page 3269)	Locates a Web module given its class reference.
A	FindModuleName (see page 3270)	Locates a Web module given its name.

13.1.4.2.1 TAbstractWebModuleList.AddModuleClass Method

Creates a new Web module, given its class, and adds it to the Items (see page 3273) property array.

Pascal

```
function AddModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * AddModuleClass(TComponentClass AClass) = 0;
```

Description

AddModuleClass searches for a Web module factory that can create a Web module of the specified class. If it finds a match, it instantiates the Web module and adds it to the Items (see page 3273) property array.

AClass is a class reference to the Web module you want to instantiate. It should not match the class type of any Web module already listed in the Items (see page 3273) property array.

AddModuleClass returns the new Web module instance. If the Factory (see page 3272) property array does not include a Web module factory that can create a Web module of the specified class, AddModuleClass returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 3266), AddModuleClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

AddModuleName (see page 3269)

Items (see page 3273)

Factory (see page 3272)

FindModuleClass (see page 3269)

13.1.4.2.2 TAbstractWebModuleList.AddModuleName Method

Creates a new Web module, given its name, and adds it to the Items (see page 3273) property array.

Pascal

```
function AddModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * AddModuleName(const UnicodeString AClass) = 0;
```

Description

AddModuleName searches for a Web module factory that can create a Web module of the specified name. If it finds a match, it instantiates the Web module and adds it to the Items (see page 3273) property array.

AName is the name of the Web module you want to instantiate. It should not match the name of any Web module already listed in the Items (see page 3273) property array.

AddModuleClass (see page 3268) returns the new Web module instance. If the Factory (see page 3272) property array does not include a Web module factory that can create a Web module of the specified name, AddModuleClass (see page 3268) returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 3266), AddModuleName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

AddModuleClass (see page 3268)

Items (see page 3273)

Factory (see page 3272)

FindModuleName (see page 3270)

13.1.4.2.3 TAbstractWebModuleList.FindModuleClass Method

Locates a Web module given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass) = 0;
```

Description

Call FindModuleClass to locate the Web module instance of a given class type. FindModuleClass searches the Items (see page 3273) property array for a Web module instance of the specified class.

AClass is a class reference to the class type of the desired Web module.

FindModuleClass returns the requested Web module instance if it exists. If there is no Web module of the specified class, FindModuleClass returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 3266), FindModuleClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

FindModuleName (see page 3270)

Items (see page 3273)

13.1.4.2.4 TAbstractWebModuleList.FindModuleName Method

Locates a Web module given its name.

Pascal

```
function FindModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AClass) = 0;
```

Description

Call FindModuleName to locate the Web module instance of a given name. FindModuleClass (see page 3269) searches the Items (see page 3273) property array for a Web module instance of the specified name.

AName is the name of the desired Web module.

FindModuleName returns the requested Web module instance if it exists. If there is no Web module with the specified name, FindModuleClass (see page 3269) returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 3266), FindModuleClass (see page 3269) is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

FindModuleClass (see page 3269)

Items (see page 3273)

13.1.4.2.5 TAbstractWebModuleList.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractWebModuleFactory; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebModuleFactory GetFactory(int I) = 0;
```

Description

This is GetFactory, a member of class TAbstractWebModuleList.

13.1.4.2.6 TAbstractWebModuleList.GetFactoryCount Method

Pascal

```
function GetFactoryCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetFactoryCount() = 0;
```

See Also

GetFactory (see page 3270)

FactoryCount (see page 3272)

13.1.4.2.7 TAbstractWebModuleList.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * GetItem(int I) = 0;
```

Description

This is GetItem, a member of class TAbstractWebModuleList.

13.1.4.2.8 TAbstractWebModuleList.GetItemCount Method

Pascal

```
function GetItemCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemCount() = 0;
```

See Also

GetItem (see page 3271)

ItemCount (see page 3273)

13.1.4.2.9 TAbstractWebModuleList.GetOnModuleAdded Method

Pascal

```
function GetOnModuleAdded: TModuleAddedProc; virtual; abstract;
```

C++

```
virtual __fastcall TModuleAddedProc GetOnModuleAdded() = 0;
```

See Also

OnModuleAdded (see page 3274)

SetOnModuleAdded (see page 3271)

13.1.4.2.10 TAbstractWebModuleList.SetOnModuleAdded Method

Pascal

```
procedure SetOnModuleAdded(AProc: TModuleAddedProc); virtual; abstract;
```

C++

```
virtual __fastcall SetOnModuleAdded(TModuleAddedProc AProc) = 0;
```

See Also

GetOnModuleAdded (see page 3271)

OnModuleAdded (see page 3274)

13.1.4.3 TAbstractWebModuleList Properties

The properties of the TAbstractWebModuleList class are listed here.

Public Properties

	Name	Description
	Factory (see page 3272)	Lists the Web module factories for all the Web modules available in the application.
	FactoryCount (see page 3272)	Indicates the number of Web module factories listed in the Factory (see page 3272) property array.
	ItemCount (see page 3273)	Indicates the number of Web modules that have been instantiated in the current thread.
	Items (see page 3273)	Lists the instantiated Web modules in the current thread.

13.1.4.3.1 TAbstractWebModuleList.Factory Property

Lists the Web module factories for all the Web modules available in the application.

Pascal

```
property Factory [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory Factory[int I];
```

Description

Read Factory to access the factory objects that instantiate Web modules for the WebSnap application. Each factory object includes methods to create or (if an instance exists) access the corresponding Web module, as well as properties that provide information about the Web module it creates.

Use the FactoryCount (see page 3272) property in conjunction with Factory to iterate over all the Web module factory objects in the WebSnap application.

I is the index of the factory object to read, where 0 identifies the first factory, 1 identifies the second factory, and so on, until FactoryCount (see page 3272) – 1 identifies the last factory.

See Also

FactoryCount (see page 3272)

Items (see page 3273)

13.1.4.3.2 TAbstractWebModuleList.FactoryCount Property

Indicates the number of Web module factories listed in the Factory (see page 3272) property array.

Pascal

```
property FactoryCount: Integer;
```

C++

```
__property int FactoryCount;
```

Description

Read FactoryCount to determine the number of Web module factory object (and hence the potential number of Web modules) in the WebSnap application. FactoryCount provides an upper bound to the index of the Factory (see page 3272) property array.

See Also

Factory (see page 3272)

ItemCount (see page 3273)

13.1.4.3.3 TAbstractWebModuleList.ItemCount Property

Indicates the number of Web modules that have been instantiated in the current thread.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Read ItemCount to determine the number of instantiated Web modules in the current thread of the WebSnap application. ItemCount provides an upper bound to the index of the Items (see page 3273) property array.

Because ItemCount counts only the instantiated Web modules, its value is always less than or equal to the value of FactoryCount (see page 3272), which gives the number of Web module factories.

See Also

Items (see page 3273)

FactoryCount (see page 3272)

13.1.4.3.4 TAbstractWebModuleList.Items Property

Lists the instantiated Web modules in the current thread.

Pascal

```
property Items [I: Integer]: TComponent;
```

C++

```
__property TComponent * Items[int I];
```

Description

Read Items to access the Web modules that have been created to process the current HTTP request.

Use the ItemCount (see page 3273) property in conjunction with Items to iterate over all the Web modules in the current thread.

I is the index of the object to read, where 0 identifies the first Web module, 1 identifies the second Web module, and so on, until ItemCount (see page 3273) – 1 identifies the last Web module.

Note: In Delphi, Items is the default property of TAbstractWebModuleList (see page 3266). This means that the property name may be omitted. Thus the line

Note: MyModule := WebContext.WebModules.Items[I];

Note: can be written

Note: MyModule := WebContext.WebModules[I];

Note: In C++, a similar effect to the use of Items as a default property can be achieved using the [] operator.

See Also

ItemCount (see page 3273)

Factory (see page 3272)

operator_sb

13.1.4.4 TAbstractWebModuleList Events

The events of the TAbstractWebModuleList class are listed here.

Public Events

	Name	Description
	OnModuleAdded (see page 3274)	Occurs when a new Web module is added to the Items (see page 3273) property array.

13.1.4.4.1 TAbstractWebModuleList.OnModuleAdded Event

Occurs when a new Web module is added to the Items (see page 3273) property array.

Pascal

```
property OnModuleAdded: TModuleAddedProc;
```

C++

```
__property TModuleAddedProc OnModuleAdded;
```

Description

Write an OnModuleAdded event handler to take specific action when a Web module is instantiated in the current Web context. When the WebSnap application creates a new Web module instance, it generates an OnModuleAdded event immediately after adding the instance to the Items (see page 3273) property array.

Note: Do not write an OnModuleAdded event handler if the application contains a TWebAppComponents object. TWebAppComponents uses this event to initialize the Web modules when they are added.

See Also

Items (see page 3273)

AddModuleClass (see page 3268)

AddModuleName (see page 3269)

TWebAppComponents

13.1.5 TAbstractWebPageInfo Class

TAbstractWebPageInfo represents the attributes of a Web page module.

Class Hierarchy

Pascal

```
TAbstractWebPageInfo = class;
```

C++

```
class TAbstractWebPageInfo;
```

File

HTTPApp

Description

TAbstractWebPageInfo is the abstract base class for Web page information objects. It introduces a number of properties that describe the attributes of a Web page module, including

PageName (see page 3283), which is the logical name for the Web page the module generates.

IsPublished (see page 3282), which indicates whether the WebSnap application automatically displays the page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 3283).

IsLoginRequired (see page 3281), which indicates whether the user must log in to access the Web page.

ViewAccess (see page 3284), which can be used to restrict access to the Web page.

PageTitle (see page 3284), which is the title that appears on the Web page.

PageFile (see page 3283), which identifies an associated HTML template that the Web page module uses to generate its Web page.

PageHREF (see page 3283), which identifies the HREF to use for references to the page that appear in hypertext links.

PageDescription (see page 3282), which describes the Web page.

Do not create instances of TAbstractWebPageInfo. It includes abstract or, in C++ terminology, pure virtual methods (methods that have no implementation) that must be overridden in a descendant class.

Web page information objects are used to set the attributes of a Web page module. An instance of a TAbstractWebPageInfo descendant, such as TWebPageInfo, is supplied to the constructor of the factory object that creates instances of the Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard. You can change this generated code, which appears in the generated unit for the Web page module, to alter the properties of the Web page information object. For example, you may want to add a ViewAccess (see page 3284) string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageInfo

TAbstractWebContext

TWebAppPageModuleFactory

TWebPageModuleFactory

WebContext

13.1.5.1 TAbstractWebPageInfo Members

The following tables list the members exposed by TAbstractWebPageInfo.

Protected Methods

	Name	Description
	GetFactory (see page 3277)	Returns the value of the Factory (see page 3281) property.
V	GetIsLoginRequired (see page 3277)	Returns the value of the IsLoginRequired (see page 3281) property.
V	GetIsPublished (see page 3277)	Returns the value of the IsPublished (see page 3282) property.
V	GetPageDescription (see page 3278)	Returns the value of the PageDescription (see page 3282) property.
A	GetPageFile (see page 3278)	Returns the value of the PageFile (see page 3283) property.
V	GetPageHREF (see page 3278)	Returns the value of the PageHREF (see page 3283) property.
A	GetPageName (see page 3279)	Returns the value of the PageName (see page 3283) property.
V	GetPageTitle (see page 3279)	Returns the value of the PageTitle (see page 3284) property.
V	GetViewAccess (see page 3279)	Returns the value of the ViewAccess (see page 3284) property.
V	SetFactory (see page 3280)	Sets the value of the Factory (see page 3281) property.

Public Properties

	Name	Description
	Factory (see page 3281)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 3281)	Indicates whether callers must log in before accessing the associated Web page.
R	IsPublished (see page 3282)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 3283).
R	PageDescription (see page 3282)	Describes the Web page.
R	PageFile (see page 3283)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 3283)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 3283)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 3284)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 3284)	Indicates the view access string that controls access to the associated Web page.

13.1.5.2 TAbstractWebPageInfo Methods

The methods of the TAbstractWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetFactory (see page 3277)	Returns the value of the Factory (see page 3281) property.
V	GetIsLoginRequired (see page 3277)	Returns the value of the IsLoginRequired (see page 3281) property.
V	GetIsPublished (see page 3277)	Returns the value of the IsPublished (see page 3282) property.
V	GetPageDescription (see page 3278)	Returns the value of the PageDescription (see page 3282) property.
A	GetPageFile (see page 3278)	Returns the value of the PageFile (see page 3283) property.
V	GetPageHREF (see page 3278)	Returns the value of the PageHREF (see page 3283) property.
A	GetPageName (see page 3279)	Returns the value of the PageName (see page 3283) property.
V	GetPageTitle (see page 3279)	Returns the value of the PageTitle (see page 3284) property.

	GetViewAccess (see page 3279)	Returns the value of the ViewAccess (see page 3284) property.
	SetFactory (see page 3280)	Sets the value of the Factory (see page 3281) property.

13.1.5.2.1 TAbstractWebPageInfo.GetFactory Method

Returns the value of the Factory (see page 3281) property.

Pascal

```
function GetFactory: TAbstractWebPageModuleFactory;
```

C++

```
__fastcall TAbstractWebPageModuleFactory GetFactory();
```

Description

GetFactory is the protected read implementation of the Factory (see page 3281) property.

See Also

Factory (see page 3281)

SetFactory (see page 3280)

13.1.5.2.2 TAbstractWebPageInfo.GetIsLoginRequired Method

Returns the value of the IsLoginRequired (see page 3281) property.

Pascal

```
function GetIsLoginRequired: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsLoginRequired();
```

Description

GetIsLoginRequired is the protected read implementation of the IsLoginRequired (see page 3281) property.

In TAbstractWebPageInfo (see page 3275), GetIsLoginRequired always returns false. Descendant classes override this method to indicate when users must log in to access the associated Web page.

See Also

IsLoginRequired (see page 3281)

13.1.5.2.3 TAbstractWebPageInfo.GetIsPublished Method

Returns the value of the IsPublished (see page 3282) property.

Pascal

```
function GetIsPublished: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsPublished();
```

Description

GetIsPublished is the protected read implementation of the IsPublished (see page 3282) property.

In TAbstractWebPageInfo (see page 3275), GetIsPublished always returns false. Descendant classes override this method to indicate when the page dispatcher can automatically forward HTTP request messages that address the associated Web module.

See Also

IsPublished (see page 3282)

13.1.5.2.4 TAbstractWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription (see page 3282) property.

Pascal

```
function GetPageDescription: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the protected read implementation of the PageDescription (see page 3282) property.

In TAbstractWebPageInfo (see page 3275), GetPageDescription returns the value of the PageName (see page 3283) property. Descendant classes override this method to allow page descriptions that differ from the Web page's logical name.

See Also

PageDescription (see page 3282)

PageName (see page 3283)

13.1.5.2.5 TAbstractWebPageInfo.GetPageFile Method

Returns the value of the PageFile (see page 3283) property.

Pascal

```
function GetPageFile: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetPageFile() = 0;
```

Description

GetPageFile is the protected read implementation of the PageFile (see page 3283) property.

In TAbstractWebPageInfo (see page 3275), GetPageFile is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to provide an implementation for the PageFile (see page 3283) property.

See Also

PageFile (see page 3283)

13.1.5.2.6 TAbstractWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF (see page 3283) property.

Pascal

```
function GetPageHREF: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the protected read implementation of the PageHREF (see page 3283) property.

In TAbstractWebPageInfo (see page 3275), GetPageHREF returns the value of the PageName (see page 3283) property. Descendant classes override this method to allow HREF values that differ from the Web page's logical name.

See Also

PageHREF (see page 3283)

PageName (see page 3283)

13.1.5.2.7 TAbstractWebPageInfo.GetPageName Method

Returns the value of the PageName (see page 3283) property.

Pascal

```
function GetPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetPageName() = 0;
```

Description

GetPageName is the protected read implementation of the PageName (see page 3283) property.

In TAbstractWebPageInfo (see page 3275), GetPageName is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to provide an implementation for the PageName (see page 3283) property.

See Also

PageName (see page 3283)

13.1.5.2.8 TAbstractWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle (see page 3284) property.

Pascal

```
function GetPageTitle: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the protected read implementation of the PageTitle (see page 3284) property.

In TAbstractWebPageInfo (see page 3275), GetPageTitle returns the value of the PageName (see page 3283) property. Descendant classes override this method to allow Web pages to have titles that differ from their logical name.

See Also

PageTitle (see page 3284)

PageName (see page 3283)

13.1.5.2.9 TAbstractWebPageInfo.GetViewAccess Method

Returns the value of the ViewAccess (see page 3284) property.

Pascal

```
function GetViewAccess: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetViewAccess();
```

Description

GetViewAccess is the protected read implementation of the ViewAccess (see page 3284) property.

In TAbstractWebPageInfo (see page 3275), GetViewAccess returns an empty string. Descendant classes override this method to allow Web pages to have view access strings.

See Also

ViewAccess (see page 3284)

13.1.5.2.10 TAbstractWebPageInfo.SetFactory Method

Sets the value of the Factory (see page 3281) property.

Pascal

```
procedure SetFactory(AFactory: TAbstractWebPageModuleFactory); virtual;
```

C++

```
virtual __fastcall SetFactory(TAbstractWebPageModuleFactory AFactory);
```

Description

SetFactory is the protected write implementation of the Factory (see page 3281) property.

AFactory is the Web page module factory object that instantiates the Web page module that this object describes.

See Also

Factory (see page 3281)

GetFactory (see page 3277)

13.1.5.3 TAbstractWebPageInfo Properties

The properties of the TAbstractWebPageInfo class are listed here.

Public Properties

	Name	Description
	Factory (see page 3281)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 3281)	Indicates whether callers must log in before accessing the associated Web page.
R	IsPublished (see page 3282)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 3283).
R	PageDescription (see page 3282)	Describes the Web page.
R	PageFile (see page 3283)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 3283)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 3283)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 3284)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 3284)	Indicates the view access string that controls access to the associated Web page.

13.1.5.3.1 TAbstractWebPageInfo.Factory Property

Specifies the factory object that creates instances of the Web page module this object describes.

Pascal

```
property Factory: TAbstractWebPageModuleFactory;
```

C++

```
__property TAbstractWebPageModuleFactory Factory;
```

Description

Factory indicates the factory object that creates instances of the associated Web page module. If the associated Web page module is also the Web application module, Factory is an instance of TWebAppPageModuleFactory. Otherwise, it is an instance of TWebPageModuleFactory.

This property is set by the factory object's constructor.

See Also

TWebAppPageModuleFactory

TWebPageModuleFactory

Create_d

Create_c

13.1.5.3.2 TAbstractWebPageInfo.IsLoginRequired Property

Indicates whether callers must log in before accessing the associated Web page.

Pascal

```
property IsLoginRequired: Boolean;
```

C++

```
__property Boolean IsLoginRequired;
```

Description

Read IsLoginRequired to determine whether users must log in before accessing the associated Web page. The page dispatcher component (TPageDispatcher) checks this value to determine whether it can forward requests to the associated Web page when the caller has not yet logged in. The application can overrule the IsLoginRequired property by supplying an OnIsLoginRequired event handler to the page dispatcher.

In TAbstractWebPageInfo (see page 3275), IsLoginRequired is always false. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

Create_c

Create_d

IsPublished (see page 3282)

ViewAccess (see page 3284)

OnIsLoginRequired

TPageDispatcher

13.1.5.3.3 TAbstractWebPageInfo.IsPublished Property

Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 3283).

Pascal

```
property IsPublished: Boolean;
```

C++

```
__property Boolean IsPublished;
```

Description

Read IsPublished to determine whether the page dispatcher (TPageDispatcher) automatically calls on the associated Web page module when it receives HTTP request messages that address it by name. HTTP request messages address a Web page module when the pathinfo portion of their target URL matches the value of the PageName (see page 3283) property.

When IsPublished is false, the WebSnap application must explicitly retrieve the content that the Web page module generates in order to use the Web page module.

In TAbstractWebPageInfo (see page 3275), IsPublished is always false. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

PageName (see page 3283)

Create_d

Create_c

TPageDispatcher

13.1.5.3.4 TAbstractWebPageInfo.PageDescription Property

Describes the Web page.

Pascal

```
property PageDescription: string;
```

C++

```
__property UnicodeString PageDescription;
```

Description

Read PageDescription to obtain a brief description of the Web page that the associated Web page module generates. This property can be used, for example, to provide descriptions for a dynamically generated catalog of available Web pages.

In TAbstractWebPageInfo (see page 3275), PageDescription is the same as the value of the PageName (see page 3283) property.

See Also

PageName (see page 3283)

PageTitle (see page 3284)

13.1.5.3.5 TAbstractWebPageInfo.PageFile Property

Identifies an associated file that the Web page module uses to generate its Web page.

Pascal

```
property PageFile: string;
```

C++

```
__property UnicodeString PageFile;
```

Description

Read PageFile to identify the file, if any, that the associated Web page module uses when generating its Web page. This can be the Web page that the Web page module returns, or it can be a template that the Web page module modifies to generate its final product.

In TAbstractWebPageInfo (see page 3275), the property read access method for PageFile is abstract pure virtual. Descendant classes must override the protected GetPageFile (see page 3278) method to provide an implementation for this property.

13.1.5.3.6 TAbstractWebPageInfo.PageHREF Property

Identifies the HREF to use for references to the associated Web page that appear in hypertext links.

Pascal

```
property PageHREF: string;
```

C++

```
__property UnicodeString PageHREF;
```

Description

Read PageHREF to obtain the string that should be inserted in HTML code to create a hypertext link to the associated Web page.

In TAbstractWebPageInfo (see page 3275), PageHREF is the same as the value of the PageName (see page 3283) property.

See Also

PageName (see page 3283)

13.1.5.3.7 TAbstractWebPageInfo.PageName Property

Denotes the logical name for the Web page that the associated module generates

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Read PageName to determine the name of the associated Web page. This name is used to identify that page in server-side scripts, and as the pathinfo portion of the URL for the page when the IsPublished (see page 3282) property is true.

In TAbstractWebPageInfo (see page 3275), the property read access method for PageName is abstract pure virtual. Descendant classes must override the protected GetPageName (see page 3279) method to provide an implementation for this property.

See Also

PageTitle (see page 3284)

PageHREF (see page 3283)

IsPublished (see page 3282)

13.1.5.3.8 TAbstractWebPageInfo.PageTitle Property

Indicates the title that appears on the associated Web page.

Pascal

```
property PageTitle: string;
```

C++

```
__property UnicodeString PageTitle;
```

Description

Read PageTitle to determine the title that appears on the associated Web page when viewed from a user's browser.

In TAbstractWebPageInfo (see page 3275), PageTitle is the same as the value of the PageName (see page 3283) property.

See Also

PageName (see page 3283)

13.1.5.3.9 TAbstractWebPageInfo.ViewAccess Property

Indicates the view access string that controls access to the associated Web page.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

Read ViewAccess to obtain the view access string that controls which users have access to the associated Web page. When a user tries to access the associated Web page, the page dispatcher checks the user's access rights against this string by calling the HasViewRights method of the end user adapter. Adapters in the Web page module check execute and modify rights against this string by calling the HasExecuteRights and HasModifyRights methods of the end user adapter.

In TAbstractWebPageInfo (see page 3275), ViewAccess is always an empty string. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

HasExecuteRights

HasModifyRights

HasViewRights

13.1.6 TAbstractWebPageModuleFactory Class

TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.

Class Hierarchy

Pascal

```
TAbstractWebPageModuleFactory = class(TAbstractWebModuleFactory);
```

C++

```
class TAbstractWebPageModuleFactory : public TAbstractWebModuleFactory;
```

File

HTTPApp

Description

All WebSnap modules are registered with the WebSnap application using a factory object. In Delphi, the factory object is typically created in the initialization section of the module source file. The factory object is responsible for creating Web module instances. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instances for reuse with subsequent request messages.

TAbstractWebPageModuleFactory is the base class for all Web module factory objects that generate Web page modules. In addition to holding components that perform tasks for the WebSnap application, Web page modules can generate a Web page. The Web page is described by a Web page information object, which the Web page module factory makes available as its WebPageInfo (see page 3266) property.

Do not create instances of TAbstractWebPageModuleFactory. Most of its methods are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. Instead, create instances of a TAbstractWebPageModuleFactory descendant such as TWebPageModuleFactory or TWebAppPageModuleFactory.

See Also

TWebAppPageModuleFactory

TWebPageModuleFactory

TWebRequestHandler

TWebPageInfo

13.1.6.1 TAbstractWebPageModuleFactory Members

The following tables list the members exposed by TAbstractWebPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 3260)	Returns the value of the CacheMode (see page 3264) property.
	GetComponentClass (see page 3261)	Returns the value of the ComponentClass (see page 3264) property.
	GetCreateMode (see page 3261)	Returns the value of the CreateMode (see page 3265) property.
	GetIsAppModule (see page 3261)	Returns the value of the IsAppModule (see page 3265) property.

	GetModuleName (see page 3262)	Returns the value of the ModuleName (see page 3265) property.
	GetWebPageInfo (see page 3287)	Returns the value of the WebPageInfo (see page 3266) property.

Public Methods

	Name	Description
	Create (see page 3286)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 3286)	Disposes of an object instance.
	GetModule (see page 3262)	Creates an instance of the associated Web module.
	PreventDestruction (see page 3263)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
	CacheMode (see page 3264)	Indicates whether the factory can cache Web modules for reuse.
	ComponentClass (see page 3264)	Specifies the class type of the Web module that this factory creates.
	CreateMode (see page 3265)	Indicates when the factory creates instances of its Web module.
	IsAppModule (see page 3265)	Indicates whether the corresponding Web module is the main Web application module.
	ModuleName (see page 3265)	Specifies the name of the Web module class that this factory instantiates.
	WebPageInfo (see page 3266)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

13.1.6.2 TAbstractWebPageModuleFactory.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebPageInfo: TAbstractWebPageInfo);
```

C++

```
__fastcall TAbstractWebPageModuleFactory(TAbstractWebPageInfo AWebPageInfo);
```

Description

Do not create instances of TAbstractWebPageModuleFactory (see page 3285). Instead, construct descendants of TAbstractWebPageModuleFactory (see page 3285) using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy

new

13.1.6.3 TAbstractWebPageModuleFactory.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractWebPageModuleFactory();
```

Description

Do not call ~TAbstractWebPageModuleFactory (see page 3285) directly. Instead, use the delete keyword when destroying VCL objects.

~TAbstractWebPageModuleFactory (see page 3285) deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

13.1.6.4 TAbstractWebPageModuleFactory Methods

The methods of the TAbstractWebPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetWebPageInfo (see page 3287)	Returns the value of the WebPageInfo (see page 3266) property.

13.1.6.4.1 TAbstractWebPageModuleFactory.GetWebPageInfo Method

Returns the value of the WebPageInfo (see page 3266) property.

Pascal

```
function GetWebPageInfo: TAbstractWebPageInfo; override;
```

C++

```
virtual __fastcall TAbstractWebPageInfo GetWebPageInfo();
```

Description

GetWebPageInfo is the protected read implementation of the WebPageInfo (see page 3266) property. It returns the Web page information object that is supplied to the TAbstractWebPageModuleFactory (see page 3285) constructor.

See Also

WebPageInfo

Create_c

13.1.7 TAbstractWebRequestFile Class

TAbstractWebRequestFile describes an uploaded file from an HTTP request.

Class Hierarchy

Pascal

```
TAbstractWebRequestFile = class(TObject);
```

C++

```
class TAbstractWebRequestFile : public TObject;
```

File

HTTPApp

Description

TAbstractWebRequestFile introduces properties to describe and provide access to an uploaded file from an HTTP request. These include

FileName (see page 3291), which gives the name of the uploaded file.

FieldName (see page 3291), which gives the name of the field whose value is the uploaded file.

ContentType (see page 3291), which describes how the file data is encoded.

Stream (see page 3292), which is a stream object from which you can read the file content.

The methods that implement these properties are all abstract or, in C++ terminology, pure virtual. This means that the Web server application does not instantiate TAbstractWebRequestFile. Instead, it uses a TAbstractWebRequestFile descendant such as TWebRequestFile (which is included in WebSnap).

Each TAbstractWebRequestFile descendant describes a single uploaded file. A collection of all the uploaded files from the current request is available as the Files property of TWebRequest (see page 3358). In addition, in WebSnap applications, adapter fields that work with uploaded files surface those files using a TUpdateFileList object or IActionFieldValues interface.

Note: Support for uploaded files is only included in WebSnap. For other Web server applications, this class exists only as a placeholder for the Files property on TWebRequest (see page 3358).

See Also

TWebRequestFile (see page 6696)

TAbstractWebRequestFiles (see page 3292)

IActionFieldValues

TUpdateFileList

TAdapterFileField

13.1.7.1 TAbstractWebRequestFile Members

The following tables list the members exposed by TAbstractWebRequestFile.

Protected Methods

	Name	Description
	GetContentType (see page 3289)	Returns the value of the ContentType (see page 3291) property.
	GetFieldName (see page 3289)	Returns the value of the FieldName (see page 3291) property.
	GetFileName (see page 3290)	Returns the value of the FileName (see page 3291) property.
	GetStream (see page 3290)	Returns the value of the Stream (see page 3292) property.

Public Properties

	Name	Description
	ContentType (see page 3291)	Describes how the file data is encoded.
	FieldName (see page 3291)	Indicates the name of the adapter field whose value is the contents of the file.
	FileName (see page 3291)	Indicates the name of the uploaded file.
	Stream (see page 3292)	Provides access to the contents of the uploaded file.

13.1.7.2 TAbstractWebRequestFile Methods

The methods of the TAbstractWebRequestFile class are listed here.

Protected Methods

	Name	Description
	GetContentType (see page 3289)	Returns the value of the ContentType (see page 3291) property.
	GetFieldName (see page 3289)	Returns the value of the FieldName (see page 3291) property.
	GetFileName (see page 3290)	Returns the value of the FileName (see page 3291) property.
	GetStream (see page 3290)	Returns the value of the Stream (see page 3292) property.

13.1.7.2.1 TAbstractWebRequestFile.GetContentType Method

Returns the value of the ContentType (see page 3291) property.

Pascal

```
function GetContentType: AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetContentType() = 0;
```

Description

GetContentType is the protected read implementation for the ContentType (see page 3291) property.

In TAbstractWebRequestFile (see page 3288), GetContentType is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ContentType (see page 3291) property.

See Also

ContentType (see page 3291)

13.1.7.2.2 TAbstractWebRequestFile.GetFieldName Method

Returns the value of the FieldName (see page 3291) property.

Pascal

```
function GetFieldName: AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFieldName() = 0;
```

Description

GetFieldName is the protected read implementation for the FieldName (see page 3291) property.

In TAbstractWebRequestFile (see page 3288), GetFieldName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FieldName (see page 3291) property.

See Also

FieldName (see page 3291)

13.1.7.2.3 TAbstractWebRequestFile.GetFileName Method

Returns the value of the FileName (see page 3291) property.

Pascal

```
function GetFileName: AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFileName() = 0;
```

Description

GetFileName is the protected read implementation for the FileName (see page 3291) property.

In TAbstractWebRequestFile (see page 3288), GetFileName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FileName (see page 3291) property.

See Also

FileName (see page 3291)

13.1.7.2.4 TAbstractWebRequestFile.GetStream Method

Returns the value of the Stream (see page 3292) property.

Pascal

```
function GetStream: TStream; virtual; abstract;
```

C++

```
virtual __fastcall TStream GetStream() = 0;
```

Description

GetStream is the protected read implementation for the Stream (see page 3292) property.

In TAbstractWebRequestFile (see page 3288), GetStream is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Stream (see page 3292) property.

See Also

Stream (see page 3292)

13.1.7.3 TAbstractWebRequestFile Properties

The properties of the TAbstractWebRequestFile class are listed here.

Public Properties

	Name	Description
	ContentType (see page 3291)	Describes how the file data is encoded.
	FieldName (see page 3291)	Indicates the name of the adapter field whose value is the contents of the file.
	FileName (see page 3291)	Indicates the name of the uploaded file.
	Stream (see page 3292)	Provides access to the contents of the uploaded file.

13.1.7.3.1 TAbstractWebRequestFile.ContentType Property

Describes how the file data is encoded.

Pascal

```
property ContentType: AnsiString;
```

C++

```
__property AnsiString ContentType;
```

Description

Read ContentType to determine the format of the data you can read from the Stream (see page 3292) property. ContentType is a string of the format Type/SubType, which names the MIME type and subtype used to encode the file contents.

See Also

Stream (see page 3292)

13.1.7.3.2 TAbstractWebRequestFile.FieldName Property

Indicates the name of the adapter field whose value is the contents of the file.

Pascal

```
property FieldName: AnsiString;
```

C++

```
__property AnsiString FieldName;
```

Description

Read FieldName to identify the adapter field for which this file was uploaded. This is the same as the FieldName property of the adapter field object.

See Also

FieldName

FileName (see page 3291)

13.1.7.3.3 TAbstractWebRequestFile.FileName Property

Indicates the name of the uploaded file.

Pascal

```
property FileName: AnsiString;
```

C++

```
__property AnsiString FileName;
```

Description

Read FileName to obtain the name of the uploaded file.

See Also

Stream (see page 3292)

13.1.7.3.4 TAbstractWebRequestFile.Stream Property

Provides access to the contents of the uploaded file.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Use Stream to read the content of the uploaded file. Call the Read method of this stream to read the file contents.

Note: The ContentType (see page 3291) property indicates the format used to encode the file contents that you read from Stream.

See Also

FileName (see page 3291)

ContentType (see page 3291)

Read

13.1.8 TAbstractWebRequestFiles Class

TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 3288) objects.

Class Hierarchy

Pascal

```
TAbstractWebRequestFiles = class(TObject);
```

C++

```
class TAbstractWebRequestFiles : public TObject;
```

File

HTTPApp

Description

TAbstractWebRequestFiles is the base class for objects that manage a set of Web request files objects

(TAbstractWebRequestFile (see page 3288) descendants). Each Web request file object in the collection describes an uploaded file that was extracted from the current HTTP request object.

TAbstractWebRequestFiles should not be instantiated. The methods that implement its properties are abstract or, in C++ terminology, pure virtual. Instead, applications use a TAbstractWebRequestFiles descendant such as TWebRequestFiles (which is included in WebSnap).

Note: Support for uploaded files is only included in WebSnap. For other Web server applications, this class exists only as a placeholder for the Files property on TWebRequest (see page 3358).

See Also

TAbstractWebRequestFile (see page 3288)

TWebRequest (see page 3358)

TUpdateFileList

TWebRequestFiles (see page 6700)

13.1.8.1 TAbstractWebRequestFiles Members

The following tables list the members exposed by TAbstractWebRequestFiles.

Protected Methods

	Name	Description
	GetCount (see page 3293)	Returns the value of the Count (see page 3294) property.
	GetItem (see page 3294)	Returns the value of the Items (see page 3295) property.

Public Properties

	Name	Description
	Count (see page 3294)	Indicates the number of uploaded files in the collection.
	Items (see page 3295)	Lists the uploaded files in the collection.

13.1.8.2 TAbstractWebRequestFiles Methods

The methods of the TAbstractWebRequestFiles class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 3293)	Returns the value of the Count (see page 3294) property.
	GetItem (see page 3294)	Returns the value of the Items (see page 3295) property.

13.1.8.2.1 TAbstractWebRequestFiles.GetCount Method

Returns the value of the Count (see page 3294) property.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCount is the protected read implementation of the Count (see page 3294) property.

In TAbstractWebRequestFile (see page 3288), GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 3294) property.

See Also

Count (see page 3294)

13.1.8.2.2 TAbstractWebRequestFiles.GetItem Method

Returns the value of the Items (see page 3295) property.

Pascal

```
function GetItem(I: Integer): TAbstractWebRequestFile; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebRequestFile GetItem(int I) = 0;
```

Description

GetItem is the protected read implementation of the Items (see page 3295) property.

I is the index of the Web request file object to return.

In TAbstractWebRequestFile (see page 3288), GetCount (see page 3293) is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 3294) property.

13.1.8.3 TAbstractWebRequestFiles Properties

The properties of the TAbstractWebRequestFiles class are listed here.

Public Properties

	Name	Description
R	Count (see page 3294)	Indicates the number of uploaded files in the collection.
R	Items (see page 3295)	Lists the uploaded files in the collection.

13.1.8.3.1 TAbstractWebRequestFiles.Count Property

Indicates the number of uploaded files in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of files in this collection. Count provides an upper bound to the index of the Items (see page 3295) property.

See Also

Items (see page 3295)

13.1.8.3.2 TAbstractWebRequestFiles.Items Property

Lists the uploaded files in the collection.

Pascal

```
property Items [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Items[int I];
```

Description

Read Items to access the descriptions of the individual uploaded files in the collection.

I is the index of the file to access, where 0 indicates the first file, 1 indicates the second file, and so on. The value of I must be less than the value of the Count (see page 3294) property.

Note: Items is the default property of TAbstractWebRequestFiles (see page 3292). This means that the property name can be omitted when using this property. Thus, the line

Note: Upload := WebContext.Request.Files.Items[0];

Note: can also be written

Note: Upload := WebContext.Request.Files[0];

See Also

Count (see page 3294)

Files (see page 3379)

operator_sb

13.1.9 TAbstractWebSession Class

TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.

Class Hierarchy

Pascal

```
TAbstractWebSession = class(TObject);
```

C++

```
class TAbstractWebSession : public TObject;
```

File

HTTPApp

Description

TAbstractWebSession is the base class for all Web session objects. These include active sessions, which are represented by TWebSession objects, and invalid sessions (TInvalidWebSession), which are used to represent sessions that can no longer be used because they have expired.

A Web session has an ID and a list of name/value pairs. The ID is a random number that uniquely identifies the session. The name/value pairs are used to associate information with the ID. The values are variants. You can use these named Variants to store application-defined information that is associated with a single user. In this way, the information can persist over multiple calls from the same user.

The session ID is stored on the end user's computer using a cookie. When a request is received, the WebSnap application examines the cookie. If a session associated with this cookie exists, then it becomes the current session.

There is no need to instantiate session objects. A Session is created automatically when the WebSnap application receives an HTTP request message.

When the current session is terminated, its session ID and name/value pairs are discarded. For example, the current session is typically terminated when the user logs out. Sessions can also be terminated when they are not accessed within a specified time-out period, or when the application explicitly calls the session's Terminate (see page 3299) method. A new session is created after a session has been terminated.

See Also

TSessionsService

TAbstractWebContext

TWebSession

13.1.9.1 TAbstractWebSession Members

The following tables list the members exposed by TAbstractWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 3297)	Returns the value of the SessionID (see page 3300) property.
	GetTimeoutMinutes (see page 3297)	Returns the value of the TimeoutMinutes (see page 3300) property.
	GetValue (see page 3298)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 3298)	Sets the value of the TimoutMinutes property.
	SetValue (see page 3299)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 3299)	Terminates the session.
	UpdateResponse (see page 3299)	Adds the session ID to a cookie in a response message.

Public Properties

	Name	Description
	SessionID (see page 3300)	Uniquely identifies the Session.
	TimeoutMinutes (see page 3300)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 3301)	Lists the named values that the session stores.

13.1.9.2 TAbstractWebSession Methods

The methods of the TAbstractWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 3297)	Returns the value of the SessionID (see page 3300) property.
	GetTimeoutMinutes (see page 3297)	Returns the value of the TimeoutMinutes (see page 3300) property.
	GetValue (see page 3298)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 3298)	Sets the value of the TimoutMinutes property.
	SetValue (see page 3299)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 3299)	Terminates the session.
	UpdateResponse (see page 3299)	Adds the session ID to a cookie in a response message.

13.1.9.2.1 TAbstractWebSession.GetSessionID Method

Returns the value of the SessionID (see page 3300) property.

Pascal

```
function GetSessionID: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetSessionID() = 0;
```

Description

GetSessionID is the protected read implementation of the SessionID (see page 3300) property.

In TAbstractWebSession (see page 3295), GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID (see page 3300) property.

See Also

SessionID (see page 3300)

13.1.9.2.2 TAbstractWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes (see page 3300) property.

Pascal

```
function GetTimeoutMinutes: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetTimeoutMinutes() = 0;
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes (see page 3300) property.

In TAbstractWebSession (see page 3295), GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes (see page 3300) property.

See Also

TimeoutMinutes (see page 3300)

SetTimeoutMinutes (see page 3298)

13.1.9.2.3 TAbstractWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; virtual; abstract;
```

C++

```
virtual __fastcall Variant GetValue(const UnicodeString AName) = 0;
```

Description

GetValue is the protected read implementation of the Values (see page 3301) property.

AName is the name of the value to return.

In TAbstractWebSession (see page 3295), GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values (see page 3301) property.

See Also

Values (see page 3301)

SetValue (see page 3299)

13.1.9.2.4 TAbstractWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue) = 0;
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes (see page 3300) property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession (see page 3295), SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes (see page 3300) property.

See Also

GetTimeoutMinutes (see page 3297)

TimeoutMinutes (see page 3300)

13.1.9.2.5 TAbstractWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall SetValue(const UnicodeString AName, const Variant AValue) = 0;
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession (see page 3295), SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values (see page 3301) property.

See Also

GetValue (see page 3298)

Values (see page 3301)

13.1.9.2.6 TAbstractWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate; virtual; abstract;
```

C++

```
virtual __fastcall Terminate() = 0;
```

Description

Call Terminate to explicitly terminate a session. Once a session terminates, the application can no longer access its Values (see page 3301).

Sessions are terminated automatically when the application does not receive any requests for the session for TimoutMinutes minutes.

In TAbstractWebSession (see page 3295), Terminate is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation.

See Also

TimeoutMinutes (see page 3300)

Values (see page 3301)

13.1.9.2.7 TAbstractWebSession.UpdateResponse Method

Adds the session ID to a cookie in a response message.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); virtual; abstract;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse) = 0;
```

Description

Call UpdateResponse to add the value of the SessionID (see page 3300) property to one of the response message's cookies. This allows the user's next request message to use the Values (see page 3301) stored in this session.

AResponse is the Web response message to which a cookie with the session ID should be added.

See Also

SessionID (see page 3300)

Values (see page 3301)

13.1.9.3 TAbstractWebSession Properties

The properties of the TAbstractWebSession class are listed here.

Public Properties

	Name	Description
R	SessionID (see page 3300)	Uniquely identifies the Session.
	TimeoutMinutes (see page 3300)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 3301)	Lists the named values that the session stores.

13.1.9.3.1 TAbstractWebSession.SessionID Property

Uniquely identifies the Session.

Pascal

```
property SessionID: string;
```

C++

```
__property UnicodeString SessionID;
```

Description

SessionID is an automatically-generated identifier that uniquely identifies the session. The UpdateResponse (see page 3299) method adds this identifier to response messages, using a cookie. When the WebSnap application receives a request containing a cookie with a session ID, the WebSnap application uses the ID to recreate a copy of the previous session object.

See Also

UpdateResponse (see page 3299)

13.1.9.3.2 TAbstractWebSession.TimeoutMinutes Property

Indicates the time, in minutes, that a session can be idle before it terminates itself.

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

Set TimeoutMinutes to specify the number of minutes that can elapse before a session times out. If this number of minutes elapses with no request messages that call for the session, the session is terminated.

If you do not specify a value for TimeoutMinutes, the session times out after the default time-out period associated with the application's sessions service (TSessionsService).

See Also

Terminate (see page 3299)

DefaultTimeout

13.1.9.3.3 TAbstractWebSession.Values Property

Lists the named values that the session stores.

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const AName];
```

Description

Use Values to get or set the information that the session stores. Because sessions are associated with individual users, this information typically represents information about the user or choices the user makes. The Values property can be used to implement application-specific tasks, such as saving user preferences or maintaining a list user selections.

Each value is indexed by a name that indicates its meaning. The names can be any application-defined string.

See Also

TEndUserSessionAdapter

13.1.10 TContentParser Class

TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.

Class Hierarchy

Pascal

```
TContentParser = class(TAbstractContentParser);
```

C++

```
class TContentParser : public TAbstractContentParser;
```

File

HTTPApp

Description

Web request objects use a TContentParser object to parse the content of an HTTP request message when there is no other registered content parser that knows how to interpret that content. The Web request uses the content parser to assign values for its ContentFields and Files properties.

TContentParser does not know how to parse special request formats. It simply calls the Web request object's own ExtractContentFields method to extract named value strings for the request's ContentFields property. It does not parse uploaded files, but merely returns an empty list for the request's Files property.

See Also

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TMultipartContentParser (see page 6706)

TWebRequest (see page 3358)

13.1.10.1 TContentParser Members

The following tables list the members exposed by TContentParser.

Protected Properties

	Name	Description
	WebRequest (see page 3258)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
	CanParse (see page 3303)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	Create (see page 3256)	Creates and initializes a new TAbstractContentParser (see page 3255) object.
	Destroy (see page 3302)	Frees the memory associated with the TContentParser (see page 3301) object.
	GetContentFields (see page 3303)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 3304)	Returns a set of uploaded files that the parser extracts from the current Web request.

13.1.10.2 TContentParser.Destroy Destructor

Frees the memory associated with the TContentParser (see page 3301) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TContentParser();
```

Description

Do not free content parser instances. The Web request object frees the content parser when it no longer needs it.

See Also

System::TObject::Free

13.1.10.3 TContentParser Methods

The methods of the TContentParser class are listed here.

Public Methods

	Name	Description
	CanParse (see page 3303)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	GetContentFields (see page 3303)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 3304)	Returns a set of uploaded files that the parser extracts from the current Web request.

13.1.10.3.1 TContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a static class method, which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered content parser until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TContentParser (see page 3301), CanParse always returns true. TContentParser (see page 3301) is always the last content parser that the Web request object checks, and provides the default parsing when no other parser is available.

See Also

GetContentFields (see page 3303)

GetFiles (see page 3304)

13.1.10.3.2 TContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; override;
```

C++

```
virtual __fastcall TStrings GetContentFields();
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TContentParser (see page 3301), GetContentFields simply calls the ExtractContentFields method of the current Web request. This method assumes that fields are strings with the form "Name=Value", separated by ampersands (&).

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 3375)

ExtractContentFields (see page 3363)

13.1.10.3.3 TContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles();
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TContentParser (see page 3301), GetFiles always returns an empty list. That is, TContentParser (see page 3301) does not implement support for uploaded files in HTTP requests.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

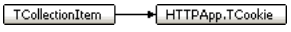
Files (see page 3379)

TAbstractWebRequestFile (see page 3288)

13.1.11 TCookie Class

TCookie represents a Cookie header in an HTTP response message.

Class Hierarchy



Pascal

```
TCookie = class(TCollectionItem);
```

C++

```
class TCookie : public TCollectionItem;
```

File

HTTPApp

Description

Use TCookie to specify the cookie header that should be sent with a response message when responding to an HTTP request

message. Cookie headers contain state information that the Web client passes on to other locations on the Internet or Intranet.

See Also

TCookieCollection (see page 3309)

TWebResponse (see page 3389)

13.1.11.1 TCookie Members

The following tables list the members exposed by TCookie.

Protected Methods

	Name	Description
	GetHeaderValue (see page 3306)	This is GetHeaderValue, a member of class TCookie.

Public Methods

	Name	Description
V	AssignTo (see page 3306)	Copies the properties of the TCookie (see page 3304) object to another TCookie (see page 3304) object.
V	Create (see page 3305)	Creates an instance of TCookie (see page 3304).

Public Properties

	Name	Description
	Domain (see page 3307)	Specifies the Domain field of the cookie header.
	Expires (see page 3307)	Specifies when the cookie is no longer valid.
R	HeaderValue (see page 3307)	Specifies the Cookie header string that represents this TCookie (see page 3304) object.
	Name (see page 3308)	Specifies the name of the custom field represented by this Cookie header.
	Path (see page 3308)	Identifies the paths of URLs to which the Cookie should be sent.
	Secure (see page 3309)	Determines whether the Web client must use a secure connection to pass on the Cookie header.
	Value (see page 3309)	Specifies the value of the Cookie header field named by the Name (see page 3308) property.

13.1.11.2 TCookie.Create Constructor

Creates an instance of TCookie (see page 3304).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCookie(TCollection Collection);
```

Description

It is seldom necessary to call the constructor for TCookie (see page 3304) directly. Instead, create TCookie (see page 3304) objects indirectly by calling the Add method of TCookieCollection (see page 3309).

After calling the inherited constructor, Create initializes the Expires (see page 3307) property to indicate no expiration date.

See Also

Add (see page 3311)

Expires (see page 3307)

13.1.11.3 TCookie Methods

The methods of the TCookie class are listed here.

Protected Methods

	Name	Description
	GetHeaderValue (see page 3306)	This is GetHeaderValue, a member of class TCookie.

Public Methods

	Name	Description
	AssignTo (see page 3306)	Copies the properties of the TCookie (see page 3304) object to another TCookie (see page 3304) object.

13.1.11.3.1 TCookie.AssignTo Method

Copies the properties of the TCookie (see page 3304) object to another TCookie (see page 3304) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Do not call the AssignTo method for TCookie (see page 3304). Instead, call the Assign method of the destination object. The Assign method of the destination object calls the AssignTo method if it does not contain code for copying the properties of a TCookie (see page 3304) object. If the destination object is a TCookie (see page 3304) object, AssignTo copies the Name (see page 3308), Value (see page 3309), Domain (see page 3307), Path (see page 3308), Expires (see page 3307), and Secure (see page 3309) properties so the destination cookie generates the same Cookie header string.

See Also

Assign

13.1.11.3.2 TCookie.GetHeaderValue Method

Pascal

```
function GetHeaderValue: AnsiString;
```

C++

```
__fastcall AnsiString GetHeaderValue();
```

Description

This is GetHeaderValue, a member of class TCookie.

13.1.11.4 TCookie Properties

The properties of the TCookie class are listed here.

Public Properties

	Name	Description
	Domain (see page 3307)	Specifies the Domain field of the cookie header.

	Expires (see page 3307)	Specifies when the cookie is no longer valid.
R	HeaderValue (see page 3307)	Specifies the Cookie header string that represents this TCookie (see page 3304) object.
	Name (see page 3308)	Specifies the name of the custom field represented by this Cookie header.
	Path (see page 3308)	Identifies the paths of URLs to which the Cookie should be sent.
	Secure (see page 3309)	Determines whether the Web client must use a secure connection to pass on the Cookie header.
	Value (see page 3309)	Specifies the value of the Cookie header field named by the Name (see page 3308) property.

13.1.11.4.1 TCookie.Domain Property

Specifies the Domain field of the cookie header.

Pascal

```
property Domain: AnsiString;
```

C++

```
__property AnsiString Domain;
```

Description

Use Domain to limit the recipients of the cookie header to certain domains. The Web client will only send this cookie on to URIs in the specified domain. To limit the recipients to certain paths within the specified domain, use the Path (see page 3308) property.

See Also

Path (see page 3308)

13.1.11.4.2 TCookie.Expires Property

Specifies when the cookie is no longer valid.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Set Expires to the date and time after which the Web client should not send the cookie on with HTTP messages.

See Also

Expires (see page 3409)

13.1.11.4.3 TCookie.HeaderValue Property

Specifies the Cookie header string that represents this TCookie (see page 3304) object.

Pascal

```
property HeaderValue: AnsiString;
```

C++

```
__property AnsiString HeaderValue;
```

Description

Read HeaderValue to determine the Cookie header that is added to the HTTP response message for this TCookie (see page 3304) object. HeaderValue combines the information specified by the Name (see page 3308) and Value (see page 3309) properties into a single Name (see page 3308)=Value (see page 3309) field, and then adds the Domain (see page 3307), Path (see page 3308), Expires (see page 3307), and Secure (see page 3309) fields based on the values of the properties with the corresponding names.

See Also

Name (see page 3308)

Value (see page 3309)

Domain (see page 3307)

Path (see page 3308)

Expires (see page 3307)

Secure (see page 3309)

13.1.11.4.4 TCookie.Name Property

Specifies the name of the custom field represented by this Cookie header.

Pascal

```
property Name: AnsiString;
```

C++

```
__property AnsiString Name;
```

Description

Set Name to the name that identifies the use of the Value (see page 3309) property in the Cookie header. The Value (see page 3309) property is combined with the Name property into a string of the form Name=Value (see page 3309). This string appears as one of the fields in the Cookie header represented by the HeaderValue (see page 3307) property.

See Also

HeaderValue (see page 3307)

Value (see page 3309)

13.1.11.4.5 TCookie.Path Property

Identifies the paths of URLs to which the Cookie should be sent.

Pascal

```
property Path: AnsiString;
```

C++

```
__property AnsiString Path;
```

Description

Use Path to limit the recipients of the cookie header to certain paths. The Web client will only send this cookie header on to URLs with the specified path. To limit the recipients to certain domains, use the Domain (see page 3307) property.

See Also

Domain (see page 3307)

PathInfo (see page 3382)

13.1.11.4.6 TCookie.Secure Property

Determines whether the Web client must use a secure connection to pass on the Cookie header.

Pascal

```
property Secure: Boolean;
```

C++

```
__property Boolean Secure;
```

Description

Set Secure to true if the Web client must use a secure connection when sending the Cookie header. Secure connections are connections where messages may not be stored on any permanent medium such as a back-up tape. If the message is stored temporarily in a file, the file must be deleted.

Set Secure to false if it does not matter whether the Cookie header is sent using a secure connection.

13.1.11.4.7 TCookie.Value Property

Specifies the value of the Cookie header field named by the Name (see page 3308) property.

Pascal

```
property Value: AnsiString;
```

C++

```
__property AnsiString Value;
```

Description

Set Value to the value associated with the field named by the Name (see page 3308) property. The Value property is combined with the Name (see page 3308) property into a string of the form Name (see page 3308)=Value. This string appears as one of the fields in the Cookie header represented by the HeaderValue (see page 3307) property.

See Also

HeaderValue (see page 3307)

Name (see page 3308)

13.1.12 TCookieCollection Class

TCookieCollection is a collection of TCookie (see page 3304) objects.

Class Hierarchy

Pascal

```
TCookieCollection = class(TCollection);
```

C++

```
class TCookieCollection : public TCollection;
```

File

HTTPApp

Description

Use TCookieCollection to represent all the Cookie headers of an HTTP response message. Each Cookie header is represented by a single TCookie (see page 3304) object. A Cookie header includes a named field with an associated value, of the form Name=Value, plus a number of other, standard fields that specify the destination of the cookie, when the cookie is valid, and so on. Web clients pass Cookie headers on to the destinations specified by the fields of the Cookie header.

See Also

TCookie (see page 3304)

TWebResponse (see page 3389)

13.1.12.1 TCookieCollection Members

The following tables list the members exposed by TCookieCollection.

Protected Methods

	Name	Description
	GetCookie (see page 3311)	This is GetCookie, a member of class TCookieCollection.
	SetCookie (see page 3311)	This is SetCookie, a member of class TCookieCollection.

Public Methods

	Name	Description
	Add (see page 3311)	Creates a new TCookie (see page 3304) object and adds it to the end of the Items (see page 3312) property array.
	Create (see page 3310)	Creates and initializes a TCookieCollection (see page 3309) object.

Public Properties

	Name	Description
	Items (see page 3312)	Provides indexed access to the TCookie (see page 3304) objects in the collection.
R	WebResponse (see page 3312)	Indicates the TWebResponse (see page 3389) object that uses the cookie collection to represent all of its Cookie headers.

13.1.12.2 TCookieCollection.Create Constructor

Creates and initializes a TCookieCollection (see page 3309) object.

Pascal

```
constructor Create(WebResponse: TWebResponse; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TCookieCollection(TWebResponse WebResponse, TCollectionItemClass ItemClass);
```

Description

Applications should not need to call the Create method directly. The Web response object that uses the cookie collection to represent its Cookie headers creates the TCookieCollection (see page 3309) object from its own constructor.

13.1.12.3 TCookieCollection Methods

The methods of the TCookieCollection class are listed here.

Protected Methods

	Name	Description
	GetCookie (see page 3311)	This is GetCookie, a member of class TCookieCollection.
	SetCookie (see page 3311)	This is SetCookie, a member of class TCookieCollection.

Public Methods

	Name	Description
	Add (see page 3311)	Creates a new TCookie (see page 3304) object and adds it to the end of the Items (see page 3312) property array.

13.1.12.3.1 TCookieCollection.Add Method

Creates a new TCookie (see page 3304) object and adds it to the end of the Items (see page 3312) property array.

Pascal

```
function Add: TCookie;
```

C++

```
__fastcall TCookie Add();
```

Description

Call Add to add a new Cookie header to the set of cookie headers sent with the Web response message. Use Add when adding a Cookie header whose fields will be set at a later time. If the values of the TCookie (see page 3304) properties are known at the time of creation, use the SetCookieField method of the Web response object instead.

See Also

WebResponse (see page 3312)

SetCookieField (see page 3400)

Items (see page 3312)

13.1.12.3.2 TCookieCollection.GetCookie Method

Pascal

```
function GetCookie(Index: Integer): TCookie;
```

C++

```
__fastcall TCookie GetCookie(int Index);
```

Description

This is GetCookie, a member of class TCookieCollection.

13.1.12.3.3 TCookieCollection.SetCookie Method

Pascal

```
procedure SetCookie(Index: Integer; Cookie: TCookie);
```

C++

```
__fastcall SetCookie(int Index, TCookie Cookie);
```

Description

This is SetCookie, a member of class TCookieCollection.

13.1.12.4 TCookieCollection Properties

The properties of the TCookieCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 3312)	Provides indexed access to the TCookie (see page 3304) objects in the collection.
R	WebResponse (see page 3312)	Indicates the TWebResponse (see page 3389) object that uses the cookie collection to represent all of its Cookie headers.

13.1.12.4.1 TCookieCollection.Items Property

Provides indexed access to the TCookie (see page 3304) objects in the collection.

Pascal

```
property Items [Index: Integer]: TCookie;
```

C++

```
__property TCookie Items[int Index];
```

Description

Use Items to access the properties of the TCookie (see page 3304) objects maintained by TCookieCollection (see page 3309).

Note: In Delphi, items is the default property of TCookieCollection (see page 3309), so the name of the Items property can be left off when using this property. Thus the line

Note: FirstCookie := TApacheResponse1.Cookies.Items[0];

Note: can be written

Note: FirstCookie := TApacheResponse1.Cookies[0];

See Also

Cookies (see page 3407)

operator_sb

13.1.12.4.2 TCookieCollection.WebResponse Property

Indicates the TWebResponse (see page 3389) object that uses the cookie collection to represent all of its Cookie headers.

Pascal

```
property WebResponse: TWebResponse;
```

C++

```
__property TWebResponse WebResponse;
```

Description

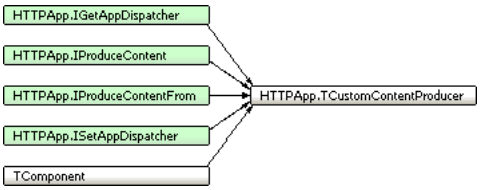
Read WebResponse to access the response object that is using the cookie collection. The response object is a descendant of TWebResponse (see page 3389), either TISAPIResponse, TWinCGIResponse,TCGIResponse, or TApacheResponse, depending on the type of Web server application.

Use the properties of the response object to obtain information about the other headers in the response message or the content of the response message. Many applications use this information to determine what field values to specify as Cookie headers.

13.1.13 TCustomContentProducer Class

TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.

Class Hierarchy



Pascal

```
TCustomContentProducer = class(TComponent, ISetAppDispatcher, IGetAppDispatcher,
IProduceContent, IProduceContentFrom);
```

C++

```
class TCustomContentProducer : public TComponent, public ISetAppDispatcher, public
IGetAppDispatcher, public IProduceContent, public IProduceContentFrom;
```

File

HTTPApp

Description

Use TCustomContentProducer as a base class when defining a class to produce the contents of an HTTP response message. In most cases, the content producer will assemble a string of HTML commands. However, descendants of TCustomContentProducer can be defined to produce any valid MIME type.

See Also

TDSTableProducer

TPageProducer (see page 3496)

TWebResponse (see page 3389)

Generating the Content of Response Messages

The Web Module

13.1.13.1 TCustomContentProducer Members

The following tables list the members exposed by TCustomContentProducer.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 3316)	This is GetAppDispatcher, a member of class TCustomContentProducer.
V	Notification (see page 3316)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 3316)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3317)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 3317)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	SetAppDispatcher (see page 3317)	This is SetAppDispatcher, a member of class TCustomContentProducer.

Public Methods

	Name	Description
V	Content (see page 3314)	Returns the content produced by the content producer.
V	ContentFromStream (see page 3315)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 3315)	Returns the content produced in response to an input string.

Public Properties

	Name	Description
R	Dispatcher (see page 3317)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.
R	DispatcherComponent (see page 3318)	Provides access to the dispatcher component.

13.1.13.2 TCustomContentProducer Methods

The methods of the TCustomContentProducer class are listed here.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 3316)	This is GetAppDispatcher, a member of class TCustomContentProducer.
V	Notification (see page 3316)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 3316)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3317)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 3317)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	SetAppDispatcher (see page 3317)	This is SetAppDispatcher, a member of class TCustomContentProducer.

Public Methods

	Name	Description
V	Content (see page 3314)	Returns the content produced by the content producer.
V	ContentFromStream (see page 3315)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 3315)	Returns the content produced in response to an input string.

13.1.13.2.1 TCustomContentProducer.Content Method

Returns the content produced by the content producer.

Pascal

```
function Content: string; virtual;
```

C++

```
virtual __fastcall UnicodeString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. TCustomContentProducer (see page 3313) returns an empty string. Descendants of TCustomContentProducer (see page 3313) return the value of the content that they produce. In most cases, this content is a string of HTML commands, although it can be any valid MIME type.

See Also

ContentFromStream (see page 3315)

ContentFromString (see page 3315)

Content (see page 3404)

Responding to Request Messages with Action Items

13.1.13.2.2 TCustomContentProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(Stream: TStream): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream Stream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter. TCustomContentProducer (see page 3313) ignores the Stream parameter and returns the value created by the Content (see page 3314) method.

Descendants of TCustomContentProducer (see page 3313) override this method to interpret streamed input in their own way.

See Also

Content (see page 3314)

ContentFromString (see page 3315)

Content (see page 3404)

13.1.13.2.3 TCustomContentProducer.ContentFromString Method

Returns the content produced in response to an input string.

Pascal

```
function ContentFromString(const S: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromString(const UnicodeString S);
```

Description

Call ContentFromString to retrieve the content produced by the content producer based on the information provided by the S

parameter. TCustomContentProducer (see page 3313) ignores the string and returns the value created by the Content (see page 3314) method.

Descendants of TCustomContentProducer (see page 3313) override this method to interpret the input string in their own way.

See Also

Content (see page 3314)

ContentFromStream (see page 3315)

Content (see page 3404)

13.1.13.2.4 TCustomContentProducer.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class TCustomContentProducer.

13.1.13.2.5 TCustomContentProducer.Notification Method

Overrides the inherited method to respond to messages that the dispatcher is going away.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether this call was triggered by the removal of the dispatcher. If so, the Dispatcher (see page 3317) property is set to nil. NULL.

See Also

DispatcherComponent (see page 3318)

Dispatcher (see page 3317)

13.1.13.2.6 TCustomContentProducer.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall UnicodeString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content (see page 3314)

13.1.13.2.7 TCustomContentProducer.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class TCustomContentProducer.

13.1.13.2.8 TCustomContentProducer.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromString(const UnicodeString S);
```

Description

This is ProduceContentFromString, a member of class TCustomContentProducer.

13.1.13.2.9 TCustomContentProducer.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class TCustomContentProducer.

13.1.13.3 TCustomContentProducer Properties

The properties of the TCustomContentProducer class are listed here.

Public Properties

	Name	Description
	Dispatcher (see page 3317)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.
	DispatcherComponent (see page 3318)	Provides access to the dispatcher component.

13.1.13.3.1 TCustomContentProducer.Dispatcher Property

Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.

Pascal

```
property Dispatcher: IWebDispatcherAccess;
```

C++

```
__property IWebDispatcherAccess Dispatcher;
```

Description

Read Dispatcher to access the interface of the dispatcher. In Web Broker applications, this is the interface of a TWebDispatcher (see page 3349) or TWebModule (see page 3354) component. In WebSnap applications, this is the interface of a TWebAppComponents component.

The IWebDispatcherAccess (see page 3431) interface has two methods:

Request, which returns the HTTP request message to which this component responds.

Response, which returns the HTTP response message for which this component generates content.

See Also

TWebModule (see page 3354)

TWebDispatcher (see page 3349)

TWebAppComponents

TWebRequest (see page 3358)

TWebResponse (see page 3389)

DispatcherComponent (see page 3318)

13.1.13.3.2 TCustomContentProducer.DispatcherComponent Property

Provides access to the dispatcher component.

Pascal

```
property DispatcherComponent: TComponent;
```

C++

```
__property TComponent * DispatcherComponent;
```

Description

Dispatcher (see page 3317) component is the component that implements the interface that is the value of the Dispatcher (see page 3317) property. The content producer uses this property so that it can respond to notifications when the component is freed. To use the dispatcher, use the Dispatcher (see page 3317) property instead.

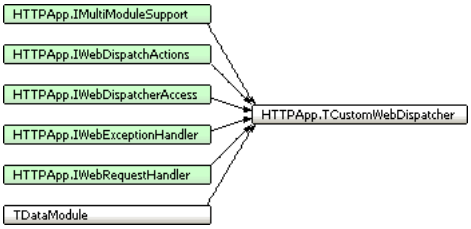
See Also

Dispatcher (see page 3317)

13.1.14 TCustomWebDispatcher Class

TCustomWebDispatcher is the abstract base class for Web dispatcher objects.

Class Hierarchy



Pascal

```
TCustomWebDispatcher = class(TDataModule, IWebRequestHandler, IWebDispatchActions,
IWebDispatcherAccess, IMultiModuleSupport, IWebExceptionHandler);
```

C++

```
class TCustomWebDispatcher : public TDataModule, public IWebRequestHandler, public
IWebDispatchActions, public IWebDispatcherAccess, public IMultiModuleSupport, public
IWebExceptionHandler;
```

File

HTTPApp

Description

Add a descendant of TCustomWebDispatcher to a Web application to enable it to respond to HTTP request messages. When a Web application is created, it automatically includes a TWebModule (see page 3354) object, which is a descendant of TCustomWebDispatcher. Keep the default Web module, or replace it with another data module that contains a descendant of TCustomWebDispatcher, such as TWebDispatcher (see page 3349).

TCustomWebDispatcher manages the action items that assemble responses to HTTP request messages. When the TWebApplication object receives an HTTP request message, TCustomWebDispatcher passes the request on to the appropriate action item or action items to service the message.

TCustomWebDispatcher also keeps track of information about the current HTTP message for the benefit of TCustomContentProducer (see page 3313) objects.

See Also

TCustomContentProducer (see page 3313)

TWebActionItem (see page 3335)

TWebActionItems (see page 3344)

TWebApplication

TWebDispatcher (see page 3349)

TWebModule (see page 3354)

TWebRequest (see page 3358)

TWebResponse (see page 3389)

The Web Module

13.1.14.1 TCustomWebDispatcher Members

The following tables list the members exposed by TCustomWebDispatcher.

Protected Events

	Name	Description
	AfterDispatch (see page 3329)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3329)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Protected Methods

	Name	Description
	Access_Request (see page 3322)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 3322)	This is Access_Response, a member of class TCustomWebDispatcher.
	DispatchAction (see page 3323)	Passes an HTTP request message to the action items in Actions (see page 3327) until it is handled.
	DoAfterDispatch (see page 3324)	Generates an AfterDispatch (see page 3329) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 3324)	Generates a BeforeDispatch (see page 3329) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
	FinishContext (see page 3325)	This is FinishContext, a member of class TCustomWebDispatcher.
	HandleException (see page 3325)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 3325)	Processes an HTTP request message.
	InitContext (see page 3326)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 3326)	This is InitModule, a member of class TCustomWebDispatcher.
V	Notification (see page 3326)	Responds to notifications when objects are added to the module.

Public Events

	Name	Description
	OnException (see page 3330)	Occurs when an unhandled exception occurs in the application.

Public Methods

	Name	Description
	ActionByName (see page 3322)	Returns the action item identified by the AName parameter.
V	Create (see page 3320)	Creates an instance of TCustomWebDispatcher (see page 3318).
V	Destroy (see page 3321)	Frees an instance of TCustomWebDispatcher (see page 3318).

Public Properties

	Name	Description
R	Action (see page 3327)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 3318) object.
	Actions (see page 3327)	Represents the collection of action items managed by the TCustomWebDispatcher (see page 3318) object.
R	Request (see page 3327)	Provides access to the TWebRequest (see page 3358) object that the dispatcher is currently handling.
R	Response (see page 3328)	Provides access to the TWebResponse (see page 3389) object for the message the dispatcher is currently handling.

13.1.14.2 TCustomWebDispatcher.Create Constructor

Creates an instance of TCustomWebDispatcher (see page 3318).

Pascal

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

C++

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

Description

Do not call the constructor for TCustomWebDispatcher (see page 3318). Web applications automatically include a Web module. If the Web module is replaced by another data model, the Web dispatcher should be added to the new data module at design time. Objects placed at design time are created automatically. If an application tries to create a dispatcher object in a Web module, or in a data module that already has a Web dispatcher, an exception is raised.

Create creates the TWebActionItems (see page 3344) object that holds the action items managed by the dispatcher. If the Web dispatcher is included in another data module, it adds a reference to itself to any content producer in that data module.

After the Web dispatcher is created automatically at runtime, an OnCreate event occurs.

See Also

Actions (see page 3327)

OnCreate

Dispatcher (see page 3317)

13.1.14.3 TCustomWebDispatcher.Destroy Destructor

Frees an instance of TCustomWebDispatcher (see page 3318).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebDispatcher();
```

Description

Do not destroy the TCustomWebDispatcher (see page 3318) directly in an application. The dispatcher is freed automatically when the server application shuts down.

Destroy frees the TWebActionItems (see page 3344) object used to hold the collection of action items.

See Also

Actions (see page 3327)

System::TObject::Free

13.1.14.4 TCustomWebDispatcher Methods

The methods of the TCustomWebDispatcher class are listed here.

Protected Methods

	Name	Description
	Access_Request (see page 3322)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 3322)	This is Access_Response, a member of class TCustomWebDispatcher.
	DispatchAction (see page 3323)	Passes an HTTP request message to the action items in Actions (see page 3327) until it is handled.
	DoAfterDispatch (see page 3324)	Generates an AfterDispatch (see page 3329) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 3324)	Generates a BeforeDispatch (see page 3329) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
	FinishContext (see page 3325)	This is FinishContext, a member of class TCustomWebDispatcher.

	HandleException (see page 3325)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 3325)	Processes an HTTP request message.
	InitContext (see page 3326)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 3326)	This is InitModule, a member of class TCustomWebDispatcher.
V	Notification (see page 3326)	Responds to notifications when objects are added to the module.

Public Methods

	Name	Description
	ActionByName (see page 3322)	Returns the action item identified by the AName parameter.

13.1.14.4.1 TCustomWebDispatcher.Access_Request Method

Pascal

```
function Access_Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Access_Request();
```

Description

This is Access_Request, a member of class TCustomWebDispatcher.

13.1.14.4.2 TCustomWebDispatcher.Access_Response Method

Pascal

```
function Access_Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Access_Response();
```

Description

This is Access_Response, a member of class TCustomWebDispatcher.

13.1.14.4.3 TCustomWebDispatcher.ActionByName Method

Returns the action item identified by the AName parameter.

Pascal

```
function ActionByName(const AName: string): TWebActionItem;
```

C++

```
__fastcall TWebActionItem ActionByName(const UnicodeString AName);
```

Description

Call ActionByName to gain access to an individual action item when its name is known. Change the properties of the action item so that it handles different HTTP request messages, or invoke its OnAction event handler to respond to an HTTP request message.

See Also

Action (see page 3327)

Actions (see page 3327)

TWebActionItem (see page 3335)

Name (see page 3341)

13.1.14.4.4 TCustomWebDispatcher.DispatchAction Method

Passes an HTTP request message to the action items in Actions (see page 3327) until it is handled.

Pascal

```
function DispatchAction(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchAction(TWebRequest Request, TWebResponse Response);
```

Description

The Web application calls DispatchAction when it receives an HTTP request message so that the dispatcher can dispatch the request to any action items that can generate the response.

Before DispatchAction looks for action items to handle the HTTP request message represented by the Request (see page 3327) parameter, it generates a BeforeDispatch (see page 3329) event. If the BeforeDispatch (see page 3329) event handler handles the action, the dispatcher does not pass the request message to any of the action items.

If there is no BeforeDispatch (see page 3329) event handler, or of the BeforeDispatch (see page 3329) event handler does not handle the request, DispatchAction attempts to match the MethodType and PathInfo of the request with the MethodType and PathInfo properties of each action item in the Actions (see page 3327) property. When an action item matches the MethodType and PathInfo of the request, DispatchAction calls its OnAction event handler.

The OnAction event handler can complete the response to the request, or indicate that the request is not fully handled and allow another action item to finish processing the request. If the OnAction event handler does not complete the request, DispatchAction continues checking the action items in the Actions (see page 3327) property and calling the OnAction event handler when a match is found.

If all of the action items are checked, and the request is still not fully handled, DispatchAction calls the OnAction event handler of the default action item.

Finally, if the request has been handled by the BeforeDispatch (see page 3329) event handler or any of the action items, DispatchAction generates an AfterDispatch (see page 3329) event. The AfterDispatch (see page 3329) event does not occur if the BeforeDispatch (see page 3329) event or any of the action items has already sent the response.

DispatchAction returns a Boolean value indicating whether the Web application should send the response message if it is not already sent. DispatchAction returns true if the action was handled by the action items and the AfterDispatch (see page 3329) event handler did not change its Handled parameter to false.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

BeforeDispatch (see page 3329)

TWebActionItem (see page 3335)

OnAction (see page 3343)

HandleRequest

TWebRequest (see page 3358)

TWebResponse (see page 3389)

13.1.14.4.5 TCustomWebDispatcher.DoAfterDispatch Method

Generates an AfterDispatch (see page 3329) event and returns a value that indicates whether the Web application can send the response.

Pascal

```
function DoAfterDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoAfterDispatch(TWebRequest Request, TWebResponse Response);
```

Description

Applications cannot call DoAfterDispatch. The DispatchAction (see page 3323) method calls DoAfterDispatch after sending the HTTP request message to any matching action items. The DispatchAction (see page 3323) method does not call DoAfterDispatch if the HTTP request message was not handled by the action items or by a BeforeDispatch (see page 3329) event handler. DoAfterDispatch is also not called if the message was already sent back to the Web client.

If the AfterDispatch (see page 3329) event handler indicates that the response should not be sent, DoAfterDispatch returns false, and the Web application drops the connection without sending the response message. Otherwise, DoAfterDispatch returns true, and the Web application sends the response message if it was not sent from within the AfterDispatch (see page 3329) event handler.

See Also

AfterDispatch (see page 3329)

BeforeDispatch (see page 3329)

DispatchAction (see page 3323)

13.1.14.4.6 TCustomWebDispatcher.DoBeforeDispatch Method

Generates a BeforeDispatch (see page 3329) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.

Pascal

```
function DoBeforeDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoBeforeDispatch(TWebRequest Request, TWebResponse Response);
```

Description

Applications cannot call DoBeforeDispatch. The DispatchAction (see page 3323) method calls DoBeforeDispatch before trying to match the HTTP request message with any of its action items.

If the BeforeDispatch (see page 3329) event handler indicates that it has handled the request message, DoBeforeDispatch returns true, and the DispatchAction (see page 3323) method does not try to pass the request message to any of the action items. Otherwise, DoBeforeDispatch returns false, and unless the response message was sent from within the BeforeDispatch (see page 3329) event handler, the DispatchAction (see page 3323) method proceeds with looking for action items to handle the request message.

If DoBeforeDispatch returns true but the response has not been sent, DispatchAction (see page 3323) generates an AfterDispatch (see page 3329) event.

See Also

AfterDispatch (see page 3329)

BeforeDispatch (see page 3329)

DispatchAction (see page 3323)

13.1.14.4.7 TCustomWebDispatcher.FinishContext Method

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

This is FinishContext, a member of class TCustomWebDispatcher.

13.1.14.4.8 TCustomWebDispatcher.HandleException Method

Provides default handling of exceptions for the web application.

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

If an exception passes through all the try blocks in the web application code, the application automatically calls HandleException, which displays a dialog box indicating an error occurred. Unless the exception object is EAbort, HandleException calls the OnException (see page 3330) event handler.

To assign other exception handling code for the web application, use the OnException (see page 3330) event handler.

See Also

OnException (see page 3330)

13.1.14.4.9 TCustomWebDispatcher.HandleRequest Method

Processes an HTTP request message.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

Call HandleRequest to process an HTTP request message and obtain the response that should be sent back to the caller.

Request is the request object that describes the request to process.

Response is the response object that should be filled out when the request is processed.

13.1.14.4.10 TCustomWebDispatcher.InitContext Method

Pascal

```
procedure InitContext(Request: TWebRequest; Response: TWebResponse);
```

C++

```
__fastcall InitContext(TWebRequest Request, TWebResponse Response);
```

Description

This is InitContext, a member of class TCustomWebDispatcher.

13.1.14.4.11 TCustomWebDispatcher.InitModule Method

Pascal

```
procedure InitModule(AModule: TComponent);
```

C++

```
__fastcall InitModule(TComponent * AModule);
```

Description

This is InitModule, a member of class TCustomWebDispatcher.

13.1.14.4.12 TCustomWebDispatcher.Notification Method

Responds to notifications when objects are added to the module.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification ensures that all TCustomContentProducer (see page 3313) objects have access to the dispatcher, by setting their Dispatcher property as they are added to the application.

See Also

Dispatcher (see page 3317)

13.1.14.5 TCustomWebDispatcher Properties

The properties of the TCustomWebDispatcher class are listed here.

Public Properties

	Name	Description
R	Action (see page 3327)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 3318) object.
	Actions (see page 3327)	Represents the collection of action items managed by the TCustomWebDispatcher (see page 3318) object.
R	Request (see page 3327)	Provides access to the TWebRequest (see page 3358) object that the dispatcher is currently handling.

	Response (see page 3328)	Provides access to the TWebResponse (see page 3389) object for the message the dispatcher is currently handling.

13.1.14.5.1 TCustomWebDispatcher.Action Property

Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 3318) object.

Pascal

```
property Action [Index: Integer]: TWebActionItem;
```

C++

```
__property TWebActionItem Action[int Index];
```

Description

Use Action to iterate over all of the actions in the Actions (see page 3327) collection. Zero is the index of the first action item, one is the index of the second action item, and so on. To access individual actions by name, use the ActionByName (see page 3322) method.

See Also

Actions (see page 3327)

TWebActionItem (see page 3335)

13.1.14.5.2 TCustomWebDispatcher.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher (see page 3318) object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 3327)

ActionByName (see page 3322)

TWebActionItems (see page 3344)

13.1.14.5.3 TCustomWebDispatcher.Request Property

Provides access to the TWebRequest (see page 3358) object that the dispatcher is currently handling.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Use Request to obtain information about the current HTTP request message. The dispatcher matches this TWebRequest (see page 3358) object to an action item, and invokes the OnAction event handler of the action item to generate and send a response to the HTTP request message. OnAction event handlers often use descendants of TCustomContentProducer (see page 3313) to help generate the response to the request message. These content producers can use the Request property to correctly produce the content of the HTTP response.

See Also

DispatchAction (see page 3323)

Response (see page 3328)

TCustomContentProducer (see page 3313)

OnAction (see page 3343)

13.1.14.5.4 TCustomWebDispatcher.Response Property

Provides access to the TWebResponse (see page 3389) object for the message the dispatcher is currently handling.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Use Response to obtain information about the emerging response to the current HTTP request message. The dispatcher matches the request object specified by the Request (see page 3327) property to an action item. It then invokes the OnAction event handler of the action item to fill in the TWebResponse (see page 3389) object specified by the Response property. OnAction event handlers often use descendants of TCustomContentProducer (see page 3313) to help generate the response. These content producers can use Response to determine the state of the emerging response that is being produced.

See Also

DispatchAction (see page 3323)

Request (see page 3327)

TCustomContentProducer (see page 3313)

OnAction (see page 3343)

13.1.14.6 TCustomWebDispatcher Events

The events of the TCustomWebDispatcher class are listed here.

Protected Events

	Name	Description
	AfterDispatch (see page 3329)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3329)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Public Events

	Name	Description
	OnException (see page 3330)	Occurs when an unhandled exception occurs in the application.

13.1.14.6.1 TCustomWebDispatcher.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 3327)

BeforeDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

13.1.14.6.2 TCustomWebDispatcher.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch (see page 3329) event.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

Enabling and Disabling Action Items

13.1.14.6.3 TCustomWebDispatcher.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException (see page 3325) method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException (see page 3325) method. Unless the exception object is EAbort, HandleException (see page 3325) calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

TWebResponse (see page 3389)

13.1.15 TDefaultWebAppServices Class

Class Hierarchy

Pascal

```
TDefaultWebAppServices = class(TInterfacedObject, IWebAppServices);
```

C++

```
class TDefaultWebAppServices : public TInterfacedObject, public IWebAppServices;
```

File

HTTPApp

Description

This is class HTTPApp.TDefaultWebAppServices.

13.1.15.1 TDefaultWebAppServices Members

The following tables list the members exposed by TDefaultWebAppServices.

Protected Methods

	Name	Description
V	FindRequestHandler (see page 3331)	This is FindRequestHandler, a member of class TDefaultWebAppServices.
	FinishContext (see page 3332)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 3332)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetRequestHandler (see page 3332)	This is GetRequestHandler, a member of class TDefaultWebAppServices.
	HandleRequest (see page 3332)	Directs the processing of the current request message.
	InitContext (see page 3333)	Creates and initializes the Web context for the current Web request.
	InvokeDispatcher (see page 3334)	This is InvokeDispatcher, a member of class TDefaultWebAppServices.

Protected Properties

	Name	Description
R	Request (see page 3334)	This is Request, a member of class TDefaultWebAppServices.
R	RequestHandler (see page 3334)	This is RequestHandler, a member of class TDefaultWebAppServices.
R	Response (see page 3334)	This is Response, a member of class TDefaultWebAppServices.
R	WebModules (see page 3335)	This is WebModules, a member of class TDefaultWebAppServices.

Public Properties

	Name	Description
R	ExceptionHandler (see page 3427)	Indicates the object that handles exceptions that arise when processing a request message.

13.1.15.2 TDefaultWebAppServices Methods

The methods of the TDefaultWebAppServices class are listed here.

Protected Methods

	Name	Description
V	FindRequestHandler (see page 3331)	This is FindRequestHandler, a member of class TDefaultWebAppServices.
	FinishContext (see page 3332)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 3332)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetRequestHandler (see page 3332)	This is GetRequestHandler, a member of class TDefaultWebAppServices.
	HandleRequest (see page 3332)	Directs the processing of the current request message.
	InitContext (see page 3333)	Creates and initializes the Web context for the current Web request.
	InvokeDispatcher (see page 3334)	This is InvokeDispatcher, a member of class TDefaultWebAppServices.

13.1.15.2.1 TDefaultWebAppServices.FindRequestHandler Method

Pascal

```
function FindRequestHandler: IWebRequestHandler; virtual;
```

C++

```
virtual __fastcall IWebRequestHandler FindRequestHandler();
```

Description

This is FindRequestHandler, a member of class TDefaultWebAppServices.

13.1.15.2.2 TDefaultWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message (including the Web context).

See Also

InitContext (see page 3426)

WebContext

13.1.15.2.3 TDefaultWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler is the read implementation of the ExceptionHandler property.

See Also

ExceptionHandler (see page 3427)

13.1.15.2.4 TDefaultWebAppServices.GetRequestHandler Method

Pascal

```
function GetRequestHandler: IWebRequestHandler;
```

C++

```
__fastcall IWebRequestHandler GetRequestHandler();
```

Description

This is GetRequestHandler, a member of class TDefaultWebAppServices.

13.1.15.2.5 TDefaultWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext is called to perform any final cleanup.

HandleRequest locates the dispatcher components that handle the request and forwards the request to a dispatcher for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

InitContext (see page 3426)

FinishContext (see page 3425)

13.1.15.2.6 TDefaultWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse
Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message.

Request describes the new request message.

Response is the response object that the WebSnap application should fill out when handling the current request.

See Also

FinishContext (see page 3425)

WebContext

13.1.15.2.7 TDefaultWebAppServices.InvokeDispatcher Method

Pascal

```
function InvokeDispatcher: Boolean;
```

C++

```
__fastcall Boolean InvokeDispatcher();
```

Description

This is InvokeDispatcher, a member of class TDefaultWebAppServices.

13.1.15.3 TDefaultWebAppServices Properties

The properties of the TDefaultWebAppServices class are listed here.

Protected Properties

	Name	Description
	Request (see page 3334)	This is Request, a member of class TDefaultWebAppServices.
	RequestHandler (see page 3334)	This is RequestHandler, a member of class TDefaultWebAppServices.
	Response (see page 3334)	This is Response, a member of class TDefaultWebAppServices.
	WebModules (see page 3335)	This is WebModules, a member of class TDefaultWebAppServices.

13.1.15.3.1 TDefaultWebAppServices.Request Property

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

This is Request, a member of class TDefaultWebAppServices.

13.1.15.3.2 TDefaultWebAppServices.RequestHandler Property

Pascal

```
property RequestHandler: IWebRequestHandler;
```

C++

```
__property IWebRequestHandler RequestHandler;
```

Description

This is RequestHandler, a member of class TDefaultWebAppServices.

13.1.15.3.3 TDefaultWebAppServices.Response Property

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

This is Response, a member of class TDefaultWebAppServices.

13.1.15.3.4 TDefaultWebAppServices.WebModules Property

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

This is WebModules, a member of class TDefaultWebAppServices.

13.1.16 TWebActionItem Class

TWebActionItem allows a Web server application to create and send responses to HTTP request messages.

Class Hierarchy

Pascal

```
TWebActionItem = class(TCollectionItem);
```

C++

```
class TWebActionItem : public TCollectionItem;
```

File

HTTPApp

Description

Use TWebActionItem to assemble the response to a particular type of HTTP request message in a Web server application. Each action item is specific to a particular URI path and HTTP request method.

The web dispatcher chooses an appropriate action item to handle an HTTP request message by

matching the path portion of the URI requested by the Web client with the PathInfo (see page 3341) property in the TWebActionItem object

matching the method requested by the Web client with the method type the action item knows how to handle.

An action item handles an HTTP request message automatically using an associated content producer or using its OnAction (see page 3343) event handler. The OnAction (see page 3343) event handler can read the request message and assemble a response by setting the properties of a TWebResponse (see page 3389) object. It can tell the TWebResponse (see page 3389) object to send the response message back to the Web client, or allow the TWebApplication object to send the message after other action items have had a chance to add to the response.

See Also

TWebActionItems (see page 3344)

TWebDispatcher (see page 3349)

TWebRequest (see page 3358)

TWebResponse (see page 3389)

Action Items

13.1.16.1 TWebActionItem Members

The following tables list the members exposed by TWebActionItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 3338)	Returns the name of the collection item as it appears in the collection editor.
	GetPathInfo (see page 3338)	This is GetPathInfo, a member of class TWebActionItem.
	SetDisplayName (see page 3338)	Sets the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	AssignTo (see page 3337)	Copies the properties of the action item to another action item.
	Create (see page 3336)	Creates an instance of TWebActionItem (see page 3335).
	Destroy (see page 3337)	Destroys an instance of TWebActionItem (see page 3335).

Published Events

	Name	Description
	OnAction (see page 3343)	Occurs when TWebDispatcher (see page 3349) matches the action item to a newly-arrived HTTP request message.

Published Properties

	Name	Description
	Default (see page 3339)	Indicates whether the action item should handle any request that is not handled by another available action item.
	Enabled (see page 3340)	Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 3341) and PathInfo (see page 3341) properties.
	MethodType (see page 3341)	This is MethodType, a member of class TWebActionItem.
	Name (see page 3341)	Identifies an individual action item when it is referenced from other objects.
	PathInfo (see page 3341)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.
	Producer (see page 3342)	Specifies a content producer that generates the content of response messages when the action item executes.
	ProducerContent (see page 3342)	Specifies a content producer that generates the content of response messages when the action item executes.

13.1.16.2 TWebActionItem.Create Constructor

Creates an instance of TWebActionItem (see page 3335).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWebActionItem(TCollection Collection);
```

Description

Call Create to create a TWebActionItem (see page 3335) object at runtime. Action items created in the action editor at design time are instantiated automatically.

After calling the inherited constructor, Create initializes the Enabled (see page 3340) property to True, and sets the action item so that it will match any HTTP request targeted to a URI with an empty PathInfo (see page 3341).

See Also

Enabled (see page 3340)

PathInfo (see page 3341)

Create

13.1.16.3 TWebActionItem.Destroy Destructor

Destroys an instance of TWebActionItem (see page 3335).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebActionItem();
```

Description

Do not call Destroy directly in an application. Action items are destroyed when the TWebActionItems (see page 3344) object that manages them calls the Free method.

See Also

System::TObject::Free

TWebActionItems (see page 3344)

13.1.16.4 TWebActionItem Methods

The methods of the TWebActionItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 3338)	Returns the name of the collection item as it appears in the collection editor.
	GetPathInfo (see page 3338)	This is GetPathInfo, a member of class TWebActionItem.
	SetDisplayName (see page 3338)	Sets the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	AssignTo (see page 3337)	Copies the properties of the action item to another action item.

13.1.16.4.1 TWebActionItem.AssignTo Method

Copies the properties of the action item to another action item.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

To assign the value of one action item to another, use the Assign method of the target action item.

AssignTo enables the Assign method of a target action item to copy the properties of a source action item. AssignTo copies the values of the Default (see page 3339), Enabled (see page 3340), MethodType (see page 3341) and PathInfo (see page 3341) properties to the Dest parameter if it is an action item. If Dest is not an action item, AssignTo calls the inherited method.

See Also

Assign

Default (see page 3339)

Enabled (see page 3340)

MethodType (see page 3341)

PathInfo (see page 3341)

AssignTo

13.1.16.4.2 TWebActionItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

13.1.16.4.3 TWebActionItem.GetPathInfo Method

Pascal

```
function GetPathInfo: string;
```

C++

```
__fastcall UnicodeString GetPathInfo();
```

Description

This is GetPathInfo, a member of class TWebActionItem.

13.1.16.4.4 TWebActionItem.SetDisplayName Method

Sets the name of the collection item as it appears in the collection editor.

Pascal

```
procedure SetDisplayName(const Value: string); override;
```

C++

```
virtual __fastcall SetDisplayName(const UnicodeString Value);
```

Description

SetDisplayName is the protected write implementation of the DisplayName property.

Value specifies the name for the item that should appear in the collection editor.

See Also

DisplayName

13.1.16.5 TWebActionItem Properties

The properties of the TWebActionItem class are listed here.

Published Properties

	Name	Description
	Default (see page 3339)	Indicates whether the action item should handle any request that is not handled by another available action item.
	Enabled (see page 3340)	Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 3341) and PathInfo (see page 3341) properties.
	MethodType (see page 3341)	This is MethodType, a member of class TWebActionItem.
	Name (see page 3341)	Identifies an individual action item when it is referenced from other objects.
	PathInfo (see page 3341)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.
	Producer (see page 3342)	Specifies a content producer that generates the content of response messages when the action item executes.
	ProducerContent (see page 3342)	Specifies a content producer that generates the content of response messages when the action item executes.

13.1.16.5.1 TWebActionItem.Default Property

Indicates whether the action item should handle any request that is not handled by another available action item.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

Set Default to true to make this action item the default action in the list of action items maintained by the TWebDispatcher (see page 3349) object. TWebDispatcher (see page 3349) attempts to match HTTP request messages to action items by matching the MethodType (see page 3341) and PathInfo (see page 3341) of the target URI. If no match can be found, the TWebDispatcher (see page 3349) invokes the default action item, regardless of its MethodType (see page 3341) or PathInfo (see page 3341) properties.

When an action item is chosen as the default action item, the value of its Enabled (see page 3340) property is ignored. Thus, if an action item sets Enabled (see page 3340) to false and Default to true, TWebDispatcher (see page 3349) will first look for a match with any other action item, even if the PathInfo (see page 3341) and MethodType (see page 3341) of the default action item match the request perfectly. Then, only after all other action items have been rejected, TWebDispatcher (see page

3349) selects the default action item. If the default action item is enabled, it may be selected before all other action items have been considered because the MethodType (see page 3341) and PathInfo (see page 3341) match the request.

The default action item must be prepared to generate a response to any HTTP request, for any MethodType (see page 3341) and PathInfo (see page 3341). This may consist of simply setting a response status code of 404 (the URI could not be matched) or 405 (the requested method is not supported).

Setting Default to true sets the Default property of the previous default action item to false.

See Also

Enabled (see page 3340)

OnAction (see page 3343)

MethodType (see page 3341)

PathInfo (see page 3341)

TWebDispatcher (see page 3349)

StatusCode (see page 3413)

Choosing a Default Action Item

13.1.16.5.2 TWebActionItem.Enabled Property

Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 3341) and PathInfo (see page 3341) properties.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Set Enabled to false to disable an action item. TWebDispatcher (see page 3349) will ignore disabled action items when it looks for an action item to handle a request.

When an action item is chosen as the default action item, the value of its Enabled property is ignored. Thus, if an action item sets Enabled to false and Default (see page 3339) to true, TWebDispatcher (see page 3349) will first look for a match with any other action item, even if the PathInfo (see page 3341) and MethodType (see page 3341) of the default action item match the request perfectly. Then, only after all other action items have been rejected, the TWebDispatcher (see page 3349) selects the default action item. If the default action item is enabled, it may be selected before all other action items have been considered because the MethodType (see page 3341) and PathInfo (see page 3341) match the request.

See Also

Default (see page 3339)

MethodType (see page 3341)

OnAction (see page 3343)

PathInfo (see page 3341)

TWebDispatcher (see page 3349)

Choosing a Default Action Item

Enabling and Disabling Action Items

13.1.16.5.3 TWebActionItem.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class TWebActionItem.

13.1.16.5.4 TWebActionItem.Name Property

Identifies an individual action item when it is referenced from other objects.

Pascal

```
property Name: string;
```

C++

```
__property UnicodeString Name;
```

Description

Use Name to uniquely specify an individual action item in code anywhere in the application. For action items, Name is equivalent to the DisplayName property. Therefore, when setting the Name property, the new value must not match the DisplayName of any other action item in the list maintained by TCustomWebDispatcher (see page 3318). Trying to set the Name to a DisplayName that is already in use raises an exception.

See Also

DisplayName

Name

13.1.16.5.5 TWebActionItem.PathInfo Property

Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.

Pascal

```
property PathInfo: string;
```

C++

```
__property UnicodeString PathInfo;
```

Description

Set PathInfo to the path portion of the URI for the target of any request messages serviced by the action item. TWebDispatcher (see page 3349) matches action items to request messages by comparing PathInfo with the path portion of the URI specified as the target of the request. If the PathInfo properties match, and the MethodType (see page 3341) matches, TWebDispatcher (see page 3349) invokes the action item to respond to the request message.

If the action item has an associated content producer, the value of PathInfo is the name of the producer unless it is explicitly changed.

See Also

OnAction (see page 3343)

MethodType (see page 3341)

TWebDispatcher (see page 3349)

PathInfo (see page 3382)

Producer (see page 3342)

The Target URL

13.1.16.5.6 TWebActionItem.Producer Property

Specifies a content producer that generates the content of response messages when the action item executes.

Pascal

```
property Producer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer Producer;
```

Description

Set Producer to associate the action item with a content producer of the type TCustomContentProducer (see page 3313). (If a content producer of type IProduceContent (see page 3420) is used instead of TCustomContentProducer (see page 3313), then the property ProducerContent (see page 3342) should be set instead.) If Producer is not nil (Delphi) or NULL (C++), then when the web dispatcher passes a request message to the action item, the action item sets the Content property of the response message to the value returned by the producer's Content method. After the content producer has assigned the response's Content property, the action item receives an OnAction (see page 3343) event, where it can optionally set other response properties or send the response message.

If Producer is nil (Delphi) or NULL (C++), the action item must use the OnAction (see page 3343) event handler to influence the response message.

The properties Producer and ProducerContent (see page 3342) are mutually exclusive. If one is set, the other must not be.

Note: Each content producer can be associated with only one action item at a time. Setting the Producer property of one action item causes the Web dispatcher that manages that item to check whether any other action item has its Producer property set to the same component. If another action item was associated with the same producer, its Producer property is changed to nil (Delphi) or NULL (C++).

See Also

Content (see page 3314)

OnAction (see page 3343)

ProducerContent (see page 3342)

Content (see page 3404)

Creating Web Pages with an InternetExpress Page Producer

Responding to Request Messages with Action Items

13.1.16.5.7 TWebActionItem.ProducerContent Property

Specifies a content producer that generates the content of response messages when the action item executes.

Pascal

```
property ProducerContent: TComponent;
```

C++

```
__property TComponent * ProducerContent;
```

Description

This property is identical to Producer (see page 3342) except for the type of content producer. ProducerContent is used when the producer is of type IProduceContent (see page 3420) instead of TCustomContentProducer (see page 3313). In the latter case, the property Producer (see page 3342) is set instead. Note that Producer (see page 3342) and ProducerContent are mutually exclusive; if one is set, the other must not be.

See Also

Producer (see page 3342)

13.1.16.6 TWebActionItem Events

The events of the TWebActionItem class are listed here.

Published Events

	Name	Description
	OnAction (see page 3343)	Occurs when TWebDispatcher (see page 3349) matches the action item to a newly-arrived HTTP request message.

13.1.16.6.1 TWebActionItem.OnAction Event

Occurs when TWebDispatcher (see page 3349) matches the action item to a newly-arrived HTTP request message.

Pascal

```
property OnAction: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAction;
```

Description

Write an OnAction event handler to respond to HTTP request messages that match the MethodType (see page 3341) and PathInfo (see page 3341) properties. OnAction occurs immediately after the response message's Content property has been initialized by the action item's Producer (see page 3342) property (if Producer (see page 3342) is set).

In an OnAction event handler, read Request and perform the requested Method if possible. Set the properties of the Response parameter as appropriate. If the OnAction event handler does not service the response, or if it only partially handles the request and must let another action item finish the response, set the Handled parameter to false.

An OnAction event handler can call the SendRedirect or SendResponse method of Response to send the HTTP response message to the Web client. If the OnAction event handler does not send the response, the Web application will send it after all action items have finished filling in the Response.

Tip: The OnAction event handler can use descendants of TCustomContentProducer (see page 3313) to help assemble the contents of the response message.

See Also

MethodType (see page 3341)

PathInfo (see page 3341)

TCustomContentProducer (see page 3313)

TWebRequest (see page 3358)

TWebResponse (see page 3389)

SendRedirect (see page 3398)

SendResponse (see page 3398)

Producer (see page 3342)

Creating Web Pages with an InternetExpress Page Producer

Responding to Request Messages with Action Items

Sending the Response

13.1.17 TWebActionItems Class

TWebActionItems is a collection of TWebActionItem (see page 3335) objects.

Class Hierarchy

Pascal

```
TWebActionItems = class(TCollection);
```

C++

```
class TWebActionItems : public TCollection;
```

File

HTTPApp

Description

TCustomWebDispatcher (see page 3318) uses TWebActionItems to store and manipulate the individual action items it manages. TWebActionItems provides properties and events to

Add (see page 3346) new action items to the collection.

Copy the list of action items from another TWebActionItems.

Get the attributes of specified action items in the collection.

Empty the collection of action items.

See Also

TCustomWebDispatcher (see page 3318)

TWebActionItem (see page 3335)

13.1.17.1 TWebActionItems Members

The following tables list the members exposed by TWebActionItems.

Protected Methods

	Name	Description
	GetAttr (see page 3346)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

	GetAttrCount (see page 3346)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 3347)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 3347)	Returns the owner of an object.
	SetItemName (see page 3348)	Initializes the name of a newly inserted collection item.
	Update (see page 3348)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 3346)	Adds a new action item to the end of the collection.
	Create (see page 3345)	Creates an instance of TWebActionItems (see page 3344).

Public Properties

	Name	Description
	Items (see page 3349)	Lists the TWebActionItem (see page 3335) objects stored by this TWebActionItems (see page 3344) object.
R	WebDispatcher (see page 3349)	Provides access to the dispatcher that uses TWebActionItems (see page 3344) to store and manipulate its action items.

13.1.17.2 TWebActionItems.Create Constructor

Creates an instance of TWebActionItems (see page 3344).

Pascal

```
constructor Create(WebDispatcher: TCustomWebDispatcher; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TWebActionItems(TCustomWebDispatcher WebDispatcher, TCollectionItemClass ItemClass);
```

Description

Do not call the constructor for TWebActionItems (see page 3344). TWebActionItems (see page 3344) is tailored to work with TCustomWebDispatcher (see page 3318). The TCustomWebDispatcher (see page 3318) object creates TWebActionItems (see page 3344) in its own constructor.

See Also

Create

13.1.17.3 TWebActionItems Methods

The methods of the TWebActionItems class are listed here.

Protected Methods

	Name	Description
	GetAttr (see page 3346)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 3346)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 3347)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 3347)	Returns the owner of an object.
	SetItemName (see page 3348)	Initializes the name of a newly inserted collection item.
	Update (see page 3348)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 3346)	Adds a new action item to the end of the collection.

13.1.17.3.1 TWebActionItems.Add Method

Adds a new action item to the end of the collection.

Pascal

```
function Add: TWebActionItem;
```

C++

```
__fastcall TWebActionItem Add();
```

Description

Call Add to add an action item to the collection of action items managed by the dispatcher. The new item is added to the end of the Items (see page 3349) array. Add returns the action item so that its properties can be set to specify its use.

Because the dispatcher looks through the action items in order when it tries to find an action item to handle an HTTP request message, add any action items that preprocess HTTP requests first. Then add the most specific action items. This way, more general action items will not be matched to request messages before a closer match can be found.

See Also

Items (see page 3349)

13.1.17.3.2 TWebActionItems.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount

GetItemAttr

13.1.17.3.3 TWebActionItems.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr

GetItemAttr

13.1.17.3.4 TWebActionItems.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

13.1.17.3.5 TWebActionItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

13.1.17.3.6 TWebActionItems.SetItemName Method

Initializes the name of a newly inserted collection item.

Pascal

```
procedure SetItemName(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall SetItemName(TCollectionItem Item);
```

Description

The Insert method calls SetItemName to initialize the Name property of items when it inserts them into the collection.

As implemented in TCollection, SetItemName does nothing. Some TCollection descendants override this method to provide collection items with default names.

See Also

Insert

Name

13.1.17.3.7 TWebActionItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed

Changed

13.1.17.4 TWebActionItems Properties

The properties of the TWebActionItems class are listed here.

Public Properties

	Name	Description
	Items (see page 3349)	Lists the TWebActionItem (see page 3335) objects stored by this TWebActionItems (see page 3344) object.
R	WebDispatcher (see page 3349)	Provides access to the dispatcher that uses TWebActionItems (see page 3344) to store and manipulate its action items.

13.1.17.4.1 TWebActionItems.Items Property

Lists the TWebActionItem (see page 3335) objects stored by this TWebActionItems (see page 3344) object.

Pascal

```
property Items [Index: Integer]: TWebActionItem;
```

C++

```
__property TWebActionItem Items[int Index];
```

Description

Use Items to get access to the individual action items stored by the TWebActionItems (see page 3344) object. Items is an indexed array, where the first action item has index 0, the second action item has index 1, and so on. Use Items with the Count property to iterate over all the action items in the collection.

See Also

Count

TWebActionItem (see page 3335)

operator_sb

13.1.17.4.2 TWebActionItems.WebDispatcher Property

Provides access to the dispatcher that uses TWebActionItems (see page 3344) to store and manipulate its action items.

Pascal

```
property WebDispatcher: TCustomWebDispatcher;
```

C++

```
__property TCustomWebDispatcher WebDispatcher;
```

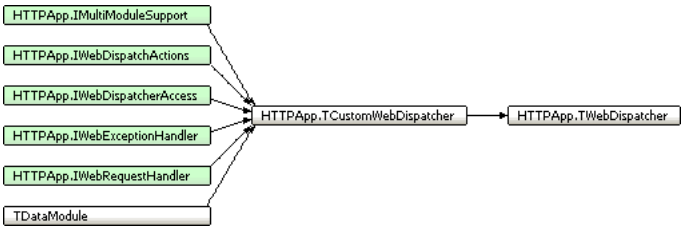
Description

Read WebDispatcher to get access to the dispatcher object that uses the collection of action items to respond to HTTP request messages.

13.1.18 TWebDispatcher Class

TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.

Class Hierarchy



Pascal

```
TWebDispatcher = class(TCustomWebDispatcher);
```

C++

```
class TWebDispatcher : public TCustomWebDispatcher;
```

File

HTTPApp

Description

Use TWebDispatcher to convert an ordinary data module to a Web module. Add a TWebDispatcher object to an existing data module that contains other nonvisual components and business rules for the Web application. Then delete the automatically generated Web module from the Web application and replace it with the data module that now contains the Web dispatcher.

TWebDispatcher enables the Web server application to respond to HTTP request messages. Only one Web dispatcher can be added to an application. This is why the automatically generated Web module, which is a type of Web dispatcher, must be deleted from the Web application before the data module containing TWebDispatcher can be added.

The TWebDispatcher object manages a collection of action items which know how to respond to HTTP request messages. View, add, and delete the action items managed by the dispatcher using the Action (see page 3327) editor which is opened by double-clicking on the TWebDispatcher icon.

TWebDispatcher publishes some of the properties and events introduced by TCustomWebDispatcher (see page 3318), but introduces no new properties, events, or methods of its own.

See Also

The Web Dispatcher

The Web Module

13.1.18.1 TWebDispatcher Members

The following tables list the members exposed by TWebDispatcher.

Protected Methods

	Name	Description
	Access_Request (see page 3322)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 3322)	This is Access_Response, a member of class TCustomWebDispatcher.
	DispatchAction (see page 3323)	Passes an HTTP request message to the action items in Actions (see page 3327) until it is handled.
	DoAfterDispatch (see page 3324)	Generates an AfterDispatch (see page 3329) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 3324)	Generates a BeforeDispatch (see page 3329) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.

	FinishContext (see page 3325)	This is FinishContext, a member of class TCustomWebDispatcher.
	HandleException (see page 3325)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 3325)	Processes an HTTP request message.
	InitContext (see page 3326)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 3326)	This is InitModule, a member of class TCustomWebDispatcher.
V	Notification (see page 3326)	Responds to notifications when objects are added to the module.

Public Methods

	Name	Description
	ActionByName (see page 3322)	Returns the action item identified by the AName parameter.
V	Create (see page 3320)	Creates an instance of TCustomWebDispatcher (see page 3318).
V	Destroy (see page 3321)	Frees an instance of TCustomWebDispatcher (see page 3318).
	FinishContext (see page 3420)	This is FinishContext, a member of class IMultiModuleSupport.
	HandleException (see page 3433)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 3435)	Processes an HTTP request message.
	InitContext (see page 3420)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 3420)	This is InitModule, a member of class IMultiModuleSupport.
	Request (see page 3432)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 3432)	Returns the Web response object for the HTTP response message that the application is assembling.

Public Properties

	Name	Description
R	Action (see page 3327)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 3318) object.
R	Request (see page 3327)	Provides access to the TWebRequest (see page 3358) object that the dispatcher is currently handling.
R	Response (see page 3328)	Provides access to the TWebResponse (see page 3389) object for the message the dispatcher is currently handling.

Published Events

	Name	Description
	AfterDispatch (see page 3352)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3353)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 3353)	Occurs when an unhandled exception occurs in the application.

Published Properties

	Name	Description
	Actions (see page 3351)	Represents the collection of action items managed by the TCustomWebDispatcher object.

13.1.18.2 TWebDispatcher Properties

The properties of the TWebDispatcher class are listed here.

Published Properties

	Name	Description
	Actions (see page 3351)	Represents the collection of action items managed by the TCustomWebDispatcher object.

13.1.18.2.1 TWebDispatcher.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 3327)

ActionByName (see page 3322)

TWebActionItems (see page 3344)

13.1.18.3 TWebDispatcher Events

The events of the TWebDispatcher class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 3352)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3353)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 3353)	Occurs when an unhandled exception occurs in the application.

13.1.18.3.1 TWebDispatcher.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 3327)

BeforeDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

13.1.18.3.2 TWebDispatcher.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch event.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

Enabling and Disabling Action Items

13.1.18.3.3 TWebDispatcher.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 3327)

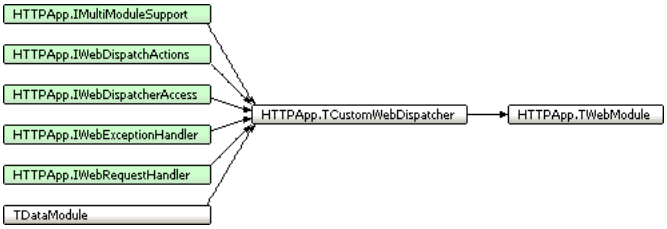
AfterDispatch (see page 3329)

TWebResponse (see page 3389)

13.1.19 TWebModule Class

TWebModule is the automatically generated Web module for a Web server application.

Class Hierarchy



Pascal

```
TWebModule = class(TCustomWebDispatcher);
```

C++

```
class TWebModule : public TCustomWebDispatcher;
```

File

HTTPApp

Description

When a new Web application is created, it automatically contains a Web module. The Web module serves as a repository for nonvisual components such as TPageProducer, TSQLDataSet, TDataSetTableProducer, etc. It also enables the Web server application to respond to HTTP request messages by passing the request and response objects to the appropriate action items. An application can have only one Web module.

The TWebModule object manages a collection of action items which know how to respond to HTTP request messages. View, add and delete the action items managed by the Web module using the Action (see page 3327) editor which is opened by double clicking on the TWebModule icon.

If a data module already exists which contains many of the non-visual components and business rules for the application, the Web module can be replaced by that data module. Simply delete the Web module, add the new data module in its place, and add a TWebDispatcher (see page 3349) object to the new data module.

See the WebServ demo for an example of how to use TWebModule.

Note: When you create a new Apache module project and save it with a new name, it may be necessary to change the name of the apache_module (in the exports section of the project source code) by hand.

See Also

TDataSetTableProducer

TPageProducer (see page 3496)

TSQLDataSet

TWebActionItems (see page 3344)

TWebDispatcher (see page 3349)

The Web Module

13.1.19.1 TWebModule Members

The following tables list the members exposed by TWebModule.

Protected Methods

	Name	Description
	Access_Request (see page 3322)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 3322)	This is Access_Response, a member of class TCustomWebDispatcher.
	DispatchAction (see page 3323)	Passes an HTTP request message to the action items in Actions (see page 3327) until it is handled.
	DoAfterDispatch (see page 3324)	Generates an AfterDispatch (see page 3329) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 3324)	Generates a BeforeDispatch (see page 3329) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
	FinishContext (see page 3325)	This is FinishContext, a member of class TCustomWebDispatcher.
	HandleException (see page 3325)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 3325)	Processes an HTTP request message.
	InitContext (see page 3326)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 3326)	This is InitModule, a member of class TCustomWebDispatcher.
V	Notification (see page 3326)	Responds to notifications when objects are added to the module.

Public Methods

	Name	Description
	ActionByName (see page 3322)	Returns the action item identified by the AName parameter.
V	Create (see page 3356)	Creates an instance of TWebModule (see page 3354).
V	Destroy (see page 3321)	Frees an instance of TCustomWebDispatcher (see page 3318).
	FinishContext (see page 3420)	This is FinishContext, a member of class IMultiModuleSupport.
	HandleException (see page 3433)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 3435)	Processes an HTTP request message.
	InitContext (see page 3420)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 3420)	This is InitModule, a member of class IMultiModuleSupport.
	Request (see page 3432)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 3432)	Returns the Web response object for the HTTP response message that the application is assembling.

Public Properties

	Name	Description
R	Action (see page 3327)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 3318) object.
R	Request (see page 3327)	Provides access to the TWebRequest (see page 3358) object that the dispatcher is currently handling.
R	Response (see page 3328)	Provides access to the TWebResponse (see page 3389) object for the message the dispatcher is currently handling.

Published Events

	Name	Description
	AfterDispatch (see page 3357)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3357)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 3358)	Occurs when an unhandled exception occurs in the application.

Published Properties

	Name	Description
	Actions (see page 3356)	Represents the collection of action items managed by the TCustomWebDispatcher object.

13.1.19.2 TWebModule.Create Constructor

Creates an instance of TWebModule (see page 3354).

Pascal

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

C++

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

Description

Do not create instances at runtime. The Web module is created automatically when a Web application project is created at design time. Every time a Web server application is run in response to an HTTP request message, a new TWebModule (see page 3354) is created based on the Web module that was created at design time.

Create generates an OnCreate event, so that Web applications can initialize the Web module before the new HTTP request message is processed.

See Also

OnCreate

Create_d

13.1.19.3 TWebModule Properties

The properties of the TWebModule class are listed here.

Published Properties

	Name	Description
	Actions (see page 3356)	Represents the collection of action items managed by the TCustomWebDispatcher object.

13.1.19.3.1 TWebModule.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 3327)

ActionByName (see page 3322)

TWebActionItems (see page 3344)

13.1.19.4 TWebModule Events

The events of the TWebModule class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 3357)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 3357)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 3358)	Occurs when an unhandled exception occurs in the application.

13.1.19.4.1 TWebModule.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 3327)

BeforeDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

13.1.19.4.2 TWebModule.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or

disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch event.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

TWebResponse (see page 3389)

Dispatching Request Messages

Enabling and Disabling Action Items

13.1.19.4.3 TWebModule.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 3327)

AfterDispatch (see page 3329)

TWebResponse (see page 3389)

13.1.20 TWebRequest Class

TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.

Class Hierarchy

Pascal

```
TWebRequest = class(TObject);
```

C++

```
class TWebRequest : public TObject;
```

File

HTTPApp

Description

Use a descendant of TWebRequest to interpret an HTTP request message.

Web application objects create TWebRequest objects to capture client requests. The properties of the TWebRequest objects are obtained from the HTTP request message. The Web application uses TWebRequest objects to create corresponding TWebResponse (see page 3389) objects. The TWebDispatcher (see page 3349) for the application then invokes the TWebActionItem (see page 3335) that is associated with the PathInfo (see page 3382) of the TWebRequest object.

TWebRequest has several descendants which represent client requests for different types of Web server applications. These descendants are

TApacheRequest objects for Apache server applications.

TCGIRequest objects for CGI server applications.

TISAPIRequest objects for DLLs that respond to the ISAPI or NSAPI servers (Windows only).

TWinCGIRequest objects for Windows-based CGI servers (Windows only).

See Also

TApacheRequest

TCGIRequest

TISAPIRequest (see page 4510)

TWebActionItem (see page 3335)

TWebApplication

TWebDispatcher (see page 3349)

TWebResponse (see page 3389)

TWinCGIRequest

Creating HTTP Response Messages

Responding to Request Messages with Action Items

The Structure of a Web Broker Application

Dispatching Requests and Responses

13.1.20.1 TWebRequest Members

The following tables list the members exposed by TWebRequest.

Protected Methods

	Name	Description
	GetDateVariable (see page 3365)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
	GetIntegerVariable (see page 3366)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
	GetInternalPathInfo (see page 3366)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 3366)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 3366)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 3369)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	Create (see page 3361)	Creates an instance of TWebRequest (see page 3358).
	Destroy (see page 3361)	Destroys an instance of TWebRequest (see page 3358).
	ExtractContentFields (see page 3363)	Parses the Content (see page 3373) property into its constituent fields.
	ExtractCookieFields (see page 3363)	Parses the Cookie (see page 3377) property into its constituent fields.
	ExtractFields (see page 3363)	This is the overview for the ExtractFields method overload.
	ExtractQueryFields (see page 3364)	Parses the Query (see page 3384) property into its constituent fields.
	GetFieldByName (see page 3365)	Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 3367)	Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.
	ReadString (see page 3368)	Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.
	ReadUnicodeString (see page 3368)	This is ReadUnicodeString, a member of class TWebRequest.
	TranslateURI (see page 3368)	Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
	WriteClient (see page 3369)	Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 3370)	Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.
	WriteString (see page 3370)	Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

Public Properties

	Name	Description
	Accept (see page 3371)	Reports the value of the Accept header of the HTTP request message.
	Authorization (see page 3372)	Reports the value of the Authorization header of the HTTP request message.
	CacheControl (see page 3372)	Reports the value of the Cache-Control header of the HTTP request message.
	Connection (see page 3373)	Reports the value of the Connection header of the HTTP request message.
	Content (see page 3373)	Represents the unparsed contents of the HTTP request message.
	ContentEncoding (see page 3374)	Reports the value of the Content (see page 3373)-Encoding header of the HTTP request message.
	ContentFields (see page 3375)	Reports the parsed content of a POST request message.
	ContentLength (see page 3375)	Reports the value of the Content (see page 3373)-Length header of the HTTP request message.
	ContentParser (see page 3376)	Refers to the object's content parser object.
	ContentType (see page 3376)	Reports the value of the Content (see page 3373)-Type header of the HTTP request message.
	ContentVersion (see page 3376)	Reports the value of the Content (see page 3373)-Version header of the HTTP request message.
	Cookie (see page 3377)	Reports the value of the Cookie header of the HTTP request message.
	CookieFields (see page 3377)	Represents the parsed contents of the Cookie (see page 3377) header of an HTTP request message.
	Date (see page 3378)	Reports the value of the Date header of the HTTP request message.

	DerivedFrom (see page 3378)	Reports the value of the Derived-From (see page 3379) header of the HTTP request message.
	Expires (see page 3379)	Reports the value of the Expires header of the HTTP request message.
	Files (see page 3379)	Provides access to the files uploaded in the HTTP request.
	From (see page 3379)	Reports the value of the From header of the HTTP request message.
	Host (see page 3380)	Reports the value of the Host header of the HTTP request message.
	IfModifiedSince (see page 3380)	Reports the value of the If-Modified-Since header of the HTTP request message.
	InternalPathInfo (see page 3381)	Returns the path info portion of the URL (see page 3388).
	InternalScriptName (see page 3381)	Returns the ScriptName (see page 3386) part of the URL (see page 3388).
	Method (see page 3382)	This is Method, a member of class TWebRequest.
	MethodType (see page 3382)	This is MethodType, a member of class TWebRequest.
	PathInfo (see page 3382)	Reports the value of the path information (if any) of the URL (see page 3388) specified in the HTTP request message.
	PathTranslated (see page 3383)	Represents a translation of the PathInfo (see page 3382) property to a fully qualified path on the Web server.
	ProtocolVersion (see page 3383)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
	Query (see page 3384)	Indicates the value of the query information (if any) of the URL (see page 3388) specified in the HTTP request message.
	QueryFields (see page 3384)	Represents the parsed contents of the Query (see page 3384) in the URL (see page 3388) of the HTTP request message.
	RawContent (see page 3385)	This is RawContent, a member of class TWebRequest.
	Referer (see page 3385)	Reports the value of the Referer header of the HTTP request message.
	RemoteAddr (see page 3386)	Indicates the remote IP address of the client associated with the HTTP request message.
	RemoteHost (see page 3386)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
	ScriptName (see page 3386)	Indicates the value of the script information (if any) of the URL (see page 3388) specified in the HTTP request message.
	ServerPort (see page 3387)	Indicates the port number of the port on the Web server that receives the HTTP request message.
	Title (see page 3387)	Reports the value of the Title header of the HTTP request message.
	URL (see page 3388)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
	UserAgent (see page 3388)	Reports the value of the User-Agent header of the HTTP request message.

13.1.20.2 TWebRequest.Create Constructor

Creates an instance of TWebRequest (see page 3358).

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebRequest();
```

Description

Applications shouldn't need to call Create to instantiate a request object. The TWebApplication object creates a TWebRequest (see page 3358) object whenever it receives a client request.

See Also

TWebApplication

13.1.20.3 TWebRequest.Destroy Destructor

Destroys an instance of TWebRequest (see page 3358).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequest();
```

Description

Do not call Destroy. When the request message is fully handled, TWebApplication destroys the request object by calling its Free method.

Destroy frees the TStrings objects used to represent the ContentFields (see page 3375), CookieFields (see page 3377), and QueryFields (see page 3384) properties.

See Also

ContentFields (see page 3375)

CookieFields (see page 3377)

System::TObject::Free

QueryFields (see page 3384)

TWebApplication

13.1.20.4 TWebRequest Methods

The methods of the TWebRequest class are listed here.

Protected Methods

	Name	Description
	GetDateVariable (see page 3365)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
	GetIntegerVariable (see page 3366)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
	GetInternalPathInfo (see page 3366)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 3366)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 3366)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 3369)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	ExtractContentFields (see page 3363)	Parses the Content (see page 3373) property into its constituent fields.
	ExtractCookieFields (see page 3363)	Parses the Cookie (see page 3377) property into its constituent fields.
	ExtractFields (see page 3363)	This is the overview for the ExtractFields method overload.
	ExtractQueryFields (see page 3364)	Parses the Query (see page 3384) property into its constituent fields.
	GetFieldByName (see page 3365)	Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 3367)	Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.
	ReadString (see page 3368)	Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.
	ReadUnicodeString (see page 3368)	This is ReadUnicodeString, a member of class TWebRequest.
	TranslateURI (see page 3368)	Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.

	WriteClient (see page 3369)	Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 3370)	Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.
	WriteString (see page 3370)	Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

13.1.20.4.1 TWebRequest.ExtractContentFields Method

Parses the Content (see page 3373) property into its constituent fields.

Pascal

```
procedure ExtractContentFields(Strings: TStrings);
```

C++

```
__fastcall ExtractContentFields(TStrings Strings);
```

Description

Call ExtractContentFields to fill the Strings parameter with the individual fields in the Content (see page 3373) property. ExtractContentFields assumes that fields are separated by ampersands (&). This is the format used when the MethodType (see page 3382) is mtPost. If the Content (see page 3373) property is not assigned, ExtractContentFields does nothing.

See Also

Content (see page 3373)

ContentFields (see page 3375)

MethodType (see page 3382)

TStrings

13.1.20.4.2 TWebRequest.ExtractCookieFields Method

Parses the Cookie (see page 3377) property into its constituent fields.

Pascal

```
procedure ExtractCookieFields(Strings: TStrings);
```

C++

```
__fastcall ExtractCookieFields(TStrings Strings);
```

Description

Call ExtractCookieFields to fill the Strings parameter with the individual fields in the Cookie (see page 3377) property. ExtractCookieFields assumes that fields are separated by semicolons. If the Cookie (see page 3377) property is not assigned, ExtractCookieFields does nothing.

See Also

Cookie (see page 3377)

CookieFields (see page 3377)

TStrings

13.1.20.4.3 ExtractFields Method

This is the overview for the ExtractFields method overload.

Overload List

	Name	Description
	TWebRequest.ExtractFields (TSysCharSet, TSysCharSet, AnsiString, TStrings) (see page 3364)	Parses a multi-valued string into its constituent fields.
	TWebRequest.ExtractFields (TSysCharSet, TSysCharSet, PChar, TStrings) (see page 3364)	Parses a multi-valued string into its constituent fields.

13.1.20.4.3.1 TWebRequest.ExtractFields Method (TSysCharSet, TSysCharSet, AnsiString, TStrings)

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content: PChar;
Strings: TStrings); overload;
procedure ExtractFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; const Content:
AnsiString; Strings: TStrings); overload;
```

C++

```
__fastcall ExtractFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings);
__fastcall ExtractFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const AnsiString
Content, TStrings Strings);
```

Description

Call ExtractFields to parse a multi-valued HTTP header into separate strings for each value. For example, if the separator character of the UserAgent (see page 3388) property is known, ExtractFields can parse this property into separate UserAgent (see page 3388) values.

Separators is a set of characters that are used to separate individual values within the multi-valued string. WhiteSpace is a set of characters that are to be ignored when parsing the string. Content (see page 3373) is the multi-valued string to be parsed. Strings is the TStrings object that receives the individual values that are parsed from Content (see page 3373).

See Also

ExtractContentFields (see page 3363)

ExtractCookieFields (see page 3363)

ExtractQueryFields (see page 3364)

TStrings

13.1.20.4.4 TWebRequest.ExtractQueryFields Method

Parses the Query (see page 3384) property into its constituent fields.

Pascal

```
procedure ExtractQueryFields(Strings: TStrings);
```

C++

```
__fastcall ExtractQueryFields(TStrings Strings);
```

Description

Call ExtractQueryFields to fill the Strings parameter with the individual fields in the Query (see page 3384) property. ExtractQueryFields assumes that fields are separated by ampersands (&). If the Query (see page 3384) property is not assigned, ExtractQueryFields does nothing.

See Also

Query (see page 3384)

QueryFields (see page 3384)

TStrings

13.1.20.4.5 TWebRequest.GetDateVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; virtual; abstract;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement GetDateVariable to provide a method by which the date-time values are read from the header strings of the HTTP request message. The request object uses GetDateVariable to access the properties that represent dates from the HTTP request headers.

See Also

Date (see page 3378)

Expires (see page 3379)

IfModifiedSince (see page 3380)

13.1.20.4.6 TWebRequest.GetFieldByName Method

Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.

Pascal

```
function GetFieldByName(const Name: AnsiString): AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement GetFieldByName to provide access to arbitrary header fields in the HTTP request message. Use GetFieldByName to read the value of an HTTP header variable that is not surfaced as one of the properties of TWebRequest (see page 3358). For information on possible HTTP request header variables, see the current Internet standards documents.

See Also

GetStringVariable (see page 3366)

Properties That Contain Request Header Information

13.1.20.4.7 TWebRequest.GetIntegerVariable Method

Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement GetIntegerVariable to provide a method by which the integer values are read from the header strings of the HTTP request message. The request object uses GetIntegerVariable to access the properties that represent integers from the HTTP request headers.

See Also

ContentLength (see page 3375)

ServerPort (see page 3387)

13.1.20.4.8 TWebRequest.GetInternalPathInfo Method

Pascal

```
function GetInternalPathInfo: AnsiString; virtual;
```

C++

```
virtual __fastcall AnsiString GetInternalPathInfo();
```

Description

This is GetInternalPathInfo, a member of class TWebRequest.

13.1.20.4.9 TWebRequest.GetInternalScriptName Method

Pascal

```
function GetInternalScriptName: AnsiString; virtual;
```

C++

```
virtual __fastcall AnsiString GetInternalScriptName();
```

Description

This is GetInternalScriptName, a member of class TWebRequest.

13.1.20.4.10 TWebRequest.GetStringVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement GetStringVariable to provide a method by which the string values are read from the header strings of the HTTP request message. The request object uses GetStringVariable to access the properties that represent strings from the HTTP request headers.

See Also

Accept (see page 3371)

CacheControl (see page 3372)

Connection (see page 3373)

Content (see page 3373)

ContentEncoding (see page 3374)

ContentType (see page 3376)

ContentVersion (see page 3376)

Cookie (see page 3377)

DerivedFrom (see page 3378)

From (see page 3379)

Host (see page 3380)

TWebRequest (see page 3358)

PathInfo (see page 3382)

PathTranslated (see page 3383)

ProtocolVersion (see page 3383)

Query (see page 3384)

Referer (see page 3385)

RemoteAddr (see page 3386)

RemoteHost (see page 3386)

ScriptName (see page 3386)

Title (see page 3387)

URL (see page 3388)

UserAgent (see page 3388)

13.1.20.4.11 TWebRequest.ReadClient Method

Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement ReadClient. ReadClient is used to read content information from the HTTP request. Use ReadClient to get the next chunk of information when the content of the request message is too large for the Content (see page 3373) property.

Use Buffer to provide a buffer to contain the content information. Use Count to specify the number of bytes to read. ReadClient returns the number of bytes actually read.

See Also

Content (see page 3373)

ReadString (see page 3368)

WriteClient (see page 3369)

13.1.20.4.12 TWebRequest.ReadString Method

Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement ReadString. ReadString is used to load content information from the HTTP request into a string. Use ReadString to get the next chunk of information when the content of the request message is too large for the Content (see page 3373) property.

See Also

Content (see page 3373)

ReadClient (see page 3367)

WriteString (see page 3370)

13.1.20.4.13 TWebRequest.ReadUnicodeString Method

Pascal

```
function ReadUnicodeString(Count: Integer): string;
```

C++

```
__fastcall UnicodeString ReadUnicodeString(int Count);
```

Description

This is ReadUnicodeString, a member of class TWebRequest.

13.1.20.4.14 TWebRequest.TranslateURI Method

Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal

representation of a location on the server.

Pascal

```
function TranslateURI(const URI: string): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString TranslateURI(const UnicodeString URI) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement TranslateURI. TranslateURI is used to translate the URI specified in a client request into a format that can be used by the server application. TranslateURI is not relevant to CGI applications.

Note: On Windows, the internal format varies, depending on whether the server application is an ISAPI or NSAPI application.

See Also

PathTranslated (see page 3383)

URL (see page 3388)

13.1.20.4.15 TWebRequest.UpdateMethodType Method

Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Pascal

```
procedure UpdateMethodType;
```

C++

```
__fastcall UpdateMethodType();
```

Description

Use UpdateMethodType to update the type of response that is requested by the Web client.

See Also

TWebRequest (see page 3358)

13.1.20.4.16 TWebRequest.WriteClient Method

Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement WriteClient. WriteClient is used by TWebResponse (see page 3389) to send the contents of a response to the Web client.

See Also

ReadClient (see page 3367)

TWebResponse (see page 3389)

WriteString (see page 3370)

13.1.20.4.17 TWebRequest.WriteHeaders Method

Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.

Pascal

```
function WriteHeaders(StatusCode: Integer; const ReasonString: AnsiString; const Headers:
AnsiString): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean WriteHeaders(int StatusCode, const AnsiString ReasonString, const
AnsiString Headers) = 0;
```

Description

Use WriteHeaders to write an HTTP header string.

See Also

WriteClient (see page 3369)

WriteString (see page 3370)

13.1.20.4.18 TWebRequest.WriteString Method

Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: AnsiString): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString) = 0;
```

Description

Descendants of TWebRequest (see page 3358) implement WriteString. WriteString is used by TWebResponse (see page 3389) to send the contents of a response to the Web client.

See Also

ReadString (see page 3368)

TWebResponse (see page 3389)

WriteClient (see page 3369)

13.1.20.5 TWebRequest Properties

The properties of the TWebRequest class are listed here.

Public Properties

	Name	Description
R	Accept (see page 3371)	Reports the value of the Accept header of the HTTP request message.
R	Authorization (see page 3372)	Reports the value of the Authorization header of the HTTP request message.
R	CacheControl (see page 3372)	Reports the value of the Cache-Control header of the HTTP request message.
R	Connection (see page 3373)	Reports the value of the Connection header of the HTTP request message.
R	Content (see page 3373)	Represents the unparsed contents of the HTTP request message.
R	ContentEncoding (see page 3374)	Reports the value of the Content (see page 3373)-Encoding header of the HTTP request message.

	ContentFields (see page 3375)	Reports the parsed content of a POST request message.
	ContentLength (see page 3375)	Reports the value of the Content (see page 3373)-Length header of the HTTP request message.
	ContentParser (see page 3376)	Refers to the object's content parser object.
	ContentType (see page 3376)	Reports the value of the Content (see page 3373)-Type header of the HTTP request message.
	ContentVersion (see page 3376)	Reports the value of the Content (see page 3373)-Version header of the HTTP request message.
	Cookie (see page 3377)	Reports the value of the Cookie header of the HTTP request message.
	CookieFields (see page 3377)	Represents the parsed contents of the Cookie (see page 3377) header of an HTTP request message.
	Date (see page 3378)	Reports the value of the Date header of the HTTP request message.
	DerivedFrom (see page 3378)	Reports the value of the Derived-From (see page 3379) header of the HTTP request message.
	Expires (see page 3379)	Reports the value of the Expires header of the HTTP request message.
	Files (see page 3379)	Provides access to the files uploaded in the HTTP request.
	From (see page 3379)	Reports the value of the From header of the HTTP request message.
	Host (see page 3380)	Reports the value of the Host header of the HTTP request message.
	IfModifiedSince (see page 3380)	Reports the value of the If-Modified-Since header of the HTTP request message.
	InternalPathInfo (see page 3381)	Returns the path info portion of the URL (see page 3388).
	InternalScriptName (see page 3381)	Returns the ScriptName (see page 3386) part of the URL (see page 3388).
	Method (see page 3382)	This is Method, a member of class TWebRequest.
	MethodType (see page 3382)	This is MethodType, a member of class TWebRequest.
	PathInfo (see page 3382)	Reports the value of the path information (if any) of the URL (see page 3388) specified in the HTTP request message.
	PathTranslated (see page 3383)	Represents a translation of the PathInfo (see page 3382) property to a fully qualified path on the Web server.
	ProtocolVersion (see page 3383)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
	Query (see page 3384)	Indicates the value of the query information (if any) of the URL (see page 3388) specified in the HTTP request message.
	QueryFields (see page 3384)	Represents the parsed contents of the Query (see page 3384) in the URL (see page 3388) of the HTTP request message.
	RawContent (see page 3385)	This is RawContent, a member of class TWebRequest.
	Referer (see page 3385)	Reports the value of the Referer header of the HTTP request message.
	RemoteAddr (see page 3386)	Indicates the remote IP address of the client associated with the HTTP request message.
	RemoteHost (see page 3386)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
	ScriptName (see page 3386)	Indicates the value of the script information (if any) of the URL (see page 3388) specified in the HTTP request message.
	ServerPort (see page 3387)	Indicates the port number of the port on the Web server that receives the HTTP request message.
	Title (see page 3387)	Reports the value of the Title header of the HTTP request message.
	URL (see page 3388)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
	UserAgent (see page 3388)	Reports the value of the User-Agent header of the HTTP request message.

13.1.20.5.1 TWebRequest.Accept Property

Reports the value of the Accept header of the HTTP request message.

Pascal

```
property Accept: AnsiString;
```

C++

```
__property AnsiString Accept;
```

Description

Read Accept to determine the media types the Web client can accept in response to the request. Not every HTTP request message has an Accept header. If the Accept property is not set, then any media type is potentially acceptable in the response.

Accept headers usually indicate that the response should be limited to a small set of desired types. For example, a Web client requesting an audio response might produce a TWebRequest (see page 3358) object with Accept set to the following string:

```
audio/*; q=0.2, audio/basic
```

This request indicates that the preferred response media type is audio/basic, but that any audio type is acceptable if it is the best available after an 80% markdown in quality.

For information on the format of Accept headers, see the current Internet standards documentation.

See Also

ContentType (see page 3406)

Properties That Describe the Expected Response

13.1.20.5.2 TWebRequest.Authorization Property

Reports the value of the Authorization header of the HTTP request message.

Pascal

```
property Authorization: AnsiString;
```

C++

```
__property AnsiString Authorization;
```

Description

Read Authorization to check whether the client that sent the request message has provided the correct authentication information. Authorization allows the Web application to provided limited access to the resource specified by the URL (see page 3388) property.

For more information about HTTP authentication schemes, see the Internet standards documentation.

See Also

WWWAuthenticate (see page 3415)

URL (see page 3388)

13.1.20.5.3 TWebRequest.CacheControl Property

Reports the value of the Cache-Control header of the HTTP request message.

Pascal

```
property CacheControl: AnsiString;
```

C++

```
__property AnsiString CacheControl;
```

Description

Read CacheControl to determine any Web client directives that must be obeyed by all caching mechanisms along the request/response chain. The CacheControl directives of the HTTP request need not apply to the corresponding HTTP response.

The HTTP 1.1 standard defines the following directives for CacheControl:

Value	Meaning
no-cache	The request may not be cached. The Web server validates the request before the TWebRequest (see page 3358) object is created.
no-store	The request contains sensitive information and may not be stored on any permanent medium such as a backup tape. If the request is stored temporarily in a file, the file must be deleted.
max-age=ss	The request is considered invalid if it is older than ss seconds. This directive takes precedence over the Expires (see page 3379) property, even if Expires (see page 3379) is more restrictive. Invalid requests are validated by the Web server before the TWebRequest (see page 3358) object is created.
max-stale or max-stale=ss	The client is willing to accept a response that has exceeded its expiration time. The max-stale directive can optionally specify the number of seconds by which the response can exceed its expiration time.
min-fresh=ss	The client is not willing to accept a response that expires in less than ss seconds.

See Also

Date (see page 3378)

Expires (see page 3379)

13.1.20.5.4 TWebRequest.Connection Property

Reports the value of the Connection header of the HTTP request message.

Pascal

```
property Connection: AnsiString;
```

C++

```
__property AnsiString Connection;
```

Description

Read Connection to determine the options requested by the Web client for its connection to the server application. The HTTP 1.1 standard specifies only one value for the Connection header. If the value of Connection is close, the connection should be closed after the completion of the response.

See Also

Host (see page 3380)

RemoteAddr (see page 3386)

RemoteHost (see page 3386)

ServerPort (see page 3387)

Connecting to a Data Store Using TADOConnection

13.1.20.5.5 TWebRequest.Content Property

Represents the unparsed contents of the HTTP request message.

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

Use Content to read the contents of an HTTP request message that is sending information to the server. The value of Content depends on the Method (see page 3382) of the HTTP request. When the Method (see page 3382) is POST, Content is the unparsed string giving the values to be posted with their associated fields. When the Method (see page 3382) is PUT, Content is the value that should replace the URL (see page 3388) given by the URL (see page 3388) property.

Tip: When the Method (see page 3382) is POST, the parsed version of the post request is available in the ContentFields (see page 3375) property.

Note: If the content of a message is particularly large, it is transferred in chunks. In this case, the Content property does not contain the entire message content. To read the rest of the content, use the ReadClient (see page 3367) method.

See Also

ContentEncoding (see page 3374)

ContentFields (see page 3375)

ContentLength (see page 3375)

ContentType (see page 3376)

ContentVersion (see page 3376)

ExtractContentFields (see page 3363)

TWebRequest (see page 3358)

The Content of HTTP Request Messages

13.1.20.5.6 TWebRequest.ContentEncoding Property

Reports the value of the Content (see page 3373)-Encoding header of the HTTP request message.

Pascal

```
property ContentEncoding: AnsiString;
```

C++

```
__property AnsiString ContentEncoding;
```

Description

Read ContentEncoding to determine what encoding mechanisms, if any, have been applied to the Content (see page 3373) of the HTTP request. The encoding mechanisms are listed in the order in which the Web client applied them, separated by commas. Typically, ContentEncoding indicates a compression method or an encoding scheme used to transfer extended ASCII characters over the net.

Reverse the ContentEncoding mechanisms to decode the Content (see page 3373) into the form indicated by the ContentType (see page 3376) property.

See Also

Content (see page 3373)

ContentType (see page 3376)

Properties That Describe the Content

13.1.20.5.7 TWebRequest.ContentFields Property

Reports the parsed content of a POST request message.

Pascal

```
property ContentFields: TStrings;
```

C++

```
__property TStrings ContentFields;
```

Description

Use ContentFields to read the fields of an HTTP request message when the MethodType (see page 3382) is mtPost. The Content (see page 3373) of the HTTP request is parsed, separating into fields at the ampersand character.

Each field appears as a separate string in the ContentFields list with the form Name=Value. As a TStrings object, ContentFields allows applications to manipulate the individual fields by name or value.

See Also

Content (see page 3373)

ExtractContentFields (see page 3363)

TWebRequest (see page 3358)

TStrings

Names

Values

The Content of HTTP Request Messages

13.1.20.5.8 TWebRequest.ContentLength Property

Reports the value of the Content (see page 3373)-Length header of the HTTP request message.

Pascal

```
property ContentLength: Integer;
```

C++

```
__property int ContentLength;
```

Description

Read the ContentLength property to determine the length of the Content (see page 3373) property. ContentLength allows an application to determine the end of the Content (see page 3373) property when the ContentType (see page 3376) is not text/html. A ContentLength of 0 indicates that the length of the Content (see page 3373) property is not determined, either because the request does not include any Content (see page 3373), or because the Web client did not know the length.

Note: The length of the Content (see page 3373) may be less than ContentLength when the Content (see page 3373) is passed using chunked transfer coding.

See Also

Content (see page 3373)

ContentType (see page 3376)

TWebRequest (see page 3358)

Properties That Describe the Content

13.1.20.5.9 TWebRequest.ContentParser Property

Refers to the object's content parser object.

Pascal

```
property ContentParser: TAbstractContentParser;
```

C++

```
__property TAbstractContentParser ContentParser;
```

Description

Read ContentParser to locate the TWebRequest (see page 3358) instance's TAbstractContentParser (see page 3255) object. The TAbstractContentParser (see page 3255) is responsible for parsing the HTTP request.

13.1.20.5.10 TWebRequest.ContentType Property

Reports the value of the Content (see page 3373)-Type header of the HTTP request message.

Pascal

```
property ContentType: AnsiString;
```

C++

```
__property AnsiString ContentType;
```

Description

Read ContentType to determine the media type of the Content (see page 3373) of the HTTP request message. The ContentType is a string of the form Type/SubType.

Most HTTP requests do not require any Content (see page 3373) at all. For most requests with a Method (see page 3382) that requires a value for Content (see page 3373), the ContentType is text/html. However, HTTP requests for server-specific values of Method (see page 3382) can use any media type appropriate to the request.

See Also

Accept (see page 3371)

Content (see page 3373)

TWebRequest (see page 3358)

Properties That Describe the Content

13.1.20.5.11 TWebRequest.ContentVersion Property

Reports the value of the Content (see page 3373)-Version header of the HTTP request message.

Pascal

```
property ContentVersion: AnsiString;
```

C++

```
__property AnsiString ContentVersion;
```

Description

Read ContentVersion to determine the version of the Content (see page 3373) of the HTTP request message. Most HTTP

requests do not require any Content (see page 3373) at all. When the Method (see page 3382) requires a value for Content (see page 3373), the ContentVersion may be set to indicate the version of the source of that content. The format of the ContentVersion depends on the ContentType (see page 3376) of the request.

See Also

Content (see page 3373)

ContentType (see page 3376)

Properties That Describe the Content

13.1.20.5.12 TWebRequest.Cookie Property

Reports the value of the Cookie header of the HTTP request message.

Pascal

```
property Cookie: AnsiString;
```

C++

```
__property AnsiString Cookie;
```

Description

Use Cookie to read the "cookie" strings provided in the Web client request. Cookie strings are used to track state information in a Web application. Cookie is the unparsed value of the Cookie header. The Cookie header can include multiple fields separated by semicolons. Use the CookieFields (see page 3377) property to work with the Cookie header on a field-by-field basis.

See Also

CookieFields (see page 3377)

ExtractCookieFields (see page 3363)

Cookies (see page 3407)

Properties That Describe the Expected Response

13.1.20.5.13 TWebRequest.CookieFields Property

Represents the parsed contents of the Cookie (see page 3377) header of an HTTP request message.

Pascal

```
property CookieFields: TStrings;
```

C++

```
__property TStrings CookieFields;
```

Description

Use CookieFields to work with individual "cookie" strings provided by the Web client request. The Cookie (see page 3377) property is parsed into separate fields by separating into substrings whenever a semicolon is encountered.

Cookie (see page 3377) strings are used to track state information in a Web application. Each field appears as a separate string in the CookieFields list with the form Name=Value. As a TStrings object, CookieFields allows applications to manipulate the individual fields by name or value.

Note: Applications can give state variables any name except one of the reserved cookie field names. See the Netscape documentation for a list of reserved cookie field names and their meanings.

See Also

Cookie (see page 3377)

ExtractCookieFields (see page 3363)

TStrings

Names

Values

13.1.20.5.14 TWebRequest.Date Property

Reports the value of the Date header of the HTTP request message.

Pascal

```
property Date: TDateTime;
```

C++

```
__property TDateTime Date;
```

Description

Read Date to determine the date and time at which the HTTP request message originated. Date can be used to determine whether the Content (see page 3373) of the request is still valid. The value of Date is translated into local time as specified by the system clock from the time units set by the Web client.

For example, if an HTTP request message specifies a MethodType (see page 3382) of mtPost, and Date is older than the most recent values of the fields that are being posted, the server application may return a status code that indicates an error rather than overwriting more recent values.

Date is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 3379)

IfModifiedSince (see page 3380)

MethodType (see page 3382)

StatusCode (see page 3413)

13.1.20.5.15 TWebRequest.DerivedFrom Property

Reports the value of the Derived-From (see page 3379) header of the HTTP request message.

Pascal

```
property DerivedFrom: AnsiString;
```

C++

```
__property AnsiString DerivedFrom;
```

Description

Read DerivedFrom to determine the URI from which the value of Content (see page 3373) originated. DerivedFrom is only meaningful when the request includes a Content (see page 3373) property and the Content (see page 3373) did not originate with the Web client.

See Also

Content (see page 3373)

Referer (see page 3385)

Properties That Describe the Web Client

13.1.20.5.16 TWebRequest.Expires Property

Reports the value of the Expires header of the HTTP request message.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Use Expires to determine whether the request is "stale." Expires is the date and time at which the request becomes invalid. The value of Expires is translated into local time as specified by the system clock from the time units set by the Web client.

If the request has a CacheControl (see page 3372) value with a max-age directive, the max-age directive overrides the value of Expires.

Expires is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Date (see page 3378)

IfModifiedSince (see page 3380)

CacheControl (see page 3372)

13.1.20.5.17 TWebRequest.Files Property

Provides access to the files uploaded in the HTTP request.

Pascal

```
property Files: TAbstractWebRequestFiles;
```

C++

```
__property TAbstractWebRequestFiles Files;
```

Description

Use this property to find the specific instance of the object's TAbstractRequestFiles object. The TAbstractRequestFiles object contains uploaded files extracted from the HTTP request.

13.1.20.5.18 TWebRequest.From Property

Reports the value of the From header of the HTTP request message.

Pascal

```
property From: AnsiString;
```

C++

```
__property AnsiString From;
```

Description

Read From to determine the Internet e-mail address of the human user for whom the Web client sent the HTTP request message. This value can be used for logging purposes, or to contact the originator of the request if problems occur.

See Also

RemoteAddr (see page 3386)

RemoteHost (see page 3386)

UserAgent (see page 3388)

Properties That Describe the Web Client

13.1.20.5.19 TWebRequest.Host Property

Reports the value of the Host header of the HTTP request message.

Pascal

```
property Host: AnsiString;
```

C++

```
__property AnsiString Host;
```

Description

Read Host to determine the Internet host and port number of resource being requested. For example, if the client request specifies the following URL (see page 3388):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

The value of Host would be

`www.TSite.com`

The value of Host can be used to differentiate between internally ambiguous URLs, such as the root "/" URL (see page 3388) of a server for multiple host names on a single IP address.

See Also

PathInfo (see page 3382)

ScriptName (see page 3386)

ServerPort (see page 3387)

URL (see page 3388)

Properties That Identify the Target

13.1.20.5.20 TWebRequest.IfModifiedSince Property

Reports the value of the If-Modified-Since header of the HTTP request message.

Pascal

```
property IfModifiedSince: TDateTime;
```

C++

```
__property TDateTime IfModifiedSince;
```

Description

Read IfModifiedSince on a request with a MethodType (see page 3382) of mtGet to determine whether the request is conditional. If the information requested by the Web client was not modified since the time indicated by IfModifiedSince, the response should be a not modified error rather than the information indicated by the URL (see page 3388) property.

The value of IfModifiedSince is given according to the local clock, not the clock of the Web client.

Note: On Windows, the value of the local clock is specified in the Windows Control Panel.

IfModifiedSince is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 3379)

Date (see page 3378)

Properties That Describe the Expected Response

13.1.20.5.21 TWebRequest.InternalPathInfo Property

Returns the path info portion of the URL (see page 3388).

Pascal

```
property InternalPathInfo: AnsiString;
```

C++

```
__property AnsiString InternalPathInfo;
```

Description

InternalPathInfo contains the path information portion of the URL (see page 3388), which identifies the destination of the message. This property generally contains the same information as PathInfo.To ensure cross-platform code compatibility, use InternalPathInfo instead of PathInfo (see page 3382).

Note: On Windows, this property varies slightly from PathInfo (see page 3382) if the application has an embedded test server. In that case, information specific to the test server is contained in the InternalScriptName (see page 3381) instead of the InternalPathInfo.

See Also

PathInfo (see page 3382)

13.1.20.5.22 TWebRequest.InternalScriptName Property

Returns the ScriptName (see page 3386) part of the URL (see page 3388).

Pascal

```
property InternalScriptName: AnsiString;
```

C++

```
__property AnsiString InternalScriptName;
```

Description

InternalScriptName contains the script name portion of the URL (see page 3388), which specifies the name of a Web server application. This property generally contains the same information as ScriptName (see page 3386). To ensure cross platform code compatibility, use InternalScriptName instead of ScriptName (see page 3386).

Note: On Windows, this property varies slightly from ScriptName (see page 3386) if the application has an embedded test server. In that case, information specific to the test server is contained in the InternalScriptName instead of the InternalPathInfo (see page 3381).

See Also

ScriptName (see page 3386)

13.1.20.5.23 TWebRequest.Method Property

Pascal

```
property Method: AnsiString;
```

C++

```
__property AnsiString Method;
```

Description

This is Method, a member of class TWebRequest.

13.1.20.5.24 TWebRequest.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class TWebRequest.

13.1.20.5.25 TWebRequest.PathInfo Property

Reports the value of the path information (if any) of the URL (see page 3388) specified in the HTTP request message.

Pascal

```
property PathInfo: AnsiString;
```

C++

```
__property AnsiString PathInfo;
```

Description

Read PathInfo to determine the path on the server machine where the target of the request is found. For example, given this URL (see page 3388):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the PathInfo portion is

`/mammals`

The TWebDispatcher (see page 3349) object uses the PathInfo to identify the appropriate TWebActionItem (see page 3335) for servicing the request message.

See Also

Host (see page 3380)

PathTranslated (see page 3383)

Query (see page 3384)

ScriptName (see page 3386)

TWebActionItem (see page 3335)

TWebDispatcher (see page 3349)

InternalPathInfo (see page 3381)

URL (see page 3388)

Dispatching Request Messages

Properties That Identify the Target

13.1.20.5.26 TWebRequest.PathTranslated Property

Represents a translation of the PathInfo (see page 3382) property to a fully qualified path on the Web server.

Pascal

```
property PathTranslated: AnsiString;
```

C++

```
__property AnsiString PathTranslated;
```

Description

Use PathTranslated to refer to the location of the target of the request in an HTTP response.

See Also

Host (see page 3380)

PathInfo (see page 3382)

TWebActionItem (see page 3335)

TWebResponse (see page 3389)

URL (see page 3388)

13.1.20.5.27 TWebRequest.ProtocolVersion Property

Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.

Pascal

```
property ProtocolVersion: AnsiString;
```

C++

```
__property AnsiString ProtocolVersion;
```

Description

Use ProtocolVersion to determine the protocol used by the Web client when generating the HTTP request message.

Some properties of the HTTP request may be interpreted differently depending on the ProtocolVersion. For example, if ProtocolVersion is 1.0, then the connection should not be considered persistent, even if the Connection (see page 3373) property does not specify the close option.

See Also

Connection (see page 3373)

13.1.20.5.28 TWebRequest.Query Property

Indicates the value of the query information (if any) of the URL (see page 3388) specified in the HTTP request message.

Pascal

```
property Query: AnsiString;
```

C++

```
__property AnsiString Query;
```

Description

Read Query to determine the details of a query specified in the URL (see page 3388) of the Web client request. For example, given this URL (see page 3388):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the Query portion is

`animal=dog&color=black`

Query is the unparsed query portion of the URL (see page 3388). The query may include multiple fields separated by ampersands. Use the QueryFields (see page 3384) property to work with the query on a field-by-field basis.

See Also

ExtractQueryFields (see page 3364)

Host (see page 3380)

PathInfo (see page 3382)

QueryFields (see page 3384)

ScriptName (see page 3386)

Properties That Identify the Target

13.1.20.5.29 TWebRequest.QueryFields Property

Represents the parsed contents of the Query (see page 3384) in the URL (see page 3388) of the HTTP request message.

Pascal

```
property QueryFields: TStrings;
```

C++

```
__property TStrings QueryFields;
```

Description

Use QueryFields to work with individual query values provided by the Web client request. The Query (see page 3384) property is parsed into separate fields by separating into substrings whenever an ampersand is encountered.

Each query field appears as a separate string in the QueryFields list with the form Name=Value. As a TStrings object,

QueryFields allows applications to manipulate the individual fields by name or value.

See Also

ExtractQueryFields (see page 3364)

Query (see page 3384)

TStrings

Names

Values

Properties That Identify the Target

13.1.20.5.30 TWebRequest.RawContent Property

Pascal

```
property RawContent: AnsiString;
```

C++

```
__property AnsiString RawContent;
```

Description

This is RawContent, a member of class TWebRequest.

13.1.20.5.31 TWebRequest.Referer Property

Reports the value of the Referer header of the HTTP request message.

Pascal

```
property Referer: AnsiString;
```

C++

```
__property AnsiString Referer;
```

Description

Read Referer to determine the URI of the resource where the HTTP request message originated. Referer allows the server application to generate back-links to the source of the request. If Referer is a partial URI, it should be interpreted as a relative path to the URL (see page 3388) property.

The Referer property is not set when the URL (see page 3388) of the HTTP request was obtained from a source that does not have its own URI, such as input from the user keyboard.

See Also

DerivedFrom (see page 3378)

From (see page 3379)

RemoteAddr (see page 3386)

RemoteHost (see page 3386)

UserAgent (see page 3388)

Properties That Describe the Web Client

13.1.20.5.32 TWebRequest.RemoteAddr Property

Indicates the remote IP address of the client associated with the HTTP request message.

Pascal

```
property RemoteAddr: AnsiString;
```

C++

```
__property AnsiString RemoteAddr;
```

Description

Read RemoteAddr to obtain the IP address of the source of the Web client request.

See Also

From (see page 3379)

RemoteHost (see page 3386)

UserAgent (see page 3388)

Properties That Describe the Web Client

13.1.20.5.33 TWebRequest.RemoteHost Property

Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.

Pascal

```
property RemoteHost: AnsiString;
```

C++

```
__property AnsiString RemoteHost;
```

Description

Use RemoteHost to compose URI values for back-links to the source of the request when the Referer (see page 3385) property is not specified. RemoteHost does not include any path information.

See Also

Referer (see page 3385)

RemoteAddr (see page 3386)

UserAgent (see page 3388)

Properties That Describe the Web Client

13.1.20.5.34 TWebRequest.ScriptName Property

Indicates the value of the script information (if any) of the URL (see page 3388) specified in the HTTP request message.

Pascal

```
property ScriptName: AnsiString;
```

C++

```
__property AnsiString ScriptName;
```

Description

Read ScriptName to determine the full path to the Web server application specified by the client request. For example, given this URL (see page 3388):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the ScriptName portion is:

`/art/gallery.cgi`

See Also

Host (see page 3380)

PathInfo (see page 3382)

Query (see page 3384)

URL (see page 3388)

Properties That Identify the Target

13.1.20.5.35 TWebRequest.ServerPort Property

Indicates the port number of the port on the Web server that receives the HTTP request message.

Pascal

```
property ServerPort: Integer;
```

C++

```
__property int ServerPort;
```

Description

Read ServerPort to determine the number of the port used in the connection to the Web server. The Host (see page 3380) property will include the ServerPort if it is specified as part of the URL (see page 3388) for the client request.

See Also

Host (see page 3380)

URL (see page 3388)

13.1.20.5.36 TWebRequest.Title Property

Reports the value of the Title header of the HTTP request message.

Pascal

```
property Title: AnsiString;
```

C++

```
__property AnsiString Title;
```

Description

Read Title to determine the Title of the information requested in the Web client request.

See Also

Cookie (see page 3377)

Title (see page 3414)

Properties That Describe the Content

13.1.20.5.37 TWebRequest.URL Property

Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.

Pascal

```
property URL: AnsiString;
```

C++

```
__property AnsiString URL;
```

Description

Use the URL property to identify the resource the Web client identifies as the target of the request message. URL may be an HTML page to return, a resource to be created or deleted, or an object that provides some other service for the Web client.

The TWebRequest (see page 3358) object parses the URL into its constituent parts, yielding the Host (see page 3380), ScriptName (see page 3386), PathInfo (see page 3382), and Query (see page 3384) properties.

Note: Usually the URL property is a Uniform Resource Locator (URL) that can be parsed into host, script, path, and query portions. However, it is possible for HTTP request messages to specify the target in more general terms, as a Uniform Resource Identifier (URI).

See Also

Host (see page 3380)

PathInfo (see page 3382)

Query (see page 3384)

ScriptName (see page 3386)

Properties That Identify the Target

13.1.20.5.38 TWebRequest.UserAgent Property

Reports the value of the User-Agent header of the HTTP request message.

Pascal

```
property UserAgent: AnsiString;
```

C++

```
__property AnsiString UserAgent;
```

Description

Read UserAgent to obtain information about the user agent originating the client request. Use the UserAgent information for statistical purposes, for tracing protocol violations, and for tailoring responses to known limitations of specific applications that generate client requests. The UserAgent string contains one or more descriptors, in order of significance. Each descriptor gives the name of the product or application that generated the request, or a comment about the user agent.

See Also

From (see page 3379)

Referer (see page 3385)

Properties That Describe the Web Client

13.1.21 TWebResponse Class

TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

Class Hierarchy

Pascal

```
TWebResponse = class(TObject);
```

C++

```
class TWebResponse : public TObject;
```

File

HTTPApp

Description

Use a descendant of TWebResponse to specify the response to an HTTP request message.

A web application creates a TWebResponse object based on the TWebRequest (see page 3358) object for an incoming HTTP request message. The Web dispatcher for the application then passes the TWebResponse object to the TWebActionItem (see page 3335) associated with the TWebRequest (see page 3358) object, so that the response can be formulated.

TWebResponse has descendants that represent HTTP response messages in different types of web server applications. These descendants are

TApacheResponse objects in Apache server applications.

TCGIResponse objects in CGI server applications.

TISAPIResponse objects in server DLLs (Windows only).

TWinCGIResponse objects in Windows-based CGI servers (Windows only).

See Also

TApacheResponse

TCGIResponse

TISAPIResponse (see page 4518)

TWebActionItem (see page 3335)

TWebApplication

TWebDispatcher (see page 3349)

TWebRequest (see page 3358)

TWinCGIResponse

Filling in the Response Header

Responding to Request Messages with Action Items

The Structure of a Web Broker Application

Dispatching Requests and Responses

13.1.21.1 TWebResponse Members

The following tables list the members exposed by TWebResponse.

Protected Fields

	Name	Description
	FCustomHeaders (see page 3392)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 3393)	This is FHTTPRequest, a member of class TWebResponse.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 3394)	Appends the custom headers specified by the CustomHeaders (see page 3408) property to the Headers parameter.
	FormatAuthenticate (see page 3394)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).
A	GetContent (see page 3395)	Provides the abstract (pure virtual) prototype of a method to access the Content (see page 3404) property.
A	GetDateVariable (see page 3395)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
A	GetIntegerVariable (see page 3396)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
A	GetLogMessage (see page 3396)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 3411) property.
A	GetStatusCode (see page 3396)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 3413) property.
A	GetStringVariable (see page 3397)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
A	SetContent (see page 3399)	Provides the abstract (pure virtual) prototype of a method to set the Content (see page 3404) property.
V	SetContentStream (see page 3400)	Sets the value of the ContentStream (see page 3406) property to the Value parameter.
A	SetDateVariable (see page 3401)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
A	SetIntegerVariable (see page 3401)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
A	SetLogMessage (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 3411) property.
A	SetStatusCode (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 3413) property.
A	SetStringVariable (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
	Create (see page 3391)	Creates an instance of Create to represent the response to the HTTPRequest (see page 3410) parameter.
V	Destroy (see page 3392)	Disposes of an object instance.
	GetCustomHeader (see page 3395)	Returns the value of the custom header with the specified name.
A	SendRedirect (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 3413) of 301 (Moved Permanently) and a Location (see page 3411) of URI.
A	SendResponse (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.
A	SendStream (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.
V	Sent (see page 3399)	Indicates whether the HTTP response message has already been sent.
	SetCookieField (see page 3400)	Adds a cookie header to the response message.
	SetCustomHeader (see page 3400)	Replaces the value of a specified custom header.

Public Properties

	Name	Description
	Allow (see page 3404)	Lists the HTTP request methods that the application can service.
	Content (see page 3404)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 3405)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 3405)	Specifies the length, in bytes, of the Content (see page 3404) property.
	ContentStream (see page 3406)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 3406)	Indicates the media type of the Content (see page 3404) of the HTTP response message.
	ContentVersion (see page 3407)	Represents the version associated with the Content (see page 3404) of the HTTP response message.
R	Cookies (see page 3407)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 3408)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 3408)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 3409)	Specifies the URI of the resource where the Content (see page 3404) of the HTTP response originated.
	Expires (see page 3409)	Specifies the date and time after which the Content (see page 3404) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 3410)	Represents the HTTP request message for which this is a response.
	LastModified (see page 3410)	Indicates the date and time that the value of the Content (see page 3404) property was last updated.
	Location (see page 3411)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 3411)	Describes the HTTP response message for event logging purposes.
	RawContent (see page 3411)	This is RawContent, a member of class TWebResponse.
	Realm (see page 3411)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 3412)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 3412)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 3413)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 3414)	Specifies a title for the Content (see page 3404) of the HTTP response message.
	Version (see page 3415)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 3415)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).

13.1.21.2 TWebResponse.Create Constructor

Creates an instance of Create to represent the response to the HTTPRequest (see page 3410) parameter.

Pascal

```
constructor Create(HTTPRequest: TWebRequest);
```

C++

```
__fastcall TWebResponse(TWebRequest HTTPRequest);
```

Description

Applications shouldn't need to call Create to instantiate a response object. The TWebApplication object creates a Create for each TWebRequest (see page 3358) it creates.

See Also

TWebApplication

TWebRequest (see page 3358)

13.1.21.3 TWebResponse.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebResponse();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

13.1.21.4 TWebResponse Fields

The fields of the TWebResponse class are listed here.

Protected Fields

	Name	Description
	FCustomHeaders (see page 3392)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 3393)	This is FHTTPRequest, a member of class TWebResponse.

13.1.21.4.1 TWebResponse.FCustomHeaders Field

Pascal

```
FCustomHeaders: TStrings;
```

C++

```
TStrings FCustomHeaders;
```

Description

This is FCustomHeaders, a member of class TWebResponse.

13.1.21.4.2 TWebResponse.FHTTPRequest Field

Pascal

```
FHTTPRequest: TWebRequest;
```

C++

```
TWebRequest FHTTPRequest;
```

Description

This is FHTTPRequest, a member of class TWebResponse.

13.1.21.5 TWebResponse Methods

The methods of the TWebResponse class are listed here.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 3394)	Appends the custom headers specified by the CustomHeaders (see page 3408) property to the Headers parameter.
	FormatAuthenticate (see page 3394)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).
A	GetContent (see page 3395)	Provides the abstract (pure virtual) prototype of a method to access the Content (see page 3404) property.
A	GetDateVariable (see page 3395)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
A	GetIntegerVariable (see page 3396)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
A	GetLogMessage (see page 3396)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 3411) property.
A	GetStatusCode (see page 3396)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 3413) property.
A	GetStringVariable (see page 3397)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
A	SetContent (see page 3399)	Provides the abstract (pure virtual) prototype of a method to set the Content (see page 3404) property.
V	SetContentStream (see page 3400)	Sets the value of the ContentStream (see page 3406) property to the Value parameter.
A	SetDateVariable (see page 3401)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
A	SetIntegerVariable (see page 3401)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
A	SetLogMessage (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 3411) property.
A	SetStatusCode (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 3413) property.
A	SetStringVariable (see page 3402)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
	GetCustomHeader (see page 3395)	Returns the value of the custom header with the specified name.
	SendRedirect (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 3413) of 301 (Moved Permanently) and a Location (see page 3411) of URI.
	SendResponse (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.
	SendStream (see page 3398)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.
	Sent (see page 3399)	Indicates whether the HTTP response message has already been sent.
	SetCookieField (see page 3400)	Adds a cookie header to the response message.
	SetCustomHeader (see page 3400)	Replaces the value of a specified custom header.

13.1.21.5.1 TWebResponse.AddCustomHeaders Method

Appends the custom headers specified by the CustomHeaders (see page 3408) property to the Headers parameter.

Pascal

```
procedure AddCustomHeaders(var Headers: string);
```

C++

```
__fastcall AddCustomHeaders(UnicodeString Headers);
```

Description

Call AddCustomHeaders to add the message headers described by the CustomHeaders (see page 3408) property to a string of message headers. Each header is added on its own line (separated by carriage return and line feed), in the format HeaderName: Value.

See Also

CustomHeaders (see page 3408)

GetCustomHeader (see page 3395)

13.1.21.5.2 TWebResponse.FormatAuthenticate Method

Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).

Pascal

```
function FormatAuthenticate: AnsiString;
```

C++

```
__fastcall AnsiString FormatAuthenticate();
```

Description

FormatAuthenticate returns the value of WWWAuthenticate (see page 3415) if Realm (see page 3411) is an empty string. If it is not empty, it returns a name/value pair describing the realm of the WWWAuthenticate (see page 3415) property.

See Also

WWWAuthenticate (see page 3415)

Realm (see page 3411)

13.1.21.5.3 TWebResponse.GetContent Method

Provides the abstract (pure virtual) prototype of a method to access the Content (see page 3404) property.

Pascal

```
function GetContent: AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetContent() = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement GetContent to provide an access method for their internal representation of the Content (see page 3404) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the Content (see page 3404) property differently.

See Also

Content (see page 3404)

SetContent (see page 3399)

13.1.21.5.4 TWebResponse.GetCustomHeader Method

Returns the value of the custom header with the specified name.

Pascal

```
function GetCustomHeader(const Name: string): String;
```

C++

```
__fastcall UnicodeString GetCustomHeader(const UnicodeString Name);
```

Description

Call GetCustomHeader to obtain the value of one of the custom headers specified in the CustomHeaders (see page 3408) property. The Name parameter specifies the name of the HTTP message header.

See Also

CustomHeaders (see page 3408)

SetCustomHeader (see page 3400)

13.1.21.5.5 TWebResponse.GetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; virtual; abstract;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement GetDateVariable to provide a method by which date-time values are read from the internal table used to store HTTP header variables. The response object uses GetDateVariable to access the properties that represent dates.

See Also

Date (see page 3408)

Expires (see page 3409)

LastModified (see page 3410)

SetDateVariable (see page 3401)

13.1.21.5.6 TWebResponse.GetIntegerVariable Method

Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement GetIntegerVariable to provide a method by which integer values are read from the internal table used to store HTTP header variables. The response object uses GetIntegerVariable to access the properties that represent integers.

See Also

ContentLength (see page 3405)

SetIntegerVariable (see page 3401)

13.1.21.5.7 TWebResponse.GetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 3411) property.

Pascal

```
function GetLogMessage: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetLogMessage() = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement GetLogMessage to provide an access method for internal representation of the LogMessage (see page 3411) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage (see page 3411) property differently.

See Also

LogMessage (see page 3411)

SetLogMessage (see page 3402)

13.1.21.5.8 TWebResponse.GetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 3413) property.

Pascal

```
function GetStatusCode: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetStatusCode() = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement GetStatusCode to provide an access method for internal representation of the StatusCode (see page 3413) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode (see page 3413) property differently.

See Also

StatusCode (see page 3413)

SetStatusCode (see page 3402)

13.1.21.5.9 TWebResponse.GetStringVariable Method

Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): AnsiString; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index) = 0;
```

Description

GetStringVariable provides the abstract (pure virtual) prototype of a method to extract a string value from an indexed table of HTTP header variables. Descendants of TWebResponse (see page 3389) implement GetStringVariable to provide a method by which the string values are read from the internal table used to store HTTP header variables. The response object uses GetStringVariable to access the properties that represent strings.

See Also

Allow (see page 3404)

ContentEncoding (see page 3405)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

Location (see page 3411)

Realm (see page 3411)

ReasonString (see page 3412)

Server (see page 3412)

SetStringVariable (see page 3402)

Title (see page 3414)

Version (see page 3415)

WWWAuthenticate (see page 3415)

13.1.21.5.10 TWebResponse.SendRedirect Method

Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 3413) of 301 (Moved Permanently) and a Location (see page 3411) of URI.

Pascal

```
procedure SendRedirect(const URI: AnsiString); virtual; abstract;
```

C++

```
virtual __fastcall SendRedirect(const AnsiString URI) = 0;
```

Description

Call SendRedirect to redirect the web client request to another URI that is the proper target of the request. SendRedirect does the same thing as setting the StatusCode (see page 3413) to 301, the Location (see page 3411) to URI, and calling SendResponse (see page 3398).

See Also

Location (see page 3411)

SendResponse (see page 3398)

StatusCode (see page 3413)

Sending the Response

13.1.21.5.11 TWebResponse.SendResponse Method

Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.

Pascal

```
procedure SendResponse; virtual; abstract;
```

C++

```
virtual __fastcall SendResponse() = 0;
```

Description

Call SendResponse after setting the properties of the HTTP response object to specify the response that should be made to the web client. If the request should be redirected to another URI, it is more efficient to call SendRedirect (see page 3398) instead.

See Also

SendRedirect (see page 3398)

Sending the Response

13.1.21.5.12 TWebResponse.SendStream Method

Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.

Pascal

```
procedure SendStream(AStream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall SendStream(TStream AStream) = 0;
```

Description

Only call SendStream after first calling SendResponse (see page 3398). Unlike SendStream, SendResponse (see page 3398) fills the header of the HTTP response message, based on the properties of the TWebResponse (see page 3389) object. If SendResponse (see page 3398) has not been called first to set the header of the HTTP response message, set the ContentStream (see page 3406) property to AStream, and call SendResponse (see page 3398).

See Also

ContentStream (see page 3406)

SendResponse (see page 3398)

Sent (see page 3399)

13.1.21.5.13 TWebResponse.Sent Method

Indicates whether the HTTP response message has already been sent.

Pascal

```
function Sent: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Sent();
```

Description

Because SendRedirect (see page 3398) and SendStream (see page 3398) are abstract (pure virtual) methods for TWebResponse (see page 3389), Sent always returns false. Descendants of TWebResponse (see page 3389) override Sent to check whether the message was sent and return true or false as appropriate.

TWebApplication objects read Sent and send unsent responses to the web client.

See Also

SendRedirect (see page 3398)

SendStream (see page 3398)

TWebApplication

13.1.21.5.14 TWebResponse.SetContent Method

Provides the abstract (pure virtual) prototype of a method to set the Content (see page 3404) property.

Pascal

```
procedure SetContent(const Value: AnsiString); virtual; abstract;
```

C++

```
virtual __fastcall SetContent(const AnsiString Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetContent to provide a method for setting the Content (see page 3404) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications can store the value of the Content (see page 3404) property differently.

See Also

Content (see page 3404)

GetContent (see page 3395)

13.1.21.5.15 TWebResponse.SetContentStream Method

Sets the value of the ContentStream (see page 3406) property to the Value parameter.

Pascal

```
procedure SetContentStream(Value: TStream); virtual;
```

C++

```
virtual __fastcall SetContentStream(TStream Value);
```

Description

SetContentStream frees the previous value of ContentStream (see page 3406) and replaces the property with the Value parameter.

See Also

ContentStream (see page 3406)

13.1.21.5.16 TWebResponse.SetCookieField Method

Adds a cookie header to the response message.

Pascal

```
procedure SetCookieField(Values: TStrings; const ADomain: AnsiString; const APath: AnsiString;
AExpires: TDateTime; ASecure: Boolean);
```

C++

```
__fastcall SetCookieField(TStrings Values, const AnsiString ADomain, const AnsiString APath,
TDateTime AExpires, Boolean ASecure);
```

Description

Call SetCookieField to add a cookie header to the Cookies (see page 3407) property. Each cookie header contains a set of name/value pairs that can be passed on by the client application. Use the Values parameter to specify the name/value pairs as strings of the form Name=Value. Use the ADomain and APath parameters to indicate which URLs the cookie should be sent to. Use the AExpires parameter to indicate when the cookie is no longer valid. Use the ASecure property to indicate whether the cookie should only be passed on by the client if a secure connection is used.

See the documentation on the Netscape Web site for more information about cookies.

See Also

Cookies (see page 3407)

13.1.21.5.17 TWebResponse.SetCustomHeader Method

Replaces the value of a specified custom header.

Pascal

```
procedure SetCustomHeader(const Name: string; const Value: string);
```

C++

```
__fastcall SetCustomHeader(const UnicodeString Name, const UnicodeString Value);
```

Description

Call SetCustomHeader to add a new custom header to the list maintained by the CustomHeaders (see page 3408) property, or to change the value of one of the headers already in the list. SetCustomHeader locates the header in the list with the name specified by the Name parameter. If a header with that name is in the list, SetCustomHeader replaces the value of that header with the value specified as the Value parameter. If the name was not found, SetCustomHeader adds a new header to the list with the given name and value.

See Also

CustomHeaders (see page 3408)

GetCustomHeader (see page 3395)

13.1.21.5.18 TWebResponse.SetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.

Pascal

```
procedure SetDateVariable(Index: Integer; const Value: TDateTime); virtual; abstract;
```

C++

```
virtual __fastcall SetDateVariable(int Index, const TDateTime Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetDateVariable to provide a method by which date-time values are written to the internal table used to store HTTP header variables. The response object uses SetDateVariable to set the properties that represent dates.

See Also

Date (see page 3408)

Expires (see page 3409)

GetDateVariable (see page 3395)

LastModified (see page 3410)

13.1.21.5.19 TWebResponse.SetIntegerVariable Method

Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.

Pascal

```
procedure SetIntegerVariable(Index: Integer; Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetIntegerVariable(int Index, int Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetIntegerVariable to provide a method by which integer values are written to the internal table used to store HTTP header variables. The response object uses SetIntegerVariable to set the properties that represent integers.

See Also

ContentLength (see page 3405)

GetIntegerVariable (see page 3396)

13.1.21.5.20 TWebResponse.SetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 3411) property.

Pascal

```
procedure SetLogMessage(const Value: string); virtual; abstract;
```

C++

```
virtual __fastcall SetLogMessage(const UnicodeString Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetLogMessage to provide a method to set the internal representation of the LogMessage (see page 3411) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage (see page 3411) property differently.

See Also

GetLogMessage (see page 3396)

LogMessage (see page 3411)

13.1.21.5.21 TWebResponse.SetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 3413) property.

Pascal

```
procedure SetStatusCode(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetStatusCode(int Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetStatusCode to provide a method to set the internal representation of the StatusCode (see page 3413) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode (see page 3413) property differently.

See Also

GetStatusCode (see page 3396)

StatusCode (see page 3413)

13.1.21.5.22 TWebResponse.SetStringVariable Method

Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Pascal

```
procedure SetStringVariable(Index: Integer; const Value: AnsiString); virtual; abstract;
```

C++

```
virtual __fastcall SetStringVariable(int Index, const AnsiString Value) = 0;
```

Description

Descendants of TWebResponse (see page 3389) implement SetStringVariable to provide a method by which string values are written to the internal table used to store HTTP header variables. The response object uses SetStringVariable to set the properties that represent strings.

See Also

Allow (see page 3404)

ContentEncoding (see page 3405)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

GetStringVariable (see page 3397)

Location (see page 3411)

Realm (see page 3411)

ReasonString (see page 3412)

Server (see page 3412)

Title (see page 3414)

Version (see page 3415)

WWWAuthenticate (see page 3415)

13.1.21.6 TWebResponse Properties

The properties of the TWebResponse class are listed here.

Public Properties

	Name	Description
	Allow (see page 3404)	Lists the HTTP request methods that the application can service.
	Content (see page 3404)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 3405)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 3405)	Specifies the length, in bytes, of the Content (see page 3404) property.
	ContentStream (see page 3406)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 3406)	Indicates the media type of the Content (see page 3404) of the HTTP response message.
	ContentVersion (see page 3407)	Represents the version associated with the Content (see page 3404) of the HTTP response message.
R	Cookies (see page 3407)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 3408)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 3408)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 3409)	Specifies the URI of the resource where the Content (see page 3404) of the HTTP response originated.
	Expires (see page 3409)	Specifies the date and time after which the Content (see page 3404) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 3410)	Represents the HTTP request message for which this is a response.
	LastModified (see page 3410)	Indicates the date and time that the value of the Content (see page 3404) property was last updated.
	Location (see page 3411)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 3411)	Describes the HTTP response message for event logging purposes.
	RawContent (see page 3411)	This is RawContent, a member of class TWebResponse.

	Realm (see page 3411)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 3412)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 3412)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 3413)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 3414)	Specifies a title for the Content (see page 3404) of the HTTP response message.
	Version (see page 3415)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 3415)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).

13.1.21.6.1 TWebResponse.Allow Property

Lists the HTTP request methods that the application can service.

Pascal

```
property Allow: AnsiString;
```

C++

```
__property AnsiString Allow;
```

Description

Use Allow to indicate what type of request methods the server application can respond to for the URI specified by HTTPRequest (see page 3410). Allow is a string of method names, separated by commas. The method names can include any pre-defined methods from the HTTP standard, or any additional methods implemented by the server application. At a minimum, the TWebResponse (see page 3389) object must allow GET and HEAD requests.

See Also

HTTPRequest (see page 3410)

TWebRequest (see page 3358)

URL (see page 3388)

Describing the Server Application

13.1.21.6.2 TWebResponse.Content Property

Contains the information sent to the web client in response to an HTTP request message.

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

Use Content to send requested information to the web client. Not every HTTP request message requires content to be returned. For example, responses to requests with a Method of OPTIONS, HEAD, POST, PUT, DELETE, or TRACE should not include any Content. In all these cases, the response information is contained entirely in the response header, which is built from the other properties of the TWebResponse (see page 3389) object.

The response content can be a string of HTML commands, the name of an HTML file, graphic content such as a bit-stream, or any other MIME content type.

Note: If the ContentStream (see page 3406) property is set, it supersedes the value of the Content property. If both Content and ContentStream (see page 3406) are set, the web client is sent the value of the ContentStream (see page 3406), not the value of the Content property.

See Also

ContentEncoding (see page 3405)

ContentLength (see page 3405)

ContentStream (see page 3406)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

Expires (see page 3409)

TWebRequest (see page 3358)

Responding to Request Messages with Action Items

Setting the Response Content

13.1.21.6.3 TWebResponse.ContentEncoding Property

Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.

Pascal

```
property ContentEncoding: AnsiString;
```

C++

```
__property AnsiString ContentEncoding;
```

Description

Set ContentEncoding when the value of Content (see page 3404) has been encoded before it is sent to the web client. The encoding mechanisms are listed in the order in which they are applied, separated by commas. Typically, ContentEncoding indicates a compression method or an encoding method used to transfer extended-ASCII characters over the net.

See Also

Content (see page 3404)

ContentType (see page 3406)

Describing the Content

13.1.21.6.4 TWebResponse.ContentLength Property

Specifies the length, in bytes, of the Content (see page 3404) property.

Pascal

```
property ContentLength: Integer;
```

C++

```
__property int ContentLength;
```

Description

Set ContentLength to the length of the Content (see page 3404) property, if it is known. If the length of the Content (see page 3404) property is unknown, or if the response message does not include any content, set ContentLength to 0. Web clients use the ContentLength when reading the contents of the HTTP response message.

If the contents of the response message are taken from a stream, the value of ContentLength is set from the stream.

See Also

Content (see page 3404)

ContentStream (see page 3406)

Describing the Content

13.1.21.6.5 TWebResponse.ContentStream Property

Specifies the stream object for writing the contents of the HTTP response message to the web client.

Pascal

```
property ContentStream: TStream;
```

C++

```
__property TStream ContentStream;
```

Description

Use ContentStream to set the contents of the HTTP response message to a value that can be written from a stream. ContentStream allows the response object to write its contents directly from a stream.

The stream specified by ContentStream can access a string of HTML commands, the contents of a BLOB database field, or any MIME content type.

Note: ContentStream supersedes the Content (see page 3404) property. If both Content (see page 3404) and ContentStream are set, the web client is sent the value of the ContentStream, not the value of the Content (see page 3404) property.

See Also

Content (see page 3404)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

Expires (see page 3409)

Setting the Response Content

13.1.21.6.6 TWebResponse.ContentType Property

Indicates the media type of the Content (see page 3404) of the HTTP response message.

Pascal

```
property ContentType: AnsiString;
```

C++

```
__property AnsiString ContentType;
```

Description

Set ContentType to indicate the media type of the Content (see page 3404) of the HTTP response message. The value of ContentType must specify a valid MIME content type. The ContentType string has the form Type/SubType.

For example, if the HTTP response message sends a string of HTML commands to the web client in response to a GET request, the ContentType is text/html.

Note: If the HTTP request message includes an Accept header, ContentType should be one of the content types listed in the Accept property of TWebRequest (see page 3358).

See Also

Content (see page 3404)

Accept (see page 3371)

Describing the Content

13.1.21.6.7 TWebResponse.ContentVersion Property

Represents the version associated with the Content (see page 3404) of the HTTP response message.

Pascal

```
property ContentVersion: AnsiString;
```

C++

```
__property AnsiString ContentVersion;
```

Description

Use ContentVersion to specify a version associated with the Content (see page 3404) of the HTTP response. The meaning of ContentVersion depends on the type of information that is sent in the HTTP response. For example, if the Content (see page 3404) of the HTTP response has a ContentType (see page 3406) of text/html, ContentVersion would be the version of HTML used.

Server (see page 3412) applications that introduce meanings for the ContentVersion property should document the interpretation they assign to ContentVersion.

See Also

Content (see page 3404)

ContentType (see page 3406)

Describing the Content

13.1.21.6.8 TWebResponse.Cookies Property

Lists all cookie headers that should be sent with the response.

Pascal

```
property Cookies: TCookieCollection;
```

C++

```
__property TCookieCollection Cookies;
```

Description

Use Cookies to obtain direct access to the individual TCookie (see page 3304) objects that represent the fields of the cookie headers in the response message. Each TCookie (see page 3304) object represents in individual cookie header that is

included with the response message.

Use the Add method of the cookie collection to add a single cookie to the collection, or use the SetCookieField (see page 3400) method to add multiple related cookie headers to this collection. SetCookieField (see page 3400) creates and adds new cookies that share a common set of values for the domain, path, expires, and secure fields. Use the HeaderValue property of the individual entries in Cookies to determine the actual response header string that is sent for each cookie header.

Note: For more information about the use of cookies, see the documentation on Netscape's Web site.

See Also

SetCookieField (see page 3400)

Cookie (see page 3377)

TCookie (see page 3304)

Describing the Server Application

13.1.21.6.9 TWebResponse.CustomHeaders Property

Lists message header strings that are added to the response message before it is sent.

Pascal

```
property CustomHeaders: TStrings;
```

C++

```
__property TStrings CustomHeaders;
```

Description

Use CustomHeaders to add header information to the response that can not be specified using the other properties of TWebResponse (see page 3389). Each string in the CustomHeaders property has the form HeaderName=Value, where HeaderName specifies the name of the header, and Value is the value for that header, formatted as a string.

Note: Setting the CustomHeaders property assigns a set of strings to the custom headers list. It does not replace the TStrings object that is the value of CustomHeaders.

Warning: CustomHeaders includes at most one value for each HeaderName. If the HTTP message includes multiple headers with the same name, only the first header's value will appear in CustomHeaders.

See Also

GetCustomHeader (see page 3395)

SetCustomHeader (see page 3400)

TStrings

Assign

13.1.21.6.10 TWebResponse.Date Property

Indicates the time and date when the HTTP response message is sent to the web client.

Pascal

```
property Date: TDateTime;
```

C++

```
__property TDateTime Date;
```

Description

Set Date to the current date and time immediately before sending the HTTP response message to the web client. The Web client uses Date to determine the age of the information returned.

Date is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 3409)

LastModified (see page 3410)

13.1.21.6.11 TWebResponse.DerivedFrom Property

Specifies the URI of the resource where the Content (see page 3404) of the HTTP response originated.

Pascal

```
property DerivedFrom: AnsiString;
```

C++

```
__property AnsiString DerivedFrom;
```

Description

Set DerivedFrom to the URI where the value of Content (see page 3404) originates. It is especially important to set DerivedFrom when the Content (see page 3404) of the HTTP response did not originate from the resource indicated by the URL property of the request object.

See Also

Content (see page 3404)

URL (see page 3388)

Describing the Content

13.1.21.6.12 TWebResponse.Expires Property

Specifies the date and time after which the Content (see page 3404) of the HTTP response message should be considered invalid.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Set Expires when the Content (see page 3404) of the HTTP response message contains information that may become invalid with the passage of time. Setting an Expires value does not imply that the information or its source will necessarily change or cease to exist when the Expires date and time is reached. Web clients are simply warned to check the freshness of the information after the Expires date and time.

The value of Expires is specified in Greenwich Mean Time.

Expires is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Date (see page 3408)

LastModified (see page 3410)

Describing the Content

13.1.21.6.13 TWebResponse.HTTPRequest Property

Represents the HTTP request message for which this is a response.

Pascal

```
property HTTPRequest: TWebRequest;
```

C++

```
__property TWebRequest HTTPRequest;
```

Description

Read HTTPRequest to determine the header and contents of the HTTP request message that spawned this response message. The properties of HTTPRequest give access to the information sent in the web client request message.

See Also

TWebRequest (see page 3358)

13.1.21.6.14 TWebResponse.LastModified Property

Indicates the date and time that the value of the Content (see page 3404) property was last updated.

Pascal

```
property LastModified: TDateTime;
```

C++

```
__property TDateTime LastModified;
```

Description

Set LastModified to indicate the last time the Content (see page 3404) of the response was modified. This allows web clients to determine whether their local version of that information is up-to-date. When the HTTP request is a HEAD message, this may allow the web client and server to skip a potentially expensive transfer of information.

The value of LastModified is specified in the local time of the machine. The HTTP response message translates this value to the date encoding used by HTTP.

LastModified is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 3409)

Date (see page 3408)

IfModifiedSince (see page 3380)

Describing the Content

13.1.21.6.15 TWebResponse.Location Property

Specifies the URI of a resource the web client should point to for the completion of its request.

Pascal

```
property Location: AnsiString;
```

C++

```
__property AnsiString Location;
```

Description

Set Location when the TWebResponse (see page 3389) created a new resource in response to the HTTP request message, or when servicing of the request must be delegated to another URI than the one specified in the HTTP request message.

See Also

URL (see page 3388)

Indicating the Need for Client Action

13.1.21.6.16 TWebResponse.LogMessage Property

Describes the HTTP response message for event logging purposes.

Pascal

```
property LogMessage: string;
```

C++

```
__property UnicodeString LogMessage;
```

Description

Set LogMessage to a description of the action carried out by the TWebActionItem (see page 3335) object that services this HTTP response. Usually, LogMessage includes the URI that handled the request message, along with date and time information.

See Also

TWebActionItem (see page 3335)

TWebRequest (see page 3358)

13.1.21.6.17 TWebResponse.RawContent Property

Pascal

```
property RawContent: AnsiString;
```

C++

```
__property AnsiString RawContent;
```

Description

This is RawContent, a member of class TWebResponse.

13.1.21.6.18 TWebResponse.Realm Property

Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.

Pascal

```
property Realm: AnsiString;
```

C++

```
__property AnsiString Realm;
```

Description

Set Realm when the response message returns 401 (Unauthorized) or 402 (Payment required). Realm is the name of a set of URIs that require a password or payment before the web client can access them. All the URIs in the realm, including the URL specified in the request message, are made available as soon as the Web client provides the password or payment for the realm. The server application should keep track, internally, of the passwords or payments the client has provided, using the client information provided by the TWebRequest (see page 3358) object.

See Also

StatusCode (see page 3413)

TWebRequest (see page 3358)

URL (see page 3388)

WWWAuthenticate (see page 3415)

13.1.21.6.19 TWebResponse.ReasonString Property

Indicates the status of the response to the web client request as a human-readable string.

Pascal

```
property ReasonString: AnsiString;
```

C++

```
__property AnsiString ReasonString;
```

Description

Set ReasonString to inform the human web client of the success or failure of the HTTP request message. Each reason string is associated with a 3-digit status code whose value indicates the status of the response. The HTTP standard defines a number of stock status codes and corresponding reason strings. In addition, server applications can define new reason strings and associate them with unused status codes.

See Also

StatusCode (see page 3413)

Indicating the Response Status

13.1.21.6.20 TWebResponse.Server Property

Provides information about the software used by the server application to handle the request.

Pascal

```
property Server: AnsiString;
```

C++

```
__property AnsiString Server;
```

Description

Set Server to a string giving the name and version number of the application that provides the Content (see page 3404) of the

response message. The value of Server can contain multiple product identifiers and comments, listed in order of significance and separated by spaces. Product identifiers take the form ProductName/VersionNumber.

See Also

Content (see page 3404)

UserAgent (see page 3388)

Describing the Server Application

13.1.21.6.21 TWebResponse.StatusCode Property

Indicates the status of the response to the HTTP request message as a 3-digit integer.

Pascal

```
property StatusCode: Integer;
```

C++

```
__property int StatusCode;
```

Description

Set StatusCode to inform the web client the status of the HTTP request message. Status codes are standard return values that can be interpreted by all web clients. Each status code has a corresponding reason string that provides a human web client with a more understandable indication of the status.

The following tables lists the standard status codes and their corresponding reason strings.

StatusCode	ReasonString (see page 3412)
100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content (see page 3404)
205	Reset Content (see page 3404)
206	Partial Content (see page 3404)
300	Multiple Choices
301	Moved Permanently
302	Moved Temporarily
303	See Other
304	Not Modified
305	Use Proxy
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden

404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
500	Internal Server (see page 3412) Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version (see page 3415) not supported

Status codes not in the table can be created by the server application. When creating a new status code, the most significant digit should indicate the class of response, as follows:

1xx: Informational (Request received, continuing process)

2xx: Success (Request received, understood, and accepted)

3xx: Redirection (Further action needed to complete the request)

4xx: Client Error (Request contains bad syntax or cannot be serviced)

5xx: Server (see page 3412) Error (Server (see page 3412) failed to handle a valid request)

See Also

ReasonString (see page 3412)

Indicating the Response Status

13.1.21.6.22 TWebResponse.Title Property

Specifies a title for the Content (see page 3404) of the HTTP response message.

Pascal

```
property Title: AnsiString;
```

C++

```
__property AnsiString Title;
```

Description

Set the Title property to add a Title header to the HTTP response message. Title is an optional application-specific title

associated with the information included in the message.

See Also

Title (see page 3387)

Describing the Content

13.1.21.6.23 TWebResponse.Version Property

Specifies the HTTP protocol version number that the web client is using.

Pascal

```
property Version: AnsiString;
```

C++

```
__property AnsiString Version;
```

Description

Read Version to ensure that the properties set in the TWebResponse (see page 3389) object will be understood by the Web client. The value of Version is taken from the ProtocolVersion property of the TWebRequest (see page 3358) object.

For more information on the different HTTP protocol versions, see the Internet standards documentation.

See Also

ProtocolVersion (see page 3383)

13.1.21.6.24 TWebResponse.WWWAuthenticate Property

Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).

Pascal

```
property WWWAuthenticate: AnsiString;
```

C++

```
__property AnsiString WWWAuthenticate;
```

Description

Set WWWAuthenticate when the HTTP request fails because it requires access to a URI in a realm for which the Web client has not provided a password. WWWAuthenticate consists of an authentication scheme name, followed by a comma-separated list of parameter/value pairs that specify the parameters to the authentication scheme. The first parameter/value pair indicates the realm.

For more information about HTTP authentication schemes, see the Internet standards documents.

See Also

Realm (see page 3411)

StatusCode (see page 3413)

Indicating the Need for Client Action

13.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IGetAppDispatcher (see page 3416)	This is class HTTPApp.IGetAppDispatcher.
	IGetWebAppServices (see page 3417)	This is class HTTPApp.IGetWebAppServices.
	IGetWebRequestHandler (see page 3418)	This is class HTTPApp.IGetWebRequestHandler.
	IMultiModuleSupport (see page 3419)	This is class HTTPApp.IMultiModuleSupport.
	IProduceContent (see page 3420)	IProduceContent is the interface for generating the content of a Web page.
	IProduceContentFrom (see page 3422)	This is class HTTPApp.IProduceContentFrom.
	ISetAppDispatcher (see page 3423)	This is class HTTPApp.ISetAppDispatcher.
	IWebAppServices (see page 3424)	IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.
	IWebDispatch (see page 3427)	
	IWebDispatchActions (see page 3430)	IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.
	IWebDispatcherAccess (see page 3431)	IWebDispatcherAccess provides access to the current HTTP request and response object.
	IWebExceptionHandler (see page 3433)	This is class HTTPApp.IWebExceptionHandler.
	IWebRequestHandler (see page 3434)	IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

13.2.1 IGetAppDispatcher Interface

Class Hierarchy

Pascal

```
IGetAppDispatcher = interface;
```

C++

```
__interface IGetAppDispatcher;
```

File

HTTPApp

Description

This is class HTTPApp.IGetAppDispatcher.

13.2.1.1 IGetAppDispatcher Members

The following tables list the members exposed by IGetAppDispatcher.

Public Methods

	Name	Description
	GetAppDispatcher (see page 3417)	This is GetAppDispatcher, a member of class IGetAppDispatcher.

13.2.1.2 IGetAppDispatcher Methods

The methods of the IGetAppDispatcher class are listed here.

Public Methods

	Name	Description
	GetAppDispatcher (see page 3417)	This is GetAppDispatcher, a member of class IGetAppDispatcher.

13.2.1.2.1 IGetAppDispatcher.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class IGetAppDispatcher.

13.2.2 IGetWebAppServices Interface

Class Hierarchy

HTTPApp.IGetWebAppServices

Pascal

```
IGetWebAppServices = interface;
```

C++

```
__interface IGetWebAppServices;
```

File

HTTPApp

Description

This is class HTTPApp.IGetWebAppServices.

13.2.2.1 IGetWebAppServices Members

The following tables list the members exposed by IGetWebAppServices.

Public Methods

	Name	Description
	GetWebAppServices (see page 3418)	This is GetWebAppServices, a member of class IGetWebAppServices.

13.2.2.2 IGetWebAppServices Methods

The methods of the IGetWebAppServices class are listed here.

Public Methods

	Name	Description
	GetWebAppServices (see page 3418)	This is GetWebAppServices, a member of class IGetWebAppServices.

13.2.2.2.1 IGetWebAppServices.GetWebAppServices Method

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

This is GetWebAppServices, a member of class IGetWebAppServices.

13.2.3 IGetWebRequestHandler Interface

Class Hierarchy

HTTPApp.IGetWebRequestHandler

Pascal

```
IGetWebRequestHandler = interface;
```

C++

```
__interface IGetWebRequestHandler;
```

File

HTTPApp

Description

This is class HTTPApp.IGetWebRequestHandler.

13.2.3.1 IGetWebRequestHandler Members

The following tables list the members exposed by IGetWebRequestHandler.

Public Methods

	Name	Description
	GetWebRequestHandler (see page 3419)	This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

13.2.3.2 IGetWebRequestHandler Methods

The methods of the IGetWebRequestHandler class are listed here.

Public Methods

	Name	Description
	GetWebRequestHandler (see page 3419)	This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

13.2.3.2.1 IGetWebRequestHandler.GetWebRequestHandler Method

Pascal

```
function GetWebRequestHandler: IWebRequestHandler;
```

C++

```
__fastcall IWebRequestHandler GetWebRequestHandler();
```

Description

This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

13.2.4 IMultiModuleSupport Interface

Class Hierarchy

HTTPApp.IMultiModuleSupport

Pascal

```
IMultiModuleSupport = interface;
```

C++

```
__interface IMultiModuleSupport;
```

File

HTTPApp

Description

This is class HTTPApp.IMultiModuleSupport.

13.2.4.1 IMultiModuleSupport Members

The following tables list the members exposed by IMultiModuleSupport.

Public Methods

	Name	Description
	FinishContext (see page 3420)	This is FinishContext, a member of class IMultiModuleSupport.
	InitContext (see page 3420)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 3420)	This is InitModule, a member of class IMultiModuleSupport.

13.2.4.2 IMultiModuleSupport Methods

The methods of the IMultiModuleSupport class are listed here.

Public Methods

	Name	Description
	FinishContext (see page 3420)	This is FinishContext, a member of class IMultiModuleSupport.
	InitContext (see page 3420)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 3420)	This is InitModule, a member of class IMultiModuleSupport.

13.2.4.2.1 IMultiModuleSupport.FinishContext Method

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

This is FinishContext, a member of class IMultiModuleSupport.

13.2.4.2.2 IMultiModuleSupport.InitContext Method

Pascal

```
procedure InitContext(Request: TWebRequest; Response: TWebResponse);
```

C++

```
__fastcall InitContext(TWebRequest Request, TWebResponse Response);
```

Description

This is InitContext, a member of class IMultiModuleSupport.

13.2.4.2.3 IMultiModuleSupport.InitModule Method

Pascal

```
procedure InitModule(AModule: TComponent);
```

C++

```
__fastcall InitModule(TComponent * AModule);
```

Description

This is InitModule, a member of class IMultiModuleSupport.

13.2.5 IProduceContent Interface

IProduceContent is the interface for generating the content of a Web page.

Class Hierarchy

HTTPApp.IProduceContent

Pascal

```
IProduceContent = interface;
```

C++

```
__interface IProduceContent;
```

File

HTTPApp

Description

Content producers (such as TAdapaterPageProducer) implement the IProduceContent interface. This interface defines a single method, ProduceContent (see page 3421), which returns the content to assign to an HTTP response message in a WebSnap application.

Note: In C++, method declarations that use IProduceContent use the _di_IProduceContent type instead. This type is a DelphiInterface wrapper around the IProduceContent interface:

Note: typedef System::DelphiInterface< IProduceContent > _di_IProduceContent;

See Also

TCustomContentProducer (see page 3313)

TCustomWebPageModule

TAdapterPagePRoducer

TWebResponse (see page 3389)

13.2.5.1 IProduceContent Members

The following tables list the members exposed by IProduceContent.

Public Methods

	Name	Description
	ProduceContent (see page 3421)	Returns the content that the producer creates.

13.2.5.2 IProduceContent Methods

The methods of the IProduceContent class are listed here.

Public Methods

	Name	Description
	ProduceContent (see page 3421)	Returns the content that the producer creates.

13.2.5.2.1 IProduceContent.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall UnicodeString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content (see page 3314)

13.2.6 IProduceContentFrom Interface

Class Hierarchy

HTTPApp.IProduceContentFrom

Pascal

```
IProduceContentFrom = interface;
```

C++

```
__interface IProduceContentFrom;
```

File

HTTPApp

Description

This is class HTTPApp.IProduceContentFrom.

13.2.6.1 IProduceContentFrom Members

The following tables list the members exposed by IProduceContentFrom.

Public Methods

	Name	Description
	ProduceContentFromStream (see page 3422)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 3423)	This is ProduceContentFromString, a member of class IProduceContentFrom.

13.2.6.2 IProduceContentFrom Methods

The methods of the IProduceContentFrom class are listed here.

Public Methods

	Name	Description
	ProduceContentFromStream (see page 3422)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 3423)	This is ProduceContentFromString, a member of class IProduceContentFrom.

13.2.6.2.1 IProduceContentFrom.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class IProduceContentFrom.

13.2.6.2.2 IProduceContentFrom.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromString(const UnicodeString S);
```

Description

This is ProduceContentFromString, a member of class IProduceContentFrom.

13.2.7 ISetAppDispatcher Interface

Class Hierarchy

HTTPApp.ISetAppDispatcher

Pascal

```
ISetAppDispatcher = interface;
```

C++

```
__interface ISetAppDispatcher;
```

File

HTTPApp

Description

This is class HTTPApp.ISetAppDispatcher.

13.2.7.1 ISetAppDispatcher Members

The following tables list the members exposed by ISetAppDispatcher.

Public Methods

	Name	Description
	SetAppDispatcher (see page 3424)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

13.2.7.2 ISetAppDispatcher Methods

The methods of the ISetAppDispatcher class are listed here.

Public Methods

	Name	Description
	SetAppDispatcher (see page 3424)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

13.2.7.2.1 ISetAppDispatcher.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class ISetAppDispatcher.

13.2.8 IWebAppServices Interface

IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.

Class Hierarchy

HTTPApp.IWebAppServices

Pascal

```
IWebAppServices = interface;
```

C++

```
__interface IWebAppServices;
```

File

HTTPApp

Description

WebSnap applications use the IWebAppServices interface to drive the processing of incoming request messages. When the WebSnap application receives an HTTP request message it

1. Calls InitContext (see page 3426) to set up a Web context for the request.
2. Calls HandleRequest (see page 3426) to forward the request to the components that process it.
3. Calls FinishContext (see page 3425) to shut down the Web context in an orderly fashion.

In addition to providing access to these core methods of the application, IWebAppServices introduces the ExceptionHandler (see page 3427) property. The ExceptionHandler (see page 3427) property allows the application access to the component that handles any exceptions that escape all of the try blocks in the application while processing the message.

Note: Method declarations that use IWebAppServices use the _di_IWebAppServices type instead. This type is a DelphiInterface wrapper around the IWebAppServices interface:

Note: typedef System::DelphiInterface< IWebAppServices > _di_IWebAppServices;

See Also

TMultiModuleWebAppServices

AppServices

WebContext

13.2.8.1 IWebAppServices Members

The following tables list the members exposed by IWebAppServices.

Public Methods

	Name	Description
	FinishContext (see page 3425)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 3426)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleRequest (see page 3426)	Directs the processing of the current request message.
	InitContext (see page 3426)	Creates and initializes the Web context for the current Web request.

Public Properties

	Name	Description
R	ExceptionHandler (see page 3427)	Indicates the object that handles exceptions that arise when processing a request message.

13.2.8.2 IWebAppServices Methods

The methods of the IWebAppServices class are listed here.

Public Methods

	Name	Description
	FinishContext (see page 3425)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 3426)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleRequest (see page 3426)	Directs the processing of the current request message.
	InitContext (see page 3426)	Creates and initializes the Web context for the current Web request.

13.2.8.2.1 IWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message (including the Web context).

See Also

InitContext (see page 3426)

WebContext

13.2.8.2.2 IWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler is the read implementation of the ExceptionHandler (see page 3427) property.

See Also

ExceptionHandler (see page 3427)

13.2.8.2.3 IWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext (see page 3426) method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext (see page 3425) is called to perform any final cleanup.

HandleRequest locates the dispatcher components that handle the request and forwards the request to a dispatcher for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

InitContext (see page 3426)

FinishContext (see page 3425)

13.2.8.2.4 IWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message.

Request describes the new request message.

Response is the response object that the WebSnap application should fill out when handling the current request.

See Also

FinishContext (see page 3425)

WebContext

13.2.8.3 IWebAppServices Properties

The properties of the IWebAppServices class are listed here.

Public Properties

	Name	Description
	ExceptionHandler (see page 3427)	Indicates the object that handles exceptions that arise when processing a request message.

13.2.8.3.1 IWebAppServices.ExceptionHandler Property

Indicates the object that handles exceptions that arise when processing a request message.

Pascal

```
property ExceptionHandler: TObject;
```

C++

```
__property TObject * ExceptionHandler;
```

Description

ExceptionHandler is the object that the WebSnap application calls when an exception is raised while trying to process a request message. The request handler must implement the IWebExceptionHandler (see page 3433) interface, which includes only a single method: HandleException.

13.2.9 IWebDispatch Interface

Class Hierarchy

HTTPApp.IWebDispatch

Pascal

```
IWebDispatch = interface;
```

C++

```
__interface IWebDispatch;
```

File

HTTPApp

See Also

TCustomWebDispatcher (see page 3318)

13.2.9.1 IWebDispatch Members

The following tables list the members exposed by IWebDispatch.

Public Methods

	Name	Description
	DispatchEnabled (see page 3428)	
	DispatchMask (see page 3429)	
	DispatchMethodType (see page 3429)	
	DispatchRequest (see page 3429)	

Public Properties

	Name	Description
R	Enabled (see page 3429)	
R	Mask (see page 3430)	
R	MethodType (see page 3430)	This is MethodType, a member of class IWebDispatch.

13.2.9.2 IWebDispatch Methods

The methods of the IWebDispatch class are listed here.

Public Methods

	Name	Description
	DispatchEnabled (see page 3428)	
	DispatchMask (see page 3429)	
	DispatchMethodType (see page 3429)	
	DispatchRequest (see page 3429)	

13.2.9.2.1 IWebDispatch.DispatchEnabled Method

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

See Also

Enabled (see page 3429)

13.2.9.2.2 IWebDispatch.DispatchMask Method

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

See Also

Mask (see page 3430)

13.2.9.2.3 IWebDispatch.DispatchMethodType Method

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

See Also

MethodType (see page 3430)

13.2.9.2.4 IWebDispatch.DispatchRequest Method

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

13.2.9.3 IWebDispatch Properties

The properties of the IWebDispatch class are listed here.

Public Properties

	Name	Description
R	Enabled (see page 3429)	
R	Mask (see page 3430)	
R	MethodType (see page 3430)	This is MethodType, a member of class IWebDispatch.

13.2.9.3.1 IWebDispatch.Enabled Property

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

See Also

DispatchEnabled (see page 3428)

13.2.9.3.2 IWebDispatch.Mask Property

Pascal

```
property Mask: TMask;
```

C++

```
__property TMask Mask;
```

See Also

DispatchMask (see page 3429)

13.2.9.3.3 IWebDispatch.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class IWebDispatch.

13.2.10 IWebDispatchActions Interface

IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.

Class Hierarchy

HTTPApp.IWebDispatchActions

Pascal

```
IWebDispatchActions = interface;
```

C++

```
__interface IWebDispatchActions;
```

File

HTTPApp

Description

Web dispatcher components (TWebDispatcher (see page 3349) and TWebModule (see page 3354)) implement the IWebDispatchActions interface. This interface serves primarily to identify components that use action items to handle HTTP request messages.

To obtain an IWebDispatchActions interface, use the DispatchActions property of a TWebAppComponents object.

Note: In C++, method declarations that use IWebDispatchActions use the _di_IWebDispatchActions type instead. This type is a DelphiInterface wrapper around the IWebDispatchActions interface:

Note: typedef System::DelphiInterface< IWebDispatchActions > _di_IWebDispatchActions;

See Also

TWebDispatcher (see page 3349)

TWEbModule

TWebAppComponents

TCustomWebAppComponents

13.2.11 IWebDispatcherAccess Interface

IWebDispatcherAccess provides access to the current HTTP request and response object.

Class Hierarchy

Pascal

```
IWebDispatcherAccess = interface;
```

C++

```
__interface IWebDispatcherAccess;
```

File

HTTPApp

Description

The IWebDispatcherAccess interface declares two methods, Request (see page 3432) and Response (see page 3432). These return the objects that represent the current HTTP request message that is being processed and the emerging HTTP response message that will be sent back to the caller.

In Web Broker applications, a TWebDispatcher (see page 3349) or TWebModule (see page 3354) component implements this interface. In WebSnap applications, a TWebAppComponents component implements this interface.

Note: In C++, method declarations that use IWebDispatcherAccess use the _di_IWebDispatcherAccess type instead. This type is a DelphiInterface wrapper around the IWebDispatcherAccess interface:

Note: typedef System::DelphiInterface< IWebDispatcherAccess > _di_IWebDispatcherAccess;

See Also

TCustomContentProducer (see page 3313)

TWebDispatcher (see page 3349)

TWebAppComponents

TWebModule (see page 3354)

13.2.11.1 IWebDispatcherAccess Members

The following tables list the members exposed by IWebDispatcherAccess.

Public Methods

	Name	Description
	Request (see page 3432)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 3432)	Returns the Web response object for the HTTP response message that the application is assembling.

13.2.11.2 IWebDispatcherAccess Methods

The methods of the IWebDispatcherAccess class are listed here.

Public Methods

	Name	Description
	Request (see page 3432)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 3432)	Returns the Web response object for the HTTP response message that the application is assembling.

13.2.11.2.1 IWebDispatcherAccess.Request Method

Returns the Web request object for the HTTP request message to which the application is responding.

Pascal

```
function Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Request();
```

Description

Call Request to obtain access to the current Web request object. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 3432)

13.2.11.2.2 IWebDispatcherAccess.Response Method

Returns the Web response object for the HTTP response message that the application is assembling.

Pascal

```
function Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Response();
```

Description

Call Response to obtain access to the current Web response object. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 3432)

13.2.12 IWebExceptionHandler Interface

Class Hierarchy

HTTPApp.IWebExceptionHandler

Pascal

```
IWebExceptionHandler = interface;
```

C++

```
__interface IWebExceptionHandler;
```

File

HTTPApp

Description

This is class HTTPApp.IWebExceptionHandler.

13.2.12.1 IWebExceptionHandler Members

The following tables list the members exposed by IWebExceptionHandler.

Public Methods

	Name	Description
	HandleException (see page 3433)	This is HandleException, a member of class IWebExceptionHandler.

13.2.12.2 IWebExceptionHandler Methods

The methods of the IWebExceptionHandler class are listed here.

Public Methods

	Name	Description
	HandleException (see page 3433)	This is HandleException, a member of class IWebExceptionHandler.

13.2.12.2.1 IWebExceptionHandler.HandleException Method

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

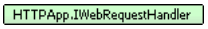
Description

This is HandleException, a member of class IWebExceptionHandler.

13.2.13 IWebRequestHandler Interface

IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

Class Hierarchy



Pascal

```
IWebRequestHandler = interface;
```

C++

```
__interface IWebRequestHandler;
```

File

HTTPApp

Description

Dispatcher components such as TPageDispatcher , TAdapterDispatcher, and TWebDispatcher (see page 3349) implement the IWebRequestHandler interface. This interface includes only a single method, HandleRequest (see page 3435), which processes an HTTP request message, filling out the corresponding response, which can then be returned to the caller.

Note: In C++, method declarations that use IWebRequestHandler use the _di_IWebRequestHandler type instead. This type is a DelphiInterface wrapper around the IWebRequestHandler interface:

Note: typedef System::DelphiInterface< IWebRequestHandler > _di_IWebRequestHandler;

See Also

TWebRequest (see page 3358)

TWebResponse (see page 3389)

TMultiModuleWebAppServices

TPageDispatcher

TAdapterDispatcher

TWebDispatcher (see page 3349)

13.2.13.1 IWebRequestHandler Members

The following tables list the members exposed by IWebRequestHandler.

Public Methods

	Name	Description
	HandleRequest (see page 3435)	Processes an HTTP request message.

13.2.13.2 IWebRequestHandler Methods

The methods of the IWebRequestHandler class are listed here.

Public Methods

	Name	Description
	HandleRequest (see page 3435)	Processes an HTTP request message.

13.2.13.2.1 IWebRequestHandler.HandleRequest Method

Processes an HTTP request message.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

Call HandleRequest to process an HTTP request message and obtain the response that should be sent back to the caller.

Request is the request object that describes the request to process.

Response is the response object that should be filled out when the request is processed.

13.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DayOfWeekStr (see page 3436)	Converts a TDateTime into a string representing the day of the week.
	DosPathToUnixPath (see page 3437)	Converts a DOS-compatible path specification to a Unix-compatible path specification.
	EncodingFromContentType (see page 3437)	This is function HTTPApp.EncodingFromContentType.
	EncodingGetBytes (see page 3437)	This is function HTTPApp.EncodingGetBytes.
	EncodingGetString (see page 3438)	This is function HTTPApp.EncodingGetString.
	ExtractHeaderFields (see page 3438)	Parses a multi-valued string into its constituent fields.
	ExtractHeaderFields (see page 3438)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 3439)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 3439)	Parses a multi-valued string into its constituent fields.
	HTMLDecode (see page 3439)	This is function HTTPApp.HTMLDecode.
	HTMLEncode (see page 3440)	This is function HTTPApp.HTMLEncode.
	HTTPDecode (see page 3440)	Decodes a string that includes HTTP escape characters.
	HTTPEncode (see page 3441)	Converts a string into a form that contains no values that are illegal in an HTTP message header.
	MonthStr (see page 3441)	Converts a TDateTime into a string representing the month.
	ParseDate (see page 3442)	Converts a date value from an HTTP message header into a TDateTime.
	RegisterContentParser (see page 3442)	This is function HTTPApp.RegisterContentParser.
	StatusString (see page 3442)	Returns the explanatory string associated with one of the predefined HTTP response status codes.
	UnixPathToDosPath (see page 3443)	Converts a Unix-compatible path specification to a Dos-compatible path specification.

13.3.1 HTTPApp.DayOfWeekStr Function

Converts a TDateTime into a string representing the day of the week.

Pascal

```
function DayOfWeekStr(DateTime: TDateTime): string;
```

C++

```
UnicodeString DayOfWeekStr(TDateTime DateTime);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. DayOfWeekStr converts a TDateTime into a string representing the day of the week. DayOfWeekStr allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

See Also

MonthStr (see page 3441)

13.3.2 HTTPApp.DosPathToUnixPath Function

Converts a DOS-compatible path specification to a Unix-compatible path specification.

Pascal

```
function DosPathToUnixPath(const Path: string): string;
```

C++

```
UnicodeString DosPathToUnixPath(const UnicodeString Path);
```

File

HTTPApp

Description

Use DosPathToUnixPath to convert a path name on the local machine to a path name that can be sent in an HTTP message, or that will be used by a Unix-based application. DosPathToUnixPath converts all backslash characters (\) to forward slashes (/).

The Path parameter is the Dos-compatible path. The return value is the corresponding Unix-compatible path.

See Also

UnixPathToDosPath (see page 3443)

13.3.3 HTTPApp.EncodingFromContentType Function

Pascal

```
function EncodingFromContentType(const AContentType: AnsiString): TEncoding;
```

C++

```
TEncoding EncodingFromContentType(const AnsiString AContentType);
```

File

HTTPApp

Description

This is function HTTPApp.EncodingFromContentType.

13.3.4 HTTPApp.EncodingGetBytes Function

Pascal

```
function EncodingGetBytes(const AContentType: AnsiString; const AValue: string): TBytes;
```

C++

```
TBytes EncodingGetBytes(const AnsiString AContentType, const UnicodeString AValue);
```

File

HTTPApp

Description

This is function HTTPApp.EncodingGetBytes.

13.3.5 HTTPApp.EncodingGetString Function

Pascal

```
function EncodingGetString(const AContentType: AnsiString; const AValue: AnsiString): string;
```

C++

```
UnicodeString EncodingGetString(const AnsiString AContentType, const AnsiString AValue);
```

File

HTTPApp

Description

This is function HTTPApp.EncodingGetString.

13.3.6 HTTPApp.ExtractHeaderFields Function

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractHeaderFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content:
PChar; Strings: TStrings; Decode: Boolean; StripQuotes: Boolean = False); overload;
procedure ExtractHeaderFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; const Content:
AnsiString; Strings: TStrings; Decode: Boolean; StripQuotes: Boolean = False); overload;
```

C++

```
ExtractHeaderFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings, Boolean Decode, Boolean StripQuotes = False);
ExtractHeaderFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const AnsiString Content,
TStrings Strings, Boolean Decode, Boolean StripQuotes = False);
```

File

HTTPApp

Description

ExtractHeaderFields is a general utility to parse multi-valued HTTP header strings into separate substrings.

Separators is a set of characters that are used to separate individual values within the multi-valued string.

WhiteSpace is a set of characters that are to be ignored when parsing the string.

Content is the multi-valued string to be parsed.

Strings is the TStrings object that receives the individual values that are parsed from Content.

StripQuotes determines whether the surrounding quotes are removed from the resulting items. When StripQuotes is true, surrounding quotes are removed before substrings are added to Strings.

Note: Characters contained in Separators or WhiteSpace are treated as part of a value substring if the substring is surrounded by single or double quote marks. HTTP escape characters are converted using the HTTPDecode (see page 3440) function.

See Also

HTTPDecode (see page 3440)

TStrings

13.3.7 HTTPApp.ExtractHTTPFields Function

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractHTTPFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content: PChar;
Strings: TStrings; StripQuotes: Boolean = False); overload;
procedure ExtractHTTPFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; const Content:
AnsiString; Strings: TStrings; StripQuotes: Boolean = False); overload;
```

C++

```
ExtractHTTPFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings, Boolean StripQuotes = False);
ExtractHTTPFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const AnsiString Content,
TStrings Strings, Boolean StripQuotes = False);
```

File

HTTPApp

Description

ExtractHTTPFields is a general utility to parse multi-valued HTTP header strings into separate substrings.

Separators is a set of characters that are used to separate individual values within the multi-valued string.

WhiteSpace is a set of characters that are to be ignored when parsing the string.

Content is the multi-valued string to be parsed.

Strings is the TStrings object that receives the individual values that are parsed from Content.

StripQuotes determines whether the surrounding quotes are removed from the resulting items. When StripQuotes is true, surrounding quotes are removed before substrings are added to Strings.

Note: Characters contained in Separators or WhiteSpace are treated as part of a value substring if the substring is surrounded by single or double quote marks. HTTP escape characters are converted using the HTTPDecode (see page 3440) function.

See Also

HTTPDecode (see page 3440)

TStrings

13.3.8 HTTPApp.HTMLDecode Function

Pascal

```
function HTMLDecode(const AStr: String): String;
```

C++

```
UnicodeString HTMLDecode(const UnicodeString AStr);
```

File

HTTPApp

Description

This is function HTTPApp.HTMLDecode.

13.3.9 HTTPApp.HTMLEncode Function

Pascal

```
function HTMLEncode(const AStr: String): String;
```

C++

```
UnicodeString HTMLEncode(const UnicodeString AStr);
```

File

HTTPApp

Description

This is function HTTPApp.HTMLEncode.

13.3.10 HTTPApp.HTTPDecode Function

Decodes a string that includes HTTP escape characters.

Pascal

```
function HTTPDecode(const AStr: AnsiString): AnsiString;
```

C++

```
AnsiString HTTPDecode(const AnsiString AStr);
```

File

HTTPApp

Description

HTTPDecode decodes the escape characters (+ and %) in a string that comes from an HTTP message header. Plus characters (+) are converted into spaces. Percent signs (%) indicate that the next two characters are the hex representation of a character. If a percent sign is followed by another percent sign (%%), it is converted into a single percent sign.

For example, the string

%%+%3f

is converted into

% ?

The AStr parameter is the undecoded string. The return value is the converted version of the string.

See Also

ExtractHTTPFields (see page 3439)

HTTPEncode (see page 3441)

13.3.11 HTTPApp.HTTPEncode Function

Converts a string into a form that contains no values that are illegal in an HTTP message header.

Pascal

```
function HTTPEncode(const AStr: AnsiString): AnsiString;
```

C++

```
AnsiString HTTPEncode(const AnsiString AStr);
```

File

HTTPApp

Description

HTTPEncode converts all characters in the AStr parameter except for the letters A through Z (and a through z), numerals 0 through 9, the asterisk (*), dollar sign ($), exclamation point (!), at sign (@), period (.), underscore (_), single quote ('), comma (,) parentheses, and hyphen (-). Spaces are converted to plus characters (+), and all other characters are converted into hex values preceded by the percent sign (%).

For example, the string

% ?

is converted into

%%+%3f

HTTPEncode returns the converted string.

See Also

HTTPDecode (see page 3440)

13.3.12 HTTPApp.MonthStr Function

Converts a TDateTime into a string representing the month.

Pascal

```
function MonthStr(DateTime: TDateTime): string;
```

C++

```
UnicodeString MonthStr(TDateTime DateTime);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. MonthStr converts a TDateTime into a string representing the month. MonthStr allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

See Also

DayOfWeekStr (see page 3436)

13.3.13 HTTPApp.ParseDate Function

Converts a date value from an HTTP message header into a TDateTime.

Pascal

```
function ParseDate(const DateStr: string): TDateTime;
```

C++

```
TDateTime ParseDate(const UnicodeString DateStr);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. ParseDate converts from any of these formats into a TDateTime. ParseDate allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

Use ParseDate when interpreting temporal header values in HTTP request messages when they are not already parsed as a property of TWebRequest (see page 3358). TWebRequest (see page 3358) parses the Date header and Expires header automatically.

See Also

TWebRequest (see page 3358)

13.3.14 HTTPApp.RegisterContentParser Function

Pascal

```
procedure RegisterContentParser(AClass: TContentParserClass);
```

C++

```
RegisterContentParser(TContentParserClass AClass);
```

File

HTTPApp

Description

This is function HTTPApp.RegisterContentParser.

13.3.15 HTTPApp.StatusString Function

Returns the explanatory string associated with one of the predefined HTTP response status codes.

Pascal

```
function StatusString(StatusCode: Integer): AnsiString;
```

C++

```
AnsiString StatusString(int StatusCode);
```

File

HTTPApp

Description

The HTTP protocol specification defines a number of standard status codes that indicate the status of an HTTP request message. HTTP applications interpret these status codes, but for human beings, each of these codes has a corresponding string that provides a more understandable indication of the status of the HTTP request.

Call StatusString to obtain the string that corresponds to a standard status code. If the StatusCode parameter is not one of the codes defined in the HTTP standard, StatusString returns an empty string.

See Also

TWebResponse (see page 3389)

13.3.16 HTTPApp.UnixPathToDosPath Function

Converts a Unix-compatible path specification to a Dos-compatible path specification.

Pascal

```
function UnixPathToDosPath(const Path: string): string;
```

C++

```
UnicodeString UnixPathToDosPath(const UnicodeString Path);
```

File

HTTPApp

Description

Use UnixPathToDosPath to convert a path name obtained from an HTTP message or other Unix-based application to a path name that can be used by Windows. UnixPathToDosPath converts all forward slashes (/) to backslash characters (\).

The Path parameter is the Unix-compatible path. The return value is the corresponding Dos-compatible path.

See Also

DosPathToUnixPath (see page 3437)

13.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMethodType (see page 3444)	This is record HTTPApp.TMethodType.
	TWebModuleCacheMode (see page 3444)	This is record HTTPApp.TWebModuleCacheMode.
	TWebModuleCreateMode (see page 3445)	This is record HTTPApp.TWebModuleCreateMode.

13.4.1 HTTPApp.TMethodType Enumeration

Pascal

```
TMethodType = (
  mtAny,
  mtGet,
  mtPut,
  mtPost,
  mtHead
);
```

C++

```
enum TMethodType {
  mtAny,
  mtGet,
  mtPut,
  mtPost,
  mtHead
};
```

File

HTTPApp

Description

This is record HTTPApp.TMethodType.

13.4.2 HTTPApp.TWebModuleCacheMode Enumeration

Pascal

```
TWebModuleCacheMode = (
  caCache,
  caDestroy
);
```

C++

```
enum TWebModuleCacheMode {
  caCache,
  caDestroy
```

```
};
```

File

HTTPApp

Description

This is record HTTPApp.TWebModuleCacheMode.

13.4.3 HTTPApp.TWebModuleCreateMode Enumeration

Pascal

```
TWebModuleCreateMode = (
  crOnDemand,
  crAlways
);
```

C++

```
enum TWebModuleCreateMode {
  crOnDemand,
  crAlways
};
```

File

HTTPApp

Description

This is record HTTPApp.TWebModuleCreateMode.

13.5 Types

The following table lists types in this documentation.

Types

Name	Description
TContentParserClass (see page 3446)	This is type HTTPApp.TContentParserClass.
THTTPMethodEvent (see page 3446)	This is type HTTPApp.THTTPMethodEvent.
TModuleAddedProc (see page 3447)	This is type HTTPApp.TModuleAddedProc.
TWebExceptionEvent (see page 3447)	This is type HTTPApp.TWebExceptionEvent.
TWebModuleContext (see page 3447)	This is type HTTPApp.TWebModuleContext.

13.5.1 HTTPApp.TContentParserClass Type

Pascal

```
TContentParserClass = class of TAbstractContentParser;
```

C++

```
TAbstractContentParser TContentParserClass;
```

File

HTTPApp

Description

This is type HTTPApp.TContentParserClass.

13.5.2 HTTPApp.THTTPMethodEvent Type

Pascal

```
THTTPMethodEvent = procedure (Sender: TObject; Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
THTTPMethodEvent)();
```

File

HTTPApp

Description

This is type HTTPApp.THTTPMethodEvent.

13.5.3 HTTPApp.TModuleAddedProc Type

Pascal

```
TModuleAddedProc = procedure (AWebModule: TComponent) of object;
```

C++

```
(AWebModule: TComponent) ( TModuleAddedProc)();
```

File

HTTPApp

Description

This is type HTTPApp.TModuleAddedProc.

13.5.4 HTTPApp.TWebExceptionEvent Type

Pascal

```
TWebExceptionEvent = procedure (Sender: TObject; E: Exception; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; E: Exception; var Handled: Boolean) ( TWebExceptionEvent)();
```

File

HTTPApp

Description

This is type HTTPApp.TWebExceptionEvent.

13.5.5 HTTPApp.TWebModuleContext Type

Pascal

```
TWebModuleContext = TObject;
```

C++

```
TObject * TWebModuleContext;
```

File

HTTPApp

Description

This is type HTTPApp.TWebModuleContext.

13.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MAX_DATETIMES (see page 3448)	This is constant HTTPApp.MAX_DATETIMES.
MAX_INTEGERS (see page 3448)	This is constant HTTPApp.MAX_INTEGERS.
MAX_STRINGS (see page 3449)	This is constant HTTPApp.MAX_STRINGS.
sDateFormat (see page 3449)	This is constant HTTPApp.sDateFormat.

13.6.1 HTTPApp.MAX_DATETIMES Constant

Pascal

```
MAX_DATETIMES = 3;
```

C++

```
MAX_DATETIMES = 3;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_DATETIMES.

13.6.2 HTTPApp.MAX_INTEGERS Constant

Pascal

```
MAX_INTEGERS = 1;
```

C++

```
MAX_INTEGERS = 1;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_INTEGERS.

13.6.3 HTTPApp.MAX_STRINGS Constant

Pascal

```
MAX_STRINGS = 12;
```

C++

```
MAX_STRINGS = 12;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_STRINGS.

13.6.4 HTTPApp.sDateFormat Constant

Pascal

```
sDateFormat = '"%s", dd "%s" yyyy hh:nn:ss';
```

C++

```
sDateFormat = '"%s", dd "%s" yyyy hh:nn:ss';
```

File

HTTPApp

Description

This is constant HTTPApp.sDateFormat.

14 HTTPProd Namespace

This is namespace HTTPProd.

Topics

Name	Description
Classes (see page 3453)	The following table lists classes in this documentation.
Interfaces (see page 3502)	The following table lists interfaces in this documentation.
Functions (see page 3518)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3521)	The following table lists structs, records, enums in this documentation.
Types (see page 3523)	The following table lists types in this documentation.
Variables (see page 3525)	The following table lists variables in this documentation.
Constants (see page 3526)	The following table lists constants in this documentation.

Classes

	Name	Description
	EScriptError (see page 3453)	This is class HTTPProd.EScriptError.
	TAbstractScriptEnginesList (see page 3455)	This is class HTTPProd.TAbstractScriptEnginesList.
	TAbstractScriptError (see page 3457)	This is class HTTPProd.TAbstractScriptError.
	TAbstractScriptErrors (see page 3460)	This is class HTTPProd.TAbstractScriptErrors.
	TAbstractScriptProducer (see page 3463)	
	TBasePageProducer (see page 3464)	TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.
	TCustomPageProducer (see page 3473)	TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.
	THTMLTableAttributes (see page 3478)	THTMLTableAttributes represents the display attributes of an HTML table tag.
	THTMLTableCellAttributes (see page 3484)	THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.
	THTMLTableElementAttributes (see page 3484)	THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.
	THTMLTableHeaderAttributes (see page 3488)	THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.
	THTMLTableRowAttributes (see page 3491)	THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.
	THTMLTagAttributes (see page 3492)	THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.
	TPageProducer (see page 3496)	TPageProducer produces a string of HTML commands based on an input template.

Constants

Name	Description
HTMLAlign (see page 3526)	This is constant HTTPProd.HTMLAlign.
HTMLVAlign (see page 3526)	This is constant HTTPProd.HTMLVAlign.

Functions

	Name	Description
	ContentFromScriptFile (see page 3518)	This is function HTTPProd.ContentFromScriptFile.
	ContentFromScriptStream (see page 3518)	This is function HTTPProd.ContentFromScriptStream.
	ContentFromStream (see page 3519)	This is function HTTPProd.ContentFromStream.
	ContentFromString (see page 3519)	This is function HTTPProd.ContentFromString.
	FindComponentWebModuleContext (see page 3519)	This is function HTTPProd.FindComponentWebModuleContext.
	GetEncodingOfStream (see page 3520)	This is function HTTPProd.GetEncodingOfStream.
	GetTagID (see page 3520)	This is function HTTPProd.GetTagID.

Interfaces

	Name	Description
	IDesignerFileManager (see page 3502)	This is class HTTPProd.IDesignerFileManager.
	IGetLocateFileService (see page 3503)	This is class HTTPProd.IGetLocateFileService.
	IGetProducerTemplate (see page 3504)	This is class HTTPProd.IGetProducerTemplate.
	ILocateFileService (see page 3505)	This is class HTTPProd.ILocateFileService.
	IScriptContext (see page 3506)	This is class HTTPProd.IScriptContext.
	IScriptProducer (see page 3508)	This is class HTTPProd.IScriptProducer.
	IWebVariableName (see page 3513)	This is class HTTPProd.IWebVariableName.
	IWebVariablesContainer (see page 3514)	This is class HTTPProd.IWebVariablesContainer.

Structs, Records, Enums

	Name	Description
	THTMLAlign (see page 3521)	This is record HTTPProd.THTMLAlign.
	THTMLVAlign (see page 3521)	This is record HTTPProd.THTMLVAlign.
	TTag (see page 3522)	This is record HTTPProd.TTag.

Types

Name	Description
THandledTagProc (see page 3523)	This is type HTTPProd.THandledTagProc.
THandleTagProc (see page 3523)	This is type HTTPProd.THandleTagProc.
THTMLBgColor (see page 3524)	This is type HTTPProd.THTMLBgColor.
THTMLTagEvent (see page 3524)	This is type HTTPProd.THTMLTagEvent.
TScriptProducerClass (see page 3524)	This is type HTTPProd.TScriptProducerClass.

Variables

Name	Description
DesignerFileManager (see page 3525)	This is variable HTTPProd.DesignerFileManager.
ScriptEnginesList (see page 3525)	This is variable HTTPProd.ScriptEnginesList.

14.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EScriptError (see page 3453)	This is class HTTPProd.EScriptError.
	TAbstractScriptEnginesList (see page 3455)	This is class HTTPProd.TAbstractScriptEnginesList.
	TAbstractScriptError (see page 3457)	This is class HTTPProd.TAbstractScriptError.
	TAbstractScriptErrors (see page 3460)	This is class HTTPProd.TAbstractScriptErrors.
	TAbstractScriptProducer (see page 3463)	
	TBasePageProducer (see page 3464)	TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.
	TCustomPageProducer (see page 3473)	TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.
	THTMLTableAttributes (see page 3478)	THTMLTableAttributes represents the display attributes of an HTML table tag.
	THTMLTableCellAttributes (see page 3484)	THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.
	THTMLTableElementAttributes (see page 3484)	THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.
	THTMLTableHeaderAttributes (see page 3488)	THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.
	THTMLTableRowAttributes (see page 3491)	THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.
	THTMLTagAttributes (see page 3492)	THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.
	TPageProducer (see page 3496)	TPageProducer produces a string of HTML commands based on an input template.

14.1.1 EScriptError Class

Class Hierarchy

Pascal

```
EScriptError = class(EWebBrokerException);
```

C++

```
class EScriptError : public EWebBrokerException;
```

File

HTTPProd

Description

This is class HTTPProd.EScriptError.

14.1.1.1 EScriptError Members

The following tables list the members exposed by EScriptError.

Public Methods

	Name	Description
	Create (see page 3454)	This is Create, a member of class EScriptError.
V	Destroy (see page 3454)	Disposes of an object instance.

Public Properties

	Name	Description
R	Content (see page 3455)	This is Content, a member of class EScriptError.
R	Errors (see page 3455)	This is Errors, a member of class EScriptError.

14.1.1.2 EScriptError.Create Constructor

Pascal

```
constructor Create(const AErrors: TAbstractScriptErrors; const AContent: string);
```

C++

```
__fastcall EScriptError(const TAbstractScriptErrors AErrors, const UnicodeString AContent);
```

Description

This is Create, a member of class EScriptError.

14.1.1.3 EScriptError.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~EScriptError();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

14.1.1.4 EScriptError Properties

The properties of the EScriptError class are listed here.

Public Properties

	Name	Description
	Content (see page 3455)	This is Content, a member of class EScriptError.
	Errors (see page 3455)	This is Errors, a member of class EScriptError.

14.1.1.4.1 EScriptError.Content Property

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

This is Content, a member of class EScriptError.

14.1.1.4.2 EScriptError.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class EScriptError.

14.1.2 TAbstractScriptEnginesList Class

Class Hierarchy

Pascal

```
TAbstractScriptEnginesList = class(TObject);
```

C++

```
class TAbstractScriptEnginesList : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptEnginesList.

14.1.2.1 TAbstractScriptEnginesList Members

The following tables list the members exposed by TAbstractScriptEnginesList.

Public Methods

	Name	Description
	Create (see page 3456)	Constructs an object and initializes its data before the object is first used.
	FindScriptProducerClass (see page 3456)	This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.

14.1.2.2 TAbstractScriptEnginesList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractScriptEnginesList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy

System::TObject::Free

14.1.2.3 TAbstractScriptEnginesList Methods

The methods of the TAbstractScriptEnginesList class are listed here.

Public Methods

	Name	Description
	FindScriptProducerClass (see page 3456)	This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.

14.1.2.3.1 TAbstractScriptEnginesList.FindScriptProducerClass Method

Pascal

```
function FindScriptProducerClass(const ALanguageName: string): TScriptProducerClass; virtual;
abstract;
```

C++

```
virtual __fastcall TScriptProducerClass FindScriptProducerClass(const UnicodeString
ALanguageName) = 0;
```

Description

This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.

14.1.3 TAbstractScriptError Class

Class Hierarchy

Pascal

```
TAbstractScriptError = class(TObject);
```

C++

```
class TAbstractScriptError : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptError.

14.1.3.1 TAbstractScriptError Members

The following tables list the members exposed by TAbstractScriptError.

Protected Methods

	Name	Description
	GetCharPos (see page 3458)	This is GetCharPos, a member of class TAbstractScriptError.
	GetDescription (see page 3458)	This is GetDescription, a member of class TAbstractScriptError.
	GetFileName (see page 3458)	This is GetFileName, a member of class TAbstractScriptError.
	GetLine (see page 3458)	This is GetLine, a member of class TAbstractScriptError.
	GetSourceLine (see page 3458)	This is GetSourceLine, a member of class TAbstractScriptError.
	SetDescription (see page 3459)	This is SetDescription, a member of class TAbstractScriptError.

Public Properties

	Name	Description
	CharPos (see page 3459)	This is CharPos, a member of class TAbstractScriptError.
	Description (see page 3459)	This is Description, a member of class TAbstractScriptError.
	FileName (see page 3460)	This is FileName, a member of class TAbstractScriptError.
	Line (see page 3460)	This is Line, a member of class TAbstractScriptError.
	SourceLine (see page 3460)	This is SourceLine, a member of class TAbstractScriptError.

14.1.3.2 TAbstractScriptError Methods

The methods of the TAbstractScriptError class are listed here.

Protected Methods

	Name	Description
	GetCharPos (see page 3458)	This is GetCharPos, a member of class TAbstractScriptError.
	GetDescription (see page 3458)	This is GetDescription, a member of class TAbstractScriptError.
	GetFileName (see page 3458)	This is GetFileName, a member of class TAbstractScriptError.
	GetLine (see page 3458)	This is GetLine, a member of class TAbstractScriptError.
	GetSourceLine (see page 3458)	This is GetSourceLine, a member of class TAbstractScriptError.

	SetDescription (see page 3459)	This is SetDescription, a member of class TAbstractScriptError.

14.1.3.2.1 TAbstractScriptError.GetCharPos Method

Pascal

```
function GetCharPos: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCharPos() = 0;
```

Description

This is GetCharPos, a member of class TAbstractScriptError.

14.1.3.2.2 TAbstractScriptError.GetDescription Method

Pascal

```
function GetDescription: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetDescription() = 0;
```

Description

This is GetDescription, a member of class TAbstractScriptError.

14.1.3.2.3 TAbstractScriptError.GetFileName Method

Pascal

```
function GetFileName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetFileName() = 0;
```

Description

This is GetFileName, a member of class TAbstractScriptError.

14.1.3.2.4 TAbstractScriptError.GetLine Method

Pascal

```
function GetLine: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetLine() = 0;
```

Description

This is GetLine, a member of class TAbstractScriptError.

14.1.3.2.5 TAbstractScriptError.GetSourceLine Method

Pascal

```
function GetSourceLine: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetSourceLine() = 0;
```

Description

This is GetSourceLine, a member of class TAbstractScriptError.

14.1.3.2.6 TAbstractScriptError.SetDescription Method

Pascal

```
procedure SetDescription(const AValue: string); virtual; abstract;
```

C++

```
virtual __fastcall SetDescription(const UnicodeString AValue) = 0;
```

Description

This is SetDescription, a member of class TAbstractScriptError.

14.1.3.3 TAbstractScriptError Properties

The properties of the TAbstractScriptError class are listed here.

Public Properties

	Name	Description
R	CharPos (see page 3459)	This is CharPos, a member of class TAbstractScriptError.
	Description (see page 3459)	This is Description, a member of class TAbstractScriptError.
R	FileName (see page 3460)	This is FileName, a member of class TAbstractScriptError.
R	Line (see page 3460)	This is Line, a member of class TAbstractScriptError.
R	SourceLine (see page 3460)	This is SourceLine, a member of class TAbstractScriptError.

14.1.3.3.1 TAbstractScriptError.CharPos Property

Pascal

```
property CharPos: Integer;
```

C++

```
__property int CharPos;
```

Description

This is CharPos, a member of class TAbstractScriptError.

14.1.3.3.2 TAbstractScriptError.Description Property

Pascal

```
property Description: string;
```

C++

```
__property UnicodeString Description;
```

Description

This is Description, a member of class TAbstractScriptError.

14.1.3.3.3 TAbstractScriptError.FileName Property

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

This is FileName, a member of class TAbstractScriptError.

14.1.3.3.4 TAbstractScriptError.Line Property

Pascal

```
property Line: Integer;
```

C++

```
__property int Line;
```

Description

This is Line, a member of class TAbstractScriptError.

14.1.3.3.5 TAbstractScriptError.SourceLine Property

Pascal

```
property SourceLine: string;
```

C++

```
__property UnicodeString SourceLine;
```

Description

This is SourceLine, a member of class TAbstractScriptError.

14.1.4 TAbstractScriptErrors Class

Class Hierarchy

Pascal

```
TAbstractScriptErrors = class(TObject);
```

C++

```
class TAbstractScriptErrors : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptErrors.

14.1.4.1 TAbstractScriptErrors Members

The following tables list the members exposed by TAbstractScriptErrors.

Protected Methods

	Name	Description
	GetError (see page 3461)	This is GetError, a member of class TAbstractScriptErrors.
	GetErrorCount (see page 3462)	This is GetErrorCount, a member of class TAbstractScriptErrors.

Public Methods

	Name	Description
	Add (see page 3461)	This is Add, a member of class TAbstractScriptErrors.

Public Properties

	Name	Description
	Count (see page 3462)	This is Count, a member of class TAbstractScriptErrors.
	Errors (see page 3462)	This is Errors, a member of class TAbstractScriptErrors.

14.1.4.2 TAbstractScriptErrors Methods

The methods of the TAbstractScriptErrors class are listed here.

Protected Methods

	Name	Description
	GetError (see page 3461)	This is GetError, a member of class TAbstractScriptErrors.
	GetErrorCount (see page 3462)	This is GetErrorCount, a member of class TAbstractScriptErrors.

Public Methods

	Name	Description
	Add (see page 3461)	This is Add, a member of class TAbstractScriptErrors.

14.1.4.2.1 TAbstractScriptErrors.Add Method

Pascal

```
procedure Add(const AError: TAbstractScriptError); virtual; abstract;
```

C++

```
virtual __fastcall Add(const TAbstractScriptError AError) = 0;
```

Description

This is Add, a member of class TAbstractScriptErrors.

14.1.4.2.2 TAbstractScriptErrors.GetError Method

Pascal

```
function GetError(I: Integer): TAbstractScriptError; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptError GetError(int I) = 0;
```

Description

This is GetError, a member of class TAbstractScriptErrors.

14.1.4.2.3 TAbstractScriptErrors.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetErrorCount() = 0;
```

Description

This is GetErrorCount, a member of class TAbstractScriptErrors.

14.1.4.3 TAbstractScriptErrors Properties

The properties of the TAbstractScriptErrors class are listed here.

Public Properties

	Name	Description
	Count (see page 3462)	This is Count, a member of class TAbstractScriptErrors.
	Errors (see page 3462)	This is Errors, a member of class TAbstractScriptErrors.

14.1.4.3.1 TAbstractScriptErrors.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAbstractScriptErrors.

14.1.4.3.2 TAbstractScriptErrors.Errors Property

Pascal

```
property Errors [I: Integer]: TAbstractScriptError;
```

C++

```
__property TAbstractScriptError Errors[int I];
```

Description

This is Errors, a member of class TAbstractScriptErrors.

14.1.5 TAbstractScriptProducer Class

Class Hierarchy

Pascal

```
TAbstractScriptProducer = class(TInterfacedObject);
```

C++

```
class TAbstractScriptProducer : public TInterfacedObject;
```

File

HTTPProd

14.1.5.1 TAbstractScriptProducer Members

The following tables list the members exposed by TAbstractScriptProducer.

Public Methods

	Name	Description
	Create (see page 3463)	Constructs an object and initializes its data before the object is first used.

14.1.5.2 TAbstractScriptProducer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebModuleContext: TWebModuleContext; AStripParamQuotes: Boolean;
AHandleTag: THandleTagProc; const AScriptEngine: string; ALocateFileService:
ILocateFileService); virtual;
```

C++

```
virtual __fastcall TAbstractScriptProducer(TWebModuleContext AWebModuleContext, Boolean
AStripParamQuotes, THandleTagProc AHandleTag, const UnicodeString AScriptEngine,
ILocateFileService ALocateFileService);
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

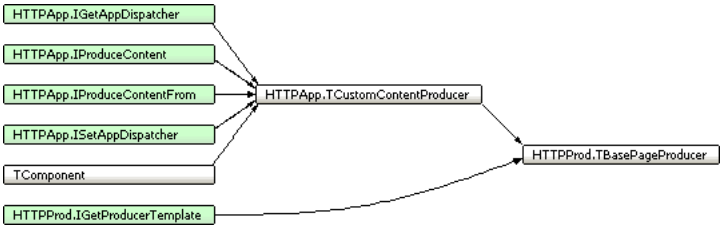
System::TObject::Destroy

new

14.1.6 TBasePageProducer Class

TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.

Class Hierarchy



Pascal

```
TBasePageProducer = class(TCustomContentProducer, IGetProducerTemplate);
```

C++

```
class TBasePageProducer : public TCustomContentProducer, public IGetProducerTemplate;
```

File

HTTPProd

Description

Do not create instances of TBasePageProducer. Instead, use TBasePageProducer as a base class when defining a page producer. TBasePageProducer has two descendants,

TCustomPageProducer (see page 3473), which is the base class for page producers that produce content based primarily on the content of HTTP messages and an OnHTML event handler. Examples include TPageProducer (see page 3496) and TDataSetPageProducer.

TComponentsPageProducer, which is the base class for page producers that assemble content from components that generate HTML. Examples include TAdapterPageProducer.

A page producer is an object that converts an HTML template into a string of HTML commands which can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page producer.

See Also

TDSTableProducer

TCustomPageProducer (see page 3473)

TPageProducer (see page 3496)

TWebResponse (see page 3389)

TDataSetPageProducer

TPageItemsProducer (see page 6394)

TInetXPageProducer (see page 5730)

TAdapterPageProducer

14.1.6.1 TBasePageProducer Members

The following tables list the members exposed by TBasePageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 3472)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Methods

	Name	Description
	DoTagEvent (see page 3468)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 3316)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetProducerTemplateFileName (see page 3468)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 3468)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 3469)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 3469)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 3469)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateStream (see page 3469)	This is GetTemplateStream, a member of class TBasePageProducer.
V	HandleScriptTag (see page 3469)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 3470)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 3470)	This is ImplHandleTag, a member of class TBasePageProducer.
V	Notification (see page 3316)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 3316)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3317)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 3317)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	ServerScriptFromStream (see page 3470)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAppDispatcher (see page 3317)	This is SetAppDispatcher, a member of class TCustomContentProducer.
V	UseScriptEngine (see page 3470)	This is UseScriptEngine, a member of class TBasePageProducer.

Public Methods

	Name	Description
V	Content (see page 3466)	Returns the content produced by the content producer.
V	ContentFromStream (see page 3467)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 3467)	Returns the content produced in response to an input string.
V	Create (see page 3466)	Creates an instance of TBasePageProducer (see page 3464).
	GetAppDispatcher (see page 3417)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	ProduceContent (see page 3421)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3422)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 3423)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	SetAppDispatcher (see page 3424)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

Public Properties

	Name	Description
R	Dispatcher (see page 3317)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.
R	DispatcherComponent (see page 3318)	Provides access to the dispatcher component.
	ScriptEngine (see page 3471)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 3471)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
R	WebModuleContext (see page 3471)	Provides access to the Web module context object.

14.1.6.2 TBasePageProducer.Create Constructor

Creates an instance of TBasePageProducer (see page 3464).

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TBasePageProducer (see page 3464).

AOwner is a component, typically a Web module, that is responsible for freeing the page producer.

See Also

Owner

14.1.6.3 TBasePageProducer Methods

The methods of the TBasePageProducer class are listed here.

Protected Methods

	Name	Description
	DoTagEvent (see page 3468)	This is DoTagEvent, a member of class TBasePageProducer.
	GetProducerTemplateFileName (see page 3468)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 3468)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 3469)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 3469)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 3469)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateStream (see page 3469)	This is GetTemplateStream, a member of class TBasePageProducer.
V	HandleScriptTag (see page 3469)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 3470)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 3470)	This is ImplHandleTag, a member of class TBasePageProducer.
	ServerScriptFromStream (see page 3470)	This is ServerScriptFromStream, a member of class TBasePageProducer.
V	UseScriptEngine (see page 3470)	This is UseScriptEngine, a member of class TBasePageProducer.

Public Methods

	Name	Description
V	Content (see page 3466)	Returns the content produced by the content producer.
V	ContentFromStream (see page 3467)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 3467)	Returns the content produced in response to an input string.

14.1.6.3.1 TBasePageProducer.Content Method

Returns the content produced by the content producer.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall UnicodeString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. TBasePageProducer (see page 3464) attempts to locate a default template and uses that to generate the resulting content.

See Also

ContentFromStream (see page 3467)

ContentFromString (see page 3467)

Content (see page 3404)

Specifying the HTML Template

14.1.6.3.2 TBasePageProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(Stream: TStream): string; override;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream Stream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter.

When the ScriptEngine (see page 3471) property is assigned, the page producer generates its content using a special script-producer component. When ScriptEngine (see page 3471) is not assigned, the page producer uses only the OnHTMLTag (see page 3472) event, plus any other built-in translations provided by a descendant class.

See Also

Content (see page 3466)

ContentFromString (see page 3467)

Content (see page 3404)

ScriptEngine (see page 3471)

OnHTMLTag (see page 3472)

Specifying the HTML Template

14.1.6.3.3 TBasePageProducer.ContentFromString Method

Returns the content produced in response to an input string.

Pascal

```
function ContentFromString(const S: string): string; override;
```

C++

```
virtual __fastcall UnicodeString ContentFromString(const UnicodeString S);
```

Description

Call ContentFromString to retrieve the content produced by the content producer based on the information provided by the S parameter.

When the ScriptEngine (see page 3471) property is assigned, the page producer generates its content using a special script-producer component. When ScriptEngine (see page 3471) is not assigned, the page producer uses only the OnHTMLTag (see page 3472) event, plus any other built-in translations provided by a descendant class.

See Also

Content (see page 3466)

ContentFromStream (see page 3467)

Content (see page 3404)

Chaining Page Producers Together

Specifying the HTML Template

14.1.6.3.4 TBasePageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); dynamic;
```

C++

```
__fastcall DoTagEvent(TTag Tag, const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceText);
```

Description

This is DoTagEvent, a member of class TBasePageProducer.

14.1.6.3.5 TBasePageProducer.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall UnicodeString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class TBasePageProducer.

14.1.6.3.6 TBasePageProducer.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class TBasePageProducer.

14.1.6.3.7 TBasePageProducer.GetScriptEngine Method

Pascal

```
function GetScriptEngine: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetScriptEngine();
```

Description

This is GetScriptEngine, a member of class TBasePageProducer.

14.1.6.3.8 TBasePageProducer.GetTagID Method

Pascal

```
function GetTagID(const TagString: string): TTag;
```

C++

```
__fastcall TTag GetTagID(const UnicodeString TagString);
```

Description

This is GetTagID, a member of class TBasePageProducer.

14.1.6.3.9 TBasePageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TBasePageProducer.

14.1.6.3.10 TBasePageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TBasePageProducer.

14.1.6.3.11 TBasePageProducer.HandleScriptTag Method

Pascal

```
function HandleScriptTag(const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandleScriptTag(const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceString);
```

Description

This is HandleScriptTag, a member of class TBasePageProducer.

14.1.6.3.12 TBasePageProducer.HandleTag Method

Pascal

```
function HandleTag(const TagString: string; TagParams: TStrings): string; virtual;
```

C++

```
virtual __fastcall UnicodeString HandleTag(const UnicodeString TagString, TStrings TagParams);
```

Description

This is HandleTag, a member of class TBasePageProducer.

14.1.6.3.13 TBasePageProducer.ImplHandleTag Method

Pascal

```
function ImplHandleTag(const TagString: string; TagParams: TStrings): string;
```

C++

```
__fastcall UnicodeString ImplHandleTag(const UnicodeString TagString, TStrings TagParams);
```

Description

This is ImplHandleTag, a member of class TBasePageProducer.

14.1.6.3.14 TBasePageProducer.ServerScriptFromStream Method

Pascal

```
function ServerScriptFromStream(Stream: TStream): string;
```

C++

```
__fastcall UnicodeString ServerScriptFromStream(TStream Stream);
```

Description

This is ServerScriptFromStream, a member of class TBasePageProducer.

14.1.6.3.15 TBasePageProducer.UseScriptEngine Method

Pascal

```
function UseScriptEngine: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseScriptEngine();
```

Description

This is UseScriptEngine, a member of class TBasePageProducer.

14.1.6.4 TBasePageProducer Properties

The properties of the TBasePageProducer class are listed here.

Public Properties

	Name	Description
	ScriptEngine (see page 3471)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 3471)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
R	WebModuleContext (see page 3471)	Provides access to the Web module context object.

14.1.6.4.1 TBasePageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property UnicodeString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag (see page 3472) event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 3472)

14.1.6.4.2 TBasePageProducer.StripParamQuotes Property

Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

Pascal

```
property StripParamQuotes: Boolean;
```

C++

```
__property Boolean StripParamQuotes;
```

Description

Set StripParamQuotes to indicate whether the page producer should strip quotation marks from the parameters of HTML-transparent tags before handling the tags. By setting StripParamQuotes to true, the page producer compensates for HTML editors that automatically insert quotation marks around the values of tag options.

14.1.6.4.3 TBasePageProducer.WebModuleContext Property

Provides access to the Web module context object.

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

Use WebModuleContext to get access to the TWebModuleContext (see page 3447) object. The TWebModuleContext (see page 3447) object maintains a list of named variables that the producer uses when translating HTML-transparent tags.

14.1.6.5 TBasePageProducer Events

The events of the TBasePageProducer class are listed here.

Protected Events

	Name	Description
	OnHTMLTag (see page 3472)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

14.1.6.5.1 TBasePageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names

Values

Customizing the InternetExpress Page Producer Template

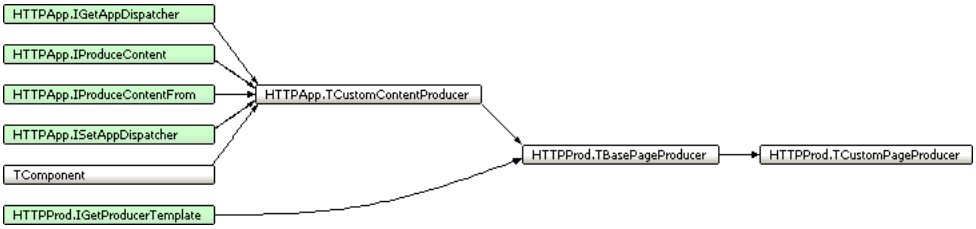
Chaining Page Producers Together

Converting HTML-transparent Tags

14.1.7 TCustomPageProducer Class

TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.

Class Hierarchy



Pascal

```
TCustomPageProducer = class(TBasePageProducer);
```

C++

```
class TCustomPageProducer : public TBasePageProducer;
```

File

HTTPProd

Description

Use TCustomPageProducer as a base class when defining an object that converts an HTML template into a string of HTML commands which can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page producer.

See Also

TDSTableProducer

TPageProducer (see page 3496)

14.1.7.1 TCustomPageProducer Members

The following tables list the members exposed by TCustomPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 3472)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Methods

	Name	Description
	DoTagEvent (see page 3468)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 3316)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetProducerTemplateFileName (see page 3468)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 3468)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 3469)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 3469)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 3475)	This is GetTemplateFileName, a member of class TCustomPageProducer.

	GetTemplateStream (see page 3476)	This is GetTemplateStream, a member of class TCustomPageProducer.
	HandleScriptTag (see page 3469)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 3476)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 3470)	This is ImplHandleTag, a member of class TBasePageProducer.
	Notification (see page 3316)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 3316)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3317)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 3317)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	ServerScriptFromStream (see page 3470)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAppDispatcher (see page 3317)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	UseScriptEngine (see page 3470)	This is UseScriptEngine, a member of class TBasePageProducer.

Public Methods

	Name	Description
	Content (see page 3314)	Returns the content produced by the content producer.
	Content (see page 3466)	Returns the content produced by the content producer.
	ContentFromStream (see page 3315)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 3467)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 3315)	Returns the content produced in response to an input string.
	ContentFromString (see page 3467)	Returns the content produced in response to an input string.
	Create (see page 3474)	Creates an instance of TCustomPageProducer (see page 3473).
	Destroy (see page 3475)	Destroys an instance of TCustomPageProducer (see page 3473).
	GetAppDispatcher (see page 3417)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetProducerTemplateFileName (see page 3505)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 3505)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	ProduceContent (see page 3421)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3422)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 3423)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	SetAppDispatcher (see page 3424)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

Public Properties

	Name	Description
	Dispatcher (see page 3317)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.
	DispatcherComponent (see page 3318)	Provides access to the dispatcher component.
	HTMLDoc (see page 3476)	Specifies the HTML template used to generate the value returned by the Content (see page 3466) method.
	HTMLFile (see page 3477)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 3466) method.
	ScriptEngine (see page 3471)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 3471)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	WebModuleContext (see page 3471)	Provides access to the Web module context object.

14.1.7.2 TCustomPageProducer.Create Constructor

Creates an instance of TCustomPageProducer (see page 3473).

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomPageProducer (see page 3473).

After calling the inherited constructor, Create allocates the TStrings object that implements the HTMLDoc (see page 3476) property.

See Also

HTMLDoc (see page 3476)

Create

14.1.7.3 TCustomPageProducer.Destroy Destructor

Destroys an instance of TCustomPageProducer (see page 3473).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomPageProducer();
```

Description

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

Destroy frees the TStrings object implements the HTMLDoc (see page 3476) property before calling the inherited destructor.

See Also

System::TObject::Free

HTMLDoc (see page 3476)

14.1.7.4 TCustomPageProducer Methods

The methods of the TCustomPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 3475)	This is GetTemplateFileName, a member of class TCustomPageProducer.
	GetTemplateStream (see page 3476)	This is GetTemplateStream, a member of class TCustomPageProducer.
	HandleTag (see page 3476)	This is HandleTag, a member of class TCustomPageProducer.

14.1.7.4.1 TCustomPageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TCustomPageProducer.

14.1.7.4.2 TCustomPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomPageProducer.

14.1.7.4.3 TCustomPageProducer.HandleTag Method

Pascal

```
function HandleTag(const TagString: string; TagParams: TStrings): string; override;
```

C++

```
virtual __fastcall UnicodeString HandleTag(const UnicodeString TagString, TStrings TagParams);
```

Description

This is HandleTag, a member of class TCustomPageProducer.

14.1.7.5 TCustomPageProducer Properties

The properties of the TCustomPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 3476)	Specifies the HTML template used to generate the value returned by the Content (see page 3466) method.
	HTMLFile (see page 3477)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 3466) method.

14.1.7.5.1 TCustomPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content (see page 3466) method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The

pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile (see page 3477) property.

See Also

Content (see page 3466)

HTMLFile (see page 3477)

Specifying the HTML Template

14.1.7.5.2 TCustomPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 3466) method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 3476) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 3466)

HTMLDoc (see page 3476)

Specifying the HTML Template

14.1.8 THTMLTableAttributes Class

THTMLTableAttributes represents the display attributes of an HTML table tag.

Class Hierarchy

Pascal

```
THTMLTableAttributes = class(THTMLTagAttributes);
```

C++

```
class THTMLTableAttributes : public THTMLTagAttributes;
```

File

HTTPProd

Description

Use THTMLTableAttributes to describe the desired appearance of an HTML table. The properties of the THTMLTableAttributes object correspond to the options that can be added to the HTML table tag. When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 3313)

TDSTableProducer

THTMLTableCellAttributes (see page 3484)

THTMLTableColumn

THTMLTableHeaderAttributes (see page 3488)

THTMLTableRowAttributes (see page 3491)

Specifying the Table Attributes

14.1.8.1 THTMLTableAttributes Members

The following tables list the members exposed by THTMLTableAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 3480)	Copies the properties of a THTMLTableAttributes (see page 3478) object to another THTMLTableAttributes (see page 3478) object.
	Changed (see page 3494)	Generates an OnChange (see page 3495) event.

Public Events

	Name	Description
	OnChange (see page 3495)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	Create (see page 3479)	Creates an instance of THTMLTableAttributes (see page 3478).
	RestoreDefaults (see page 3480)	Changes all properties of the table attributes object back to their initial settings.

Public Properties

	Name	Description
	Producer (see page 3495)	Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 3481)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 3482)	Specifies the background color of the HTML table.
	Border (see page 3482)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 3483)	Specifies the amount of space to leave around the contents of each cell in the HTML table.
	CellSpacing (see page 3483)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 3494)	Represents a tag option and its value in an HTML tag.
	Width (see page 3483)	Specifies the width of the entire HTML table.

14.1.8.2 THTMLTableAttributes.Create Constructor

Creates an instance of THTMLTableAttributes (see page 3478).

Pascal

```
constructor Create(Producer: TCustomContentProducer);
```

C++

```
__fastcall THTMLTableAttributes(TCustomContentProducer Producer);
```

Description

Call Create to create and initialize a THTMLTableAttributes (see page 3478) object. The Producer (see page 3495) parameter specifies the HTML producer that is using the THTMLTableAttributes (see page 3478) object to specify the attributes of an HTML table that makes up part of its output.

After calling the inherited constructor, Create initializes

Width (see page 3483) to a value of 100.

Border (see page 3482) to a value of -1.

CellPadding (see page 3483) to a value of -1.

CellSpacing (see page 3483) to a value of -1.

See Also

Border (see page 3482)

CellPadding (see page 3483)

CellSpacing (see page 3483)

Create (see page 3493)

Width (see page 3483)

14.1.8.3 THTMLTableAttributes Methods

The methods of the THTMLTableAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 3480)	Copies the properties of a THTMLTableAttributes (see page 3478) object to another THTMLTableAttributes (see page 3478) object.

Public Methods

	Name	Description
	RestoreDefaults (see page 3480)	Changes all properties of the table attributes object back to their initial settings.

14.1.8.3.1 THTMLTableAttributes.AssignTo Method

Copies the properties of a THTMLTableAttributes (see page 3478) object to another THTMLTableAttributes (see page 3478) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications cannot call the protected AssignTo method. To assign the properties of a THTMLTableAttributes (see page 3478) object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableAttributes (see page 3478) object as the source of the assignment, the AssignTo method of the THTMLTableAttributes (see page 3478) object is called. AssignTo checks whether the Dest parameter is another THTMLTableAttributes (see page 3478) object, and if so, copies the values of the Align (see page 3481), Border (see page 3482), CellSpacing (see page 3483), CellPadding (see page 3483), and Width (see page 3483) properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom (see page 3494) property.

See Also

Align (see page 3481)

Border (see page 3482)

CellPadding (see page 3483)

CellSpacing (see page 3483)

Assign

Width (see page 3483)

14.1.8.3.2 THTMLTableAttributes.RestoreDefaults Method

Changes all properties of the table attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table attributes either programmatically, or using the response editor at design time. RestoreAttributes sets

Align (see page 3481) to haDefault. (Delphi) or haDefault; (C++)

Border (see page 3482), CellPadding (see page 3483), and CellSpacing (see page 3483) to –1. (Delphi) or -1; (C++)

Custom (see page 3494) to an empty string.

Width (see page 3483) to 100. (Delphi) or 100; (C++)

See Also

Align (see page 3481)

Border (see page 3482)

CellPadding (see page 3483)

CellSpacing (see page 3483)

Custom (see page 3494)

Width (see page 3483)

14.1.8.4 THTMLTableAttributes Properties

The properties of the THTMLTableAttributes class are listed here.

Published Properties

	Name	Description
	Align (see page 3481)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 3482)	Specifies the background color of the HTML table.
	Border (see page 3482)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 3483)	Specifies the amount of space to leave around the contents of each cell in the HTML table.
	CellSpacing (see page 3483)	Specifies the amount of space to leave between cells in the HTML table.
	Width (see page 3483)	Specifies the width of the entire HTML table.

14.1.8.4.1 THTMLTableAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the table within the HTML document, or within the cell of a surrounding HTML table object. Align indicates the desired value of the Align option of the HTML table tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the HTML table

where it wants.

See Also

CellPadding (see page 3483)

Width (see page 3483)

14.1.8.4.2 THTMLTableAttributes.BgColor Property

Specifies the background color of the HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuschia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Use the BgColor property of the HTML attributes objects for the pieces of the table for more involved color effects.

See Also

BgColor

BgColor (see page 3487)

14.1.8.4.3 THTMLTableAttributes.Border Property

Specifies the thickness of the border drawn around the HTML table.

Pascal

```
property Border: Integer;
```

C++

```
__property int Border;
```

Description

Set Border to specify whether a border should be drawn around the table, and if so, how thick it should be. The thickness specified by any particular value of border depends on the web browser that views the HTML table. For any browser, however, larger numbers indicate thicker borders. A value of -1 indicates that no border should be drawn around the table.

See Also

CellSpacing (see page 3483)

14.1.8.4.4 THTMLTableAttributes.CellPadding Property

Specifies the amount of space to leave around the contents of each cell in the HTML table.

Pascal

```
property CellPadding: Integer;
```

C++

```
__property int CellPadding;
```

Description

Set CellPadding to specify how many pixels of whitespace should appear between a cell's contents and the edges of the cell. A value of -1 indicates that the web browser should decide how to pad the cells of the HTML table.

See Also

CellSpacing (see page 3483)

14.1.8.4.5 THTMLTableAttributes.CellSpacing Property

Specifies the amount of space to leave between cells in the HTML table.

Pascal

```
property CellSpacing: Integer;
```

C++

```
__property int CellSpacing;
```

Description

Set CellSpacing to specify how many pixels of whitespace should appear between the cells in the table. A value of -1 indicates that the web browser should decide how to separate the cells of the HTML table.

Note: CellSpacing refers to the whitespace between cells in an HTML table, not the thickness of the lines drawn to separate rows and columns.

See Also

Border (see page 3482)

CellPadding (see page 3483)

14.1.8.4.6 THTMLTableAttributes.Width Property

Specifies the width of the entire HTML table.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Set Width to indicate the desired width of the HTML table as a percentage of the width of the web browser window. If the value of Width is too small for the contents of the table, the web browser will ignore this property.

The default value of Width is 100, meaning the table will span the entire browser window. In this case, the value of the Align (

see page 3481) property is meaningless.

See Also

Align (see page 3481)

CellSpacing (see page 3483)

14.1.9 THTMLTableCellAttributes Class

THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.

Class Hierarchy

Pascal

```
THTMLTableCellAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableCellAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableCellAttributes to describe the desired appearance of a cell in an HTML table. The properties of the THTMLTableCellAttributes object correspond to the options that can be added to the HTML table data tag (TD). When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 3313)

THTMLTableAttributes (see page 3478)

THTMLTableColumn

THTMLTableHeaderAttributes (see page 3488)

THTMLTableRowAttributes (see page 3491)

14.1.10 THTMLTableElementAttributes Class

THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.

Class Hierarchy

Pascal

```
THTMLTableElementAttributes = class(THTMLTagAttributes);
```

C++

```
class THTMLTableElementAttributes : public THTMLTagAttributes;
```

File

HTTPProd

Description

Use the properties of THTMLTableElementAttributes to describe the desired appearance of a piece of an HTML table. The properties of THTMLTableElementAttributes correspond to the common options that can be added to the tags for the parts of the HTML table. When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 3313)

TDSTableProducer

THTMLTableCellAttributes (see page 3484)

THTMLTableHeaderAttributes (see page 3488)

THTMLTableRowAttributes (see page 3491)

14.1.10.1 THTMLTableElementAttributes Members

The following tables list the members exposed by THTMLTableElementAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 3486)	Copies the properties of a THTMLTableElementAttributes (see page 3484) object to another THTMLTableElementAttributes (see page 3484) object.
	Changed (see page 3494)	Generates an OnChange (see page 3495) event.

Public Events

	Name	Description
	OnChange (see page 3495)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	Create (see page 3493)	Creates an instance of THTMLTagAttributes (see page 3492).
	RestoreDefaults (see page 3486)	Changes all properties of the attributes object back to their initial settings.

Public Properties

	Name	Description
	Producer (see page 3495)	Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 3487)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 3487)	Specifies the background color of the HTML table element.
	Custom (see page 3494)	Represents a tag option and its value in an HTML tag.
	VAlign (see page 3488)	Specifies the vertical alignment of the contents of the HTML table element.

14.1.10.2 THTMLTableElementAttributes Methods

The methods of the THTMLTableElementAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 3486)	Copies the properties of a THTMLTableElementAttributes (see page 3484) object to another THTMLTableElementAttributes (see page 3484) object.

Public Methods

	Name	Description
	RestoreDefaults (see page 3486)	Changes all properties of the attributes object back to their initial settings.

14.1.10.2.1 THTMLTableElementAttributes.AssignTo Method

Copies the properties of a THTMLTableElementAttributes (see page 3484) object to another THTMLTableElementAttributes (see page 3484) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications can not call the protected AssignTo method. To assign the properties of a THTMLTableElementAttributes (see page 3484) object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableElementAttributes (see page 3484) object as the source of the assignment, the AssignTo method of the THTMLTableElementAttributes (see page 3484) object is called. AssignTo checks whether the Dest parameter is another THTMLTableElementAttributes (see page 3484) object, and if so, copies the values of the BgColor (see page 3487), Align (see page 3487), and VAlign (see page 3488) properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom (see page 3494) property.

See Also

Align (see page 3487)

Assign

VAlign (see page 3488)

14.1.10.2.2 THTMLTableElementAttributes.RestoreDefaults Method

Changes all properties of the attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table element attributes either programmatically, or using the Object Inspector at design time. RestoreAttributes sets

Align (see page 3487) to haDefault, and VAlign (see page 3488) to haVDefault.

BgColor (see page 3487) to an empty string.

Custom (see page 3494) to an empty string.

See Also

Align (see page 3487)

BgColor (see page 3487)

Custom (see page 3494)

VAlign (see page 3488)

14.1.10.3 THTMLTableElementAttributes Properties

The properties of the THTMLTableElementAttributes class are listed here.

Published Properties

	Name	Description
	Align (see page 3487)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 3487)	Specifies the background color of the HTML table element.
	VAlign (see page 3488)	Specifies the vertical alignment of the contents of the HTML table element.

14.1.10.3.1 THTMLTableElementAttributes.Align Property

Specifies the horizontal alignment of the contents of the HTML table element.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired horizontal alignment of the contents of the cell or cells indicated by the table element. Align indicates the desired value of the Align option of the HTML tag for the table element.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the contents of the table element wherever it wishes.

The Align property of table data cells or header cells overrides the Align property of the table row in which they appear.

See Also

VAlign (see page 3488)

14.1.10.3.2 THTMLTableElementAttributes.BgColor Property

Specifies the background color of the HTML table element.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table element. The value of BgColor is the name of color. This can be one of the following strings:

Use the BgColor property of the THTMLTableAttributes (see page 3478) object to set a background color for the entire table. Use the BgColor property of the table element attributes object to make a table element stand out from the rest of the table.

See Also

BgColor (see page 3482)

14.1.10.3.3 THTMLTableElementAttributes.VAlign Property

Specifies the vertical alignment of the contents of the HTML table element.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Set VAlign to indicate the desired vertical alignment of the contents of the HTML table element. VAlign indicates the desired value of the VAlign option of the HTML tag for the table element.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document positions the contents of cells in the table element wherever it wishes vertically.

The VAlign property of table data cells or header cells overrides the VAlign property of the table row in which they appear.

See Also

Align (see page 3487)

14.1.11 THTMLTableHeaderAttributes Class

THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.

Class Hierarchy

Pascal

```
THTMLTableHeaderAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableHeaderAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableHeaderAttributes to describe the desired appearance of a header cell in an HTML table. The properties of the THTMLTableHeaderAttributes object correspond to the options that can be added to the HTML header tag (TH). When the attributes of all the parts of the HTML table have been specified, the HTML content producer creates the HTML commands to specify the table as described.

See Also

THTMLTableCellAttributes (see page 3484)

14.1.11.1 THTMLTableHeaderAttributes Members

The following tables list the members exposed by THTMLTableHeaderAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 3490)	Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.
	Changed (see page 3494)	Generates an OnChange (see page 3495) event.

Public Events

	Name	Description
	OnChange (see page 3495)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	Create (see page 3493)	Creates an instance of THTMLTagAttributes (see page 3492).
	RestoreDefaults (see page 3490)	Changes all properties of the attributes object back to their initial settings.

Public Properties

	Name	Description
	Producer (see page 3495)	Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 3487)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 3487)	Specifies the background color of the HTML table element.
	Caption (see page 3491)	Represents the text that appears in the header cell.
	Custom (see page 3494)	Represents a tag option and its value in an HTML tag.
	VAlign (see page 3488)	Specifies the vertical alignment of the contents of the HTML table element.

14.1.11.2 THTMLTableHeaderAttributes Methods

The methods of the THTMLTableHeaderAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 3490)	Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.

Public Methods

	Name	Description
	RestoreDefaults (see page 3490)	Changes all properties of the attributes object back to their initial settings.

14.1.11.2.1 THTMLTableHeaderAttributes.AssignTo Method

Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications can not call the protected AssignTo method. To assign the properties of a THTMLTableElementAttributes object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableElementAttributes object as the source of the assignment, the AssignTo method of the THTMLTableElementAttributes object is called. AssignTo checks whether the Dest parameter is another THTMLTableElementAttributes object, and if so, copies the values of the BgColor, Align, and VAlign properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom property.

See Also

Align (see page 3487)

Assign

VAlign (see page 3488)

14.1.11.2.2 THTMLTableHeaderAttributes.RestoreDefaults Method

Changes all properties of the attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table element attributes either programmatically, or using the Object Inspector at design time. RestoreAttributes sets

Align (see page 3487) to haDefault, and VAlign (see page 3488) to haVDefault.

BgColor (see page 3487) to an empty string.

Caption (see page 3491) to an empty string.

Custom (see page 3494) to an empty string.

See Also

Align (see page 3487)

BgColor (see page 3487)

Caption (see page 3491)

Custom (see page 3494)

VAlign (see page 3488)

14.1.11.3 THTMLTableHeaderAttributes Properties

The properties of the THTMLTableHeaderAttributes class are listed here.

Published Properties

	Name	Description
	Caption (see page 3491)	Represents the text that appears in the header cell.

14.1.11.3.1 THTMLTableHeaderAttributes.Caption Property

Represents the text that appears in the header cell.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Set the Caption property to provide a caption for the column that contains the header cell. If the header cell is used by a dataset table producer or a query table producer, the Caption property overrides the field name that would label the column by default.

See Also

TDataSetTableProducer

TQueryTableProducer

TSQLQueryTableProducer

14.1.12 THTMLTableRowAttributes Class

THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.

Class Hierarchy

Pascal

```
THTMLTableRowAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableRowAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableRowAttributes to describe the desired appearance of a row of an HTML table. The properties of the THTMLTableRowAttributes object correspond to the options that can be added to the HTML row tag (TR). When the attributes of all the parts of the HTML table have been specified, the content producer creates the HTML commands to specify the table as described.

See Also

TDSTableProducer

THTMLTableHeaderAttributes (see page 3488)

Specifying the Row Attributes

14.1.13 THTMLTagAttributes Class

THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.

Class Hierarchy

Pascal

```
THTMLTagAttributes = class(TPersistent);
```

C++

```
class THTMLTagAttributes : public TPersistent;
```

File

HTTPProd

Description

Descendants of TCustomContentProducer (see page 3313) use THTMLTagAttributes objects to encapsulate the options added to the HTML tags in the HTML commands that they produce. When the TCustomContentProducer (see page 3313) object generates HTML commands, it looks up the tag attributes represented by the properties of its tag attributes objects, and generates the appropriate tag options.

See Also

TCustomContentProducer (see page 3313)

14.1.13.1 THTMLTagAttributes Members

The following tables list the members exposed by THTMLTagAttributes.

Protected Methods

	Name	Description
	Changed (see page 3494)	Generates an OnChange (see page 3495) event.

Public Events

	Name	Description
	OnChange (see page 3495)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	Create (see page 3493)	Creates an instance of THTMLTagAttributes (see page 3492).
V	RestoreDefaults (see page 3494)	Changes all properties of the tag attributes object back to their initial settings.

Public Properties

	Name	Description
R	Producer (see page 3495)	Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Custom (see page 3494)	Represents a tag option and its value in an HTML tag.

14.1.13.2 THTMLTagAttributes.Create Constructor

Creates an instance of THTMLTagAttributes (see page 3492).

Pascal

```
constructor Create(Producer: TCustomContentProducer);
```

C++

```
__fastcall THTMLTagAttributes(TCustomContentProducer Producer);
```

Description

Call Create to create and initialize an instance of THTMLTagAttributes (see page 3492) at runtime. After calling the inherited constructor, Create sets the Producer (see page 3495) property to the value of the Producer (see page 3495) parameter.

See Also

Producer (see page 3495)

System::TObject::Create

14.1.13.3 THTMLTagAttributes Methods

The methods of the THTMLTagAttributes class are listed here.

Protected Methods

	Name	Description
	Changed (see page 3494)	Generates an OnChange (see page 3495) event.

Public Methods

	Name	Description
V	RestoreDefaults (see page 3494)	Changes all properties of the tag attributes object back to their initial settings.

14.1.13.3.1 THTMLTagAttributes.Changed Method

Generates an OnChange (see page 3495) event.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Call Changed from a descendant of THTMLTagAttributes (see page 3492) after a tag option introduced by that descendant changes. Changed is exposed as a protected method so that component writers can create descendants of THTMLTagAttributes (see page 3492) which correctly trigger the OnChange (see page 3495) event handler.

See Also

OnChange (see page 3495)

14.1.13.3.2 THTMLTagAttributes.RestoreDefaults Method

Changes all properties of the tag attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; virtual;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the tag attributes either programmatically, or at design time. RestoreAttributes sets the Custom (see page 3494) property to an empty string.

See Also

Custom (see page 3494)

14.1.13.4 THTMLTagAttributes Properties

The properties of the THTMLTagAttributes class are listed here.

Public Properties

	Name	Description
	Producer (see page 3495)	Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Custom (see page 3494)	Represents a tag option and its value in an HTML tag.

14.1.13.4.1 THTMLTagAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options to be added to an HTML tag. Each option and its value should have the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to merge a cell in an HTML table with the three adjacent cells to the left and below, the table data tag for the cell requires the ROWSPAN and COLSPAN options to be set. The value of Custom to specify these values would be the string

```
ROWSPAN=2 COLSPAN=2
```

The HTML producer inserts the Custom string into the HTML tag when it generates its Content.

See Also

Producer (see page 3495)

Content (see page 3314)

14.1.13.4.2 THTMLTagAttributes.Producer Property

Indicates the content producer that uses the THTMLTagAttributes (see page 3492) object to represent the options for one of the HTML tags in the content it produces.

Pascal

```
property Producer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer Producer;
```

Description

Use Producer to gain access to the HTML producer object that is using the THTMLTagAttributes (see page 3492) object. The THTMLTagAttributes (see page 3492) object can use the Producer property in an OnChange (see page 3495) event handler to notify the TCustomContentProducer (see page 3313) object of changes to the attributes it represents.

See Also

OnChange (see page 3495)

14.1.13.5 THTMLTagAttributes Events

The events of the THTMLTagAttributes class are listed here.

Public Events

	Name	Description
	OnChange (see page 3495)	Occurs immediately after a tag attributes property changes.

14.1.13.5.1 THTMLTagAttributes.OnChange Event

Occurs immediately after a tag attributes property changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever any of the properties of the tag attributes object changes. When OnChange occurs, the tag attributes object already has the new property setting. Use an OnChange event handler to inform the HTML content producer whenever the attributes of a tag change so that the content producer can update the content it is producing.

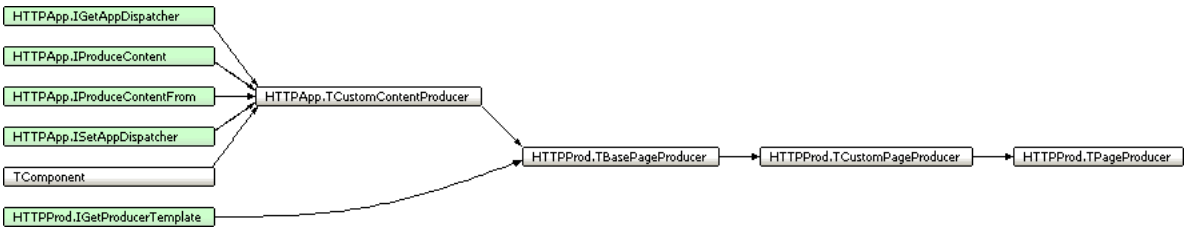
See Also

Producer (see page 3495)

14.1.14 TPageProducer Class

TPageProducer produces a string of HTML commands based on an input template.

Class Hierarchy



Pascal

```
TPageProducer = class(TCustomPageProducer);
```

C++

```
class TPageProducer : public TCustomPageProducer;
```

File

HTTPProd

Description

Use TPageProducer to convert an HTML template into a string of HTML commands that can be interpreted by a client application such as a Web browser. In Web Broker applications, HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the OnHTMLTag (see page 3500) event. WebSnap application page templates often include server-side scripting (such as JScript or JavaScript) to dynamically generate content.

See Also

TDataSetTableProducer

TSQLQueryTableProducer

TQueryTableProducer

TDataSetPageProducer

TAdapterPageProducer

The Web Module

Using Page Producer Components

14.1.14.1 TPageProducer Members

The following tables list the members exposed by TPageProducer.

Protected Methods

	Name	Description
	DoTagEvent (see page 3468)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 3316)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetProducerTemplateFileName (see page 3468)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 3468)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
	GetScriptEngine (see page 3469)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 3469)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 3469)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateFileName (see page 3475)	This is GetTemplateFileName, a member of class TCustomPageProducer.
	GetTemplateStream (see page 3469)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTemplateStream (see page 3476)	This is GetTemplateStream, a member of class TCustomPageProducer.
	HandleScriptTag (see page 3469)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 3470)	This is HandleTag, a member of class TBasePageProducer.
	HandleTag (see page 3476)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 3470)	This is ImplHandleTag, a member of class TBasePageProducer.
	Notification (see page 3316)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 3316)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3317)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 3317)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	ServerScriptFromStream (see page 3470)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAppDispatcher (see page 3317)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	UseScriptEngine (see page 3470)	This is UseScriptEngine, a member of class TBasePageProducer.

Public Methods

	Name	Description
	Content (see page 3314)	Returns the content produced by the content producer.
	Content (see page 3466)	Returns the content produced by the content producer.
	ContentFromStream (see page 3315)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 3467)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 3315)	Returns the content produced in response to an input string.
	ContentFromString (see page 3467)	Returns the content produced in response to an input string.
	Create (see page 3466)	Creates an instance of TBasePageProducer (see page 3464).
	Create (see page 3474)	Creates an instance of TCustomPageProducer (see page 3473).
	Destroy (see page 3475)	Destroys an instance of TCustomPageProducer (see page 3473).
	GetAppDispatcher (see page 3417)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetProducerTemplateFileName (see page 3505)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 3505)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	ProduceContent (see page 3421)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 3422)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 3423)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	SetAppDispatcher (see page 3424)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

Public Properties

	Name	Description
	Dispatcher (see page 3317)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 3313) object.
	DispatcherComponent (see page 3318)	Provides access to the dispatcher component.
	WebModuleContext (see page 3471)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnHTMLTag (see page 3500)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	HTMLDoc (see page 3498)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 3499)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	ScriptEngine (see page 3500)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 3500)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

14.1.14.2 TPageProducer Properties

The properties of the TPageProducer class are listed here.

Published Properties

	Name	Description
	HTMLDoc (see page 3498)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 3499)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	ScriptEngine (see page 3500)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 3500)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

14.1.14.2.1 TPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true

HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 3466)

HTMLFile (see page 3477)

Specifying the HTML Template

14.1.14.2.2 TPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 3466)

HTMLDoc (see page 3476)

Specifying the HTML Template

14.1.14.2.3 TPageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property UnicodeString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 3472)

14.1.14.2.4 TPageProducer.StripParamQuotes Property

Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

Pascal

```
property StripParamQuotes: Boolean;
```

C++

```
__property Boolean StripParamQuotes;
```

Description

Set StripParamQuotes to indicate whether the page producer should strip quotation marks from the parameters of HTML-transparent tags before handling the tags. By setting StripParamQuotes to true, the page producer compensates for HTML editors that automatically insert quotation marks around the values of tag options.

14.1.14.3 TPageProducer Events

The events of the TPageProducer class are listed here.

Published Events

	Name	Description
	OnHTMLTag (see page 3500)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

14.1.14.3.1 TPageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names

Values

Customizing the InternetExpress Page Producer Template

Chaining Page Producers Together

Converting HTML-transparent Tags

14.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDesignerFileManager (see page 3502)	This is class HTTPProd.IDesignerFileManager.
	IGetLocateFileService (see page 3503)	This is class HTTPProd.IGetLocateFileService.
	IGetProducerTemplate (see page 3504)	This is class HTTPProd.IGetProducerTemplate.
	ILocateFileService (see page 3505)	This is class HTTPProd.ILocateFileService.
	IScriptContext (see page 3506)	This is class HTTPProd.IScriptContext.
	IScriptProducer (see page 3508)	This is class HTTPProd.IScriptProducer.
	IWebVariableName (see page 3513)	This is class HTTPProd.IWebVariableName.
	IWebVariablesContainer (see page 3514)	This is class HTTPProd.IWebVariablesContainer.

14.2.1 IDesignerFileManager Interface

Class Hierarchy

HTTPProd.IDesignerFileManager

Pascal

```
IDesignerFileManager = interface;
```

C++

```
__interface IDesignerFileManager;
```

File

HTTPProd

Description

This is class HTTPProd.IDesignerFileManager.

14.2.1.1 IDesignerFileManager Members

The following tables list the members exposed by IDesignerFileManager.

Public Methods

	Name	Description
	GetStream (see page 3503)	This is GetStream, a member of class IDesignerFileManager.
	QualifyFileName (see page 3503)	This is QualifyFileName, a member of class IDesignerFileManager.

14.2.1.2 IDesignerFileManager Methods

The methods of the IDesignerFileManager class are listed here.

Public Methods

	Name	Description
	GetStream (see page 3503)	This is GetStream, a member of class IDesignerFileManager.
	QualifyFileName (see page 3503)	This is QualifyFileName, a member of class IDesignerFileManager.

14.2.1.2.1 IDesignerFileManager.GetStream Method

Pascal

```
function GetStream(const AFileName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetStream(const UnicodeString AFileName, Boolean AOwned);
```

Description

This is GetStream, a member of class IDesignerFileManager.

14.2.1.2.2 IDesignerFileManager.QualifyFileName Method

Pascal

```
function QualifyFileName(const AFileName: string): string;
```

C++

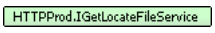
```
__fastcall UnicodeString QualifyFileName(const UnicodeString AFileName);
```

Description

This is QualifyFileName, a member of class IDesignerFileManager.

14.2.2 IGetLocateFileService Interface

Class Hierarchy



Pascal

```
IGetLocateFileService = interface;
```

C++

```
__interface IGetLocateFileService;
```

File

HTTPProd

Description

This is class HTTPProd.IGetLocateFileService.

14.2.2.1 IGetLocateFileService Members

The following tables list the members exposed by IGetLocateFileService.

Public Methods

	Name	Description
	GetLocateFileService (see page 3504)	This is GetLocateFileService, a member of class IGetLocateFileService.

14.2.2.2 IGetLocateFileService Methods

The methods of the IGetLocateFileService class are listed here.

Public Methods

	Name	Description
	GetLocateFileService (see page 3504)	This is GetLocateFileService, a member of class IGetLocateFileService.

14.2.2.2.1 IGetLocateFileService.GetLocateFileService Method

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

This is GetLocateFileService, a member of class IGetLocateFileService.

14.2.3 IGetProducerTemplate Interface

Class Hierarchy

HTTPProd.IGetProducerTemplate

Pascal

```
IGetProducerTemplate = interface;
```

C++

```
__interface IGetProducerTemplate;
```

File

HTTPProd

Description

This is class HTTPProd.IGetProducerTemplate.

14.2.3.1 IGetProducerTemplate Members

The following tables list the members exposed by IGetProducerTemplate.

Public Methods

	Name	Description
	GetProducerTemplateFileName (see page 3505)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.

	GetProducerTemplateStream (see page 3505)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

14.2.3.2 IGetProducerTemplate Methods

The methods of the IGetProducerTemplate class are listed here.

Public Methods

	Name	Description
	GetProducerTemplateFileName (see page 3505)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 3505)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

14.2.3.2.1 IGetProducerTemplate.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall UnicodeString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.

14.2.3.2.2 IGetProducerTemplate.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

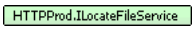
```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

14.2.4 ILocateFileService Interface

Class Hierarchy



Pascal

```
ILocateFileService = interface;
```

C++

```
__interface ILocateFileService;
```

File

HTTPProd

Description

This is class HTTPProd.ILocateFileService.

14.2.4.1 ILocateFileService Members

The following tables list the members exposed by ILocateFileService.

Public Methods

	Name	Description
	GetTemplateStream (see page 3506)	This is GetTemplateStream, a member of class ILocateFileService.

14.2.4.2 ILocateFileService Methods

The methods of the ILocateFileService class are listed here.

Public Methods

	Name	Description
	GetTemplateStream (see page 3506)	This is GetTemplateStream, a member of class ILocateFileService.

14.2.4.2.1 ILocateFileService.GetTemplateStream Method

Pascal

```
function GetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned: Boolean):
TStream;
```

C++

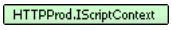
```
__fastcall TStream GetTemplateStream(TComponent * AComponent, UnicodeString AFileName, Boolean
AOwned);
```

Description

This is GetTemplateStream, a member of class ILocateFileService.

14.2.5 IScriptContext Interface

Class Hierarchy



Pascal

```
IScriptContext = interface;
```

C++

```
__interface IScriptContext;
```

File

HTTPProd

Description

This is class HTTPProd.IScriptContext.

14.2.5.1 IScriptContext Members

The following tables list the members exposed by IScriptContext.

Public Methods

	Name	Description
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.

Public Properties

	Name	Description
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.

14.2.5.2 IScriptContext Methods

The methods of the IScriptContext class are listed here.

Public Methods

	Name	Description
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.

14.2.5.2.1 IScriptContext.GetWebModuleContext Method

Pascal

```
function GetWebModuleContext: TWebModuleContext;
```

C++

```
__fastcall TWebModuleContext GetWebModuleContext();
```

Description

This is GetWebModuleContext, a member of class IScriptContext.

14.2.5.3 IScriptContext Properties

The properties of the IScriptContext class are listed here.

Public Properties

	Name	Description
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.

14.2.5.3.1 IScriptContext.WebModuleContext Property

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

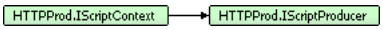
```
__property TWebModuleContext WebModuleContext;
```

Description

This is WebModuleContext, a member of class IScriptContext.

14.2.6 IScriptProducer Interface

Class Hierarchy



Pascal

```
IScriptProducer = interface(IScriptContext);
```

C++

```
__interface IScriptProducer : public IScriptContext;
```

File

HTTPProd

Description

This is class HTTPProd.IScriptProducer.

14.2.6.1 IScriptProducer Members

The following tables list the members exposed by IScriptProducer.

Public Methods

	Name	Description
	Evaluate (see page 3509)	This is Evaluate, a member of class IScriptProducer.
	GetContent (see page 3509)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 3509)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 3510)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 3510)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.
	HandleScriptError (see page 3510)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 3510)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 3510)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 3511)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 3511)	This is SetContent, a member of class IScriptProducer.
	Write (see page 3511)	This is the overview for the Write method overload.
	WriteItem (see page 3512)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
	Content (see page 3512)	This is Content, a member of class IScriptProducer.
R	Errors (see page 3512)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 3513)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 3513)	This is HTMLBlocks, a member of class IScriptProducer.
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.

14.2.6.2 IScriptProducer Methods

The methods of the IScriptProducer class are listed here.

Public Methods

	Name	Description
	Evaluate (see page 3509)	This is Evaluate, a member of class IScriptProducer.
	GetContent (see page 3509)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 3509)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 3510)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 3510)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	HandleScriptError (see page 3510)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 3510)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 3510)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 3511)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 3511)	This is SetContent, a member of class IScriptProducer.
	Write (see page 3511)	This is the overview for the Write method overload.
	WriteItem (see page 3512)	This is WriteItem, a member of class IScriptProducer.

14.2.6.2.1 IScriptProducer.Evaluate Method

Pascal

```
function Evaluate: string;
```

C++

```
__fastcall UnicodeString Evaluate();
```

Description

This is Evaluate, a member of class IScriptProducer.

14.2.6.2.2 IScriptProducer.GetContent Method

Pascal

```
function GetContent: string;
```

C++

```
__fastcall UnicodeString GetContent();
```

Description

This is GetContent, a member of class IScriptProducer.

14.2.6.2.3 IScriptProducer.GetErrors Method

Pascal

```
function GetErrors: TAbstractScriptErrors;
```

C++

```
__fastcall TAbstractScriptErrors GetErrors();
```

Description

This is GetErrors, a member of class IScriptProducer.

14.2.6.2.4 IScriptProducer.GetHTMLBlock Method

Pascal

```
function GetHTMLBlock(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetHTMLBlock(int I);
```

Description

This is GetHTMLBlock, a member of class IScriptProducer.

14.2.6.2.5 IScriptProducer.GetHTMLBlockCount Method

Pascal

```
function GetHTMLBlockCount: Integer;
```

C++

```
__fastcall int GetHTMLBlockCount();
```

Description

This is GetHTMLBlockCount, a member of class IScriptProducer.

14.2.6.2.6 IScriptProducer.HandleScriptError Method

Pascal

```
function HandleScriptError(const ScriptError: IUnknown): HRESULT;
```

C++

```
__fastcall HRESULT HandleScriptError(const IUnknown ScriptError);
```

Description

This is HandleScriptError, a member of class IScriptProducer.

14.2.6.2.7 IScriptProducer.ParseStream Method

Pascal

```
procedure ParseStream(Stream: TStream; Owned: Boolean = False);
```

C++

```
__fastcall ParseStream(TStream Stream, Boolean Owned = False);
```

Description

This is ParseStream, a member of class IScriptProducer.

14.2.6.2.8 IScriptProducer.ParseString Method

Pascal

```
procedure ParseString(const S: string);
```

C++

```
__fastcall ParseString(const UnicodeString S);
```

Description

This is ParseString, a member of class IScriptProducer.

14.2.6.2.9 IScriptProducer.ReplaceTags Method

Pascal

```
function ReplaceTags(const S: string): string;
```

C++

```
__fastcall UnicodeString ReplaceTags(const UnicodeString S);
```

Description

This is ReplaceTags, a member of class IScriptProducer.

14.2.6.2.10 IScriptProducer.SetContent Method

Pascal

```
procedure SetContent(const Value: string);
```

C++

```
__fastcall SetContent(const UnicodeString Value);
```

Description

This is SetContent, a member of class IScriptProducer.

14.2.6.2.11 Write Method

This is the overview for the Write method overload.

Overload List

	Name	Description
	IScriptProducer.Write (PWideChar, Integer) (see page 3511)	This is Write, a member of class IScriptProducer.
	IScriptProducer.Write (string) (see page 3511)	This is Write, a member of class IScriptProducer.

14.2.6.2.11.1 IScriptProducer.Write Method (PWideChar, Integer)

Pascal

```
procedure Write(const Value: PWideChar; ALength: Integer); overload;
```

C++

```
__fastcall Write(const PWideChar Value, int ALength);
```

Description

This is Write, a member of class IScriptProducer.

14.2.6.2.11.2 IScriptProducer.Write Method (string)

Pascal

```
procedure Write(const Value: string); overload;
```

C++

```
__fastcall Write(const UnicodeString Value);
```

Description

This is Write, a member of class IScriptProducer.

14.2.6.2.12 IScriptProducer.WriteItem Method

Pascal

```
procedure WriteItem(Index: Integer);
```

C++

```
__fastcall WriteItem(int Index);
```

Description

This is WriteItem, a member of class IScriptProducer.

14.2.6.3 IScriptProducer Properties

The properties of the IScriptProducer class are listed here.

Public Properties

	Name	Description
	Content (see page 3512)	This is Content, a member of class IScriptProducer.
R	Errors (see page 3512)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 3513)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 3513)	This is HTMLBlocks, a member of class IScriptProducer.

14.2.6.3.1 IScriptProducer.Content Property

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

This is Content, a member of class IScriptProducer.

14.2.6.3.2 IScriptProducer.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class IScriptProducer.

14.2.6.3.3 IScriptProducer.HTMLBlockCount Property

Pascal

```
property HTMLBlockCount: Integer;
```

C++

```
__property int HTMLBlockCount;
```

Description

This is HTMLBlockCount, a member of class IScriptProducer.

14.2.6.3.4 IScriptProducer.HTMLBlocks Property

Pascal

```
property HTMLBlocks [I: Integer]: string;
```

C++

```
__property UnicodeString HTMLBlocks[int I];
```

Description

This is HTMLBlocks, a member of class IScriptProducer.

14.2.7 IWebVariableName Interface

Class Hierarchy

HTTPProd.IWebVariableName

Pascal

```
IWebVariableName = interface;
```

C++

```
__interface IWebVariableName;
```

File

HTTPProd

Description

This is class HTTPProd.IWebVariableName.

14.2.7.1 IWebVariableName Members

The following tables list the members exposed by IWebVariableName.

Public Methods

	Name	Description
	GetVariableName (see page 3514)	This is GetVariableName, a member of class IWebVariableName.

Public Properties

	Name	Description
	VariableName (see page 3514)	This is VariableName, a member of class IWebVariableName.

14.2.7.2 IWebVariableName Methods

The methods of the IWebVariableName class are listed here.

Public Methods

	Name	Description
	GetVariableName (see page 3514)	This is GetVariableName, a member of class IWebVariableName.

14.2.7.2.1 IWebVariableName.GetVariableName Method

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

This is GetVariableName, a member of class IWebVariableName.

14.2.7.3 IWebVariableName Properties

The properties of the IWebVariableName class are listed here.

Public Properties

	Name	Description
	VariableName (see page 3514)	This is VariableName, a member of class IWebVariableName.

14.2.7.3.1 IWebVariableName.VariableName Property

Pascal

```
property VariableName: string;
```

C++

```
__property UnicodeString VariableName;
```

Description

This is VariableName, a member of class IWebVariableName.

14.2.8 IWebVariablesContainer Interface

Class Hierarchy

HTTPProd.IWebVariablesContainer

Pascal

```
IWebVariablesContainer = interface;
```

C++

```
__interface IWebVariablesContainer;
```

File

HTTPProd

Description

This is class HTTPProd.IWebVariablesContainer.

14.2.8.1 IWebVariablesContainer Members

The following tables list the members exposed by IWebVariablesContainer.

Public Methods

	Name	Description
	FindVariable (see page 3515)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 3516)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	GetVariable (see page 3516)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 3516)	This is GetVariableCount, a member of class IWebVariablesContainer.

Public Properties

	Name	Description
R	VariableCount (see page 3516)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 3517)	This is Variables, a member of class IWebVariablesContainer.

14.2.8.2 IWebVariablesContainer Methods

The methods of the IWebVariablesContainer class are listed here.

Public Methods

	Name	Description
	FindVariable (see page 3515)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 3516)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	GetVariable (see page 3516)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 3516)	This is GetVariableCount, a member of class IWebVariablesContainer.

14.2.8.2.1 IWebVariablesContainer.FindVariable Method

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

This is FindVariable, a member of class IWebVariablesContainer.

14.2.8.2.2 IWebVariablesContainer.FindVariableContainer Method

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

This is FindVariableContainer, a member of class IWebVariablesContainer.

14.2.8.2.3 IWebVariablesContainer.GetVariable Method

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

This is GetVariable, a member of class IWebVariablesContainer.

14.2.8.2.4 IWebVariablesContainer.GetVariableCount Method

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

This is GetVariableCount, a member of class IWebVariablesContainer.

14.2.8.3 IWebVariablesContainer Properties

The properties of the IWebVariablesContainer class are listed here.

Public Properties

	Name	Description
	VariableCount (see page 3516)	This is VariableCount, a member of class IWebVariablesContainer.
	Variables (see page 3517)	This is Variables, a member of class IWebVariablesContainer.

14.2.8.3.1 IWebVariablesContainer.VariableCount Property

Pascal

```
property VariableCount: Integer;
```

C++

```
__property int VariableCount;
```

Description

This is VariableCount, a member of class IWebVariablesContainer.

14.2.8.3.2 IWebVariablesContainer.Variables Property

Pascal

```
property Variables [I: Integer]: TComponent;
```

C++

```
__property TComponent * Variables[int I];
```

Description

This is Variables, a member of class IWebVariablesContainer.

14.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ContentFromScriptFile (see page 3518)	This is function HTTPProd.ContentFromScriptFile.
	ContentFromScriptStream (see page 3518)	This is function HTTPProd.ContentFromScriptStream.
	ContentFromStream (see page 3519)	This is function HTTPProd.ContentFromStream.
	ContentFromString (see page 3519)	This is function HTTPProd.ContentFromString.
	FindComponentWebModuleContext (see page 3519)	This is function HTTPProd.FindComponentWebModuleContext.
	GetEncodingOfStream (see page 3520)	This is function HTTPProd.GetEncodingOfStream.
	GetTagID (see page 3520)	This is function HTTPProd.GetTagID.

14.3.1 HTTPProd.ContentFromScriptFile Function

Pascal

```
function ContentFromScriptFile(const AFileName: TFileName; AWebModuleContext:
TWebModuleContext; AStripParamQuotes: Boolean; AHandleTag: THandleTagProc; AHandleScriptTag:
THandledTagProc; const AScriptEngine: string; ALocateFileService: ILocateFileService): string;
```

C++

```
UnicodeString ContentFromScriptFile(const TFileName AFileName, TWebModuleContext
AWebModuleContext, Boolean AStripParamQuotes, THandleTagProc AHandleTag, THandledTagProc
AHandleScriptTag, const UnicodeString AScriptEngine, ILocateFileService ALocateFileService);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromScriptFile.

14.3.2 HTTPProd.ContentFromScriptStream Function

Pascal

```
function ContentFromScriptStream(AStream: TStream; AWebModuleContext: TWebModuleContext;
AStripParamQuotes: Boolean; AHandleTag: THandleTagProc; AHandleScriptTag: THandledTagProc;
const AScriptEngine: string; ALocateFileService: ILocateFileService): string;
```

C++

```
UnicodeString ContentFromScriptStream(TStream AStream, TWebModuleContext AWebModuleContext,
Boolean AStripParamQuotes, THandleTagProc AHandleTag, THandledTagProc AHandleScriptTag, const
UnicodeString AScriptEngine, ILocateFileService ALocateFileService);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromScriptStream.

14.3.3 HTTPProd.ContentFromStream Function

Pascal

```
function ContentFromStream(AStream: TStream; AStripParamQuotes: Boolean; AHandleTag:
THandleTagProc; AHandledTag: THandledTagProc): string;
```

C++

```
UnicodeString ContentFromStream(TStream AStream, Boolean AStripParamQuotes, THandleTagProc
AHandleTag, THandledTagProc AHandledTag);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromStream.

14.3.4 HTTPProd.ContentFromString Function

Pascal

```
function ContentFromString(const AValue: string; AStripParamQuotes: Boolean; AHandleTag:
THandleTagProc; AHandledTag: THandledTagProc): string;
```

C++

```
UnicodeString ContentFromString(const UnicodeString AValue, Boolean AStripParamQuotes,
THandleTagProc AHandleTag, THandledTagProc AHandledTag);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromString.

14.3.5 HTTPProd.FindComponentWebModuleContext Function

Pascal

```
function FindComponentWebModuleContext(AComponent: TComponent): TWebModuleContext;
```

C++

```
TWebModuleContext FindComponentWebModuleContext(TComponent * AComponent);
```

File

HTTPProd

Description

This is function HTTPProd.FindComponentWebModuleContext.

14.3.6 HTTPProd.GetEncodingOfStream Function

Pascal

```
function GetEncodingOfStream(AStream: TStream; out ASignatureSize: Integer): TEncoding;
```

C++

```
TEncoding GetEncodingOfStream(TStream AStream, int ASignatureSize);
```

File

HTTPProd

Description

This is function HTTPProd.GetEncodingOfStream.

14.3.7 HTTPProd.GetTagID Function

Pascal

```
function GetTagID(const TagString: string): TTag;
```

C++

```
TTag GetTagID(const UnicodeString TagString);
```

File

HTTPProd

Description

This is function HTTPProd.GetTagID.

14.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	THTMLAlign (see page 3521)	This is record HTTPProd.THTMLAlign.
	THTMLVAlign (see page 3521)	This is record HTTPProd.THTMLVAlign.
	TTag (see page 3522)	This is record HTTPProd.TTag.

14.4.1 HTTPProd.THTMLAlign Enumeration

Pascal

```
THTMLAlign = (
  haDefault,
  haLeft,
  haRight,
  haCenter
);
```

C++

```
enum THTMLAlign {
  haDefault,
  haLeft,
  haRight,
  haCenter
};
```

File

HTTPProd

Description

This is record HTTPProd.THTMLAlign.

14.4.2 HTTPProd.THTMLVAlign Enumeration

Pascal

```
THTMLVAlign = (
  haVDefault,
  haTop,
  haMiddle,
  haBottom,
  haBaseline
);
```

C++

```
enum THTMLVAlign {
  haVDefault,
```

```
    haTop,
    haMiddle,
    haBottom,
    haBaseline
  };
```

File

HTTPProd

Description

This is record HTTPProd.THTMLVAlign.

14.4.3 HTTPProd.TTag Enumeration

Pascal

```
  TTag = (
    tgCustom,
    tgLink,
    tgImage,
    tgTable,
    tgImageMap,
    tgObject,
    tgEmbed
  );
```

C++

```
  enum TTag {
    tgCustom,
    tgLink,
    tgImage,
    tgTable,
    tgImageMap,
    tgObject,
    tgEmbed
  };
```

File

HTTPProd

Description

This is record HTTPProd.TTag.

14.5 Types

The following table lists types in this documentation.

Types

Name	Description
THandledTagProc (see page 3523)	This is type HTTPProd.THandledTagProc.
THandleTagProc (see page 3523)	This is type HTTPProd.THandleTagProc.
THTMLBgColor (see page 3524)	This is type HTTPProd.THTMLBgColor.
THTMLTagEvent (see page 3524)	This is type HTTPProd.THTMLTagEvent.
TScriptProducerClass (see page 3524)	This is type HTTPProd.TScriptProducerClass.

14.5.1 HTTPProd.THandledTagProc Type

Pascal

```
THandledTagProc = function (const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean of object;
```

C++

```
(const TagString: string; TagParams: TStrings; var ReplaceString: string): Boolean (
THandledTagProc)();
```

File

HTTPProd

Description

This is type HTTPProd.THandledTagProc.

14.5.2 HTTPProd.THandleTagProc Type

Pascal

```
THandleTagProc = function (const TagString: string; TagParams: TStrings): string of object;
```

C++

```
(const TagString: string; TagParams: TStrings): string ( THandleTagProc)();
```

File

HTTPProd

Description

This is type HTTPProd.THandleTagProc.

14.5.3 HTTPProd.THTMLBgColor Type

Pascal

```
THTMLBgColor = type string;
```

C++

```
type string THTMLBgColor;
```

File

HTTPProd

Description

This is type HTTPProd.THTMLBgColor.

14.5.4 HTTPProd.THTMLTagEvent Type

Pascal

```
THTMLTagEvent = procedure (Sender: TObject; Tag: TTag; const TagString: string; TagParams:
TStrings; var ReplaceText: string) of object;
```

C++

```
(Sender: TObject; Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string) ( THTMLTagEvent)();
```

File

HTTPProd

Description

This is type HTTPProd.THTMLTagEvent.

14.5.5 HTTPProd.TScriptProducerClass Type

Pascal

```
TScriptProducerClass = class of TAbstractScriptProducer;
```

C++

```
TAbstractScriptProducer TScriptProducerClass;
```

File

HTTPProd

Description

This is type HTTPProd.TScriptProducerClass.

14.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DesignerFileManager (see page 3525)	This is variable HTTPProd.DesignerFileManager.
ScriptEnginesList (see page 3525)	This is variable HTTPProd.ScriptEnginesList.

14.6.1 HTTPProd.DesignerFileManager Variable

Pascal

```
DesignerFileManager: IDesignerFileManager = nil;
```

C++

```
IDesignerFileManager DesignerFileManager = nil;
```

File

HTTPProd

Description

This is variable HTTPProd.DesignerFileManager.

14.6.2 HTTPProd.ScriptEnginesList Variable

Pascal

```
ScriptEnginesList: TAbstractScriptEnginesList = nil;
```

C++

```
TAbstractScriptEnginesList ScriptEnginesList = nil;
```

File

HTTPProd

Description

This is variable HTTPProd.ScriptEnginesList.

14.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
HTMLAlign (see page 3526)	This is constant HTTPProd.HTMLAlign.
HTMLVAlign (see page 3526)	This is constant HTTPProd.HTMLVAlign.

14.7.1 HTTPProd.HTMLAlign Constant

Pascal

```
HTMLAlign: array[THTMLAlign] of string = ('', ' Align="left"', ' Align="right"', '
Align="center"');
```

C++

```
array[THTMLAlign] of string HTMLAlign = ('', ' Align="left"', ' Align="right"', '
Align="center"');
```

File

HTTPProd

Description

This is constant HTTPProd.HTMLAlign.

14.7.2 HTTPProd.HTMLVAlign Constant

Pascal

```
HTMLVAlign: array[THTMLVAlign] of string = ('', ' VAlign="top"', ' VAlign="middle"', '
VAlign="bottom"', ' VAlign="baseline"');
```

C++

```
array[THTMLVAlign] of string HTMLVAlign = ('', ' VAlign="top"', ' VAlign="middle"', '
VAlign="bottom"', ' VAlign="baseline"');
```

File

HTTPProd

Description

This is constant HTTPProd.HTMLVAlign.

15 HTTPUtil Namespace

This is namespace HTTPUtil.

Topics

Name	Description
Interfaces (see page 3528)	The following table lists interfaces in this documentation.
Functions (see page 3537)	The following table lists functions in this documentation.

Functions

	Name	Description
	FirstDelimiter (see page 3537)	This is function HTTPUtil.FirstDelimiter.
	FirstDelimiter (see page 3537)	This is function HTTPUtil.FirstDelimiter.
	GetDefaultStreamLoader (see page 3538)	This is function HTTPUtil.GetDefaultStreamLoader.
	HTMLEscape (see page 3538)	This is function HTTPUtil.HTMLEscape.
	StartsWith (see page 3538)	This is function HTTPUtil.StartsWith.
	StringTokenizer (see page 3539)	This is function HTTPUtil.StringTokenizer.
	StringToStringArray (see page 3539)	This is function HTTPUtil.StringToStringArray.

Interfaces

	Name	Description
	IStreamLoader (see page 3528)	IStreamLoader is the interface for loading a WSDL document into a memory stream.
	IStringTokenizer (see page 3534)	A string tokenizer breaks a string into tokens.

15.1 Interfaces

The following table lists interfaces in this documentation.

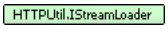
Interfaces

	Name	Description
	IStreamLoader (see page 3528)	IStreamLoader is the interface for loading a WSDL document into a memory stream.
	IStringTokenizer (see page 3534)	A string tokenizer breaks a string into tokens.

15.1.1 IStreamLoader Interface

IStreamLoader is the interface for loading a WSDL document into a memory stream.

Class Hierarchy



Pascal

```
IStreamLoader = interface;
```

C++

```
__interface IStreamLoader;
```

File

HTTPUtil

Description

IStreamLoader is an interface for objects that load WSDL documents. The properties specify information needed to connect to the server that hosts the WSDL document. In addition, IStreamLoader defines a Load (see page 3530) method, which loads a WSDL document into a memory stream. TWSDLItems uses the IStreamLoader interface to load its WSDL document.

Note: C++ method declarations that use IStreamLoader use the _di_IStreamLoader type instead. This type is a DelphiInterface wrapper around the IStreamLoader interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IStreamLoaders >  _di_IStreamLoader;
```

See Also

TWSDLItems

TMemoryStream

15.1.1.1 IStreamLoader Members

The following tables list the members exposed by IStreamLoader.

Public Methods

	Name	Description
	GetPassword (see page 3529)	Returns the value of the Password (see page 3532) property.
	GetProxy (see page 3530)	Returns the value of the Proxy (see page 3533) property.
	GetTimeout (see page 3530)	This is GetTimeout, a member of class IStreamLoader.
	GetUserName (see page 3530)	Returns the value of the UserName (see page 3533) property.
	Load (see page 3530)	Loads a WSDL document into a memory stream.
	SetPassword (see page 3531)	Sets the value of the Password (see page 3532) property.
	SetProxy (see page 3531)	Sets the value of the Proxy (see page 3533) property.
	SetTimeout (see page 3532)	This is SetTimeout, a member of class IStreamLoader.
	SetUserName (see page 3532)	Sets the value of the UserName (see page 3533) property.

Public Properties

	Name	Description
	Password (see page 3532)	Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.
	Proxy (see page 3533)	Lists any proxy servers that are required to connect to a server that hosts the WSDL document.
	Timeout (see page 3533)	This is Timeout, a member of class IStreamLoader.
	UserName (see page 3533)	Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

15.1.1.2 IStreamLoader Methods

The methods of the IStreamLoader class are listed here.

Public Methods

	Name	Description
	GetPassword (see page 3529)	Returns the value of the Password (see page 3532) property.
	GetProxy (see page 3530)	Returns the value of the Proxy (see page 3533) property.
	GetTimeout (see page 3530)	This is GetTimeout, a member of class IStreamLoader.
	GetUserName (see page 3530)	Returns the value of the UserName (see page 3533) property.
	Load (see page 3530)	Loads a WSDL document into a memory stream.
	SetPassword (see page 3531)	Sets the value of the Password (see page 3532) property.
	SetProxy (see page 3531)	Sets the value of the Proxy (see page 3533) property.
	SetTimeout (see page 3532)	This is SetTimeout, a member of class IStreamLoader.
	SetUserName (see page 3532)	Sets the value of the UserName (see page 3533) property.

15.1.1.2.1 IStreamLoader.GetPassword Method

Returns the value of the Password (see page 3532) property.

Pascal

```
function GetPassword: string;
```

C++

```
__fastcall UnicodeString GetPassword();
```

Description

GetPassword is the read implementation for the Password (see page 3532) property.

See Also

Password (see page 3532)

15.1.1.2.2 IStreamLoader.GetProxy Method

Returns the value of the Proxy (see page 3533) property.

Pascal

```
function GetProxy: string;
```

C++

```
__fastcall UnicodeString GetProxy();
```

Description

GetProxy is the read implementation for the Proxy (see page 3533) property.

See Also

Proxy (see page 3533)

15.1.1.2.3 IStreamLoader.GetTimeout Method

Pascal

```
function GetTimeout: Integer;
```

C++

```
__fastcall int GetTimeout();
```

Description

This is GetTimeout, a member of class IStreamLoader.

15.1.1.2.4 IStreamLoader.GetUserName Method

Returns the value of the UserName (see page 3533) property.

Pascal

```
function GetUserName: string;
```

C++

```
__fastcall UnicodeString GetUserName();
```

Description

GetUserName is the read implementation for the UserName (see page 3533) property.

See Also

UserName (see page 3533)

15.1.1.2.5 IStreamLoader.Load Method

Loads a WSDL document into a memory stream.

Pascal

```
procedure Load(const WSDLFileName: WideString; Stream: TMemoryStream);
```

C++

```
__fastcall Load(const BSTR WSDLFileName, TMemoryStream Stream);
```

Description

Load reads the specified WSDL document into the specified memory stream.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found. If WSDLFileName is an URL, you may need to use the UserName (see page 3533), Password (see page 3532), and Proxy (see page 3533) properties to indicate how the stream loader can connect to the host identified in that URL.

Stream is a memory stream to which the stream loader writes the contents of the WSDL document.

See Also

UserName (see page 3533)

Password (see page 3532)

Proxy (see page 3533)

15.1.1.2.6 IStreamLoader.SetPassword Method

Sets the value of the Password (see page 3532) property.

Pascal

```
procedure SetPassword(const Password: string);
```

C++

```
__fastcall SetPassword(const UnicodeString Password);
```

Description

SetPassword is the write implementation for the Password (see page 3532) property.

Password (see page 3532) is the password to use when connecting to the server that hosts the WSDL document.

See Also

Password (see page 3532)

15.1.1.2.7 IStreamLoader.SetProxy Method

Sets the value of the Proxy (see page 3533) property.

Pascal

```
procedure SetProxy(const Proxy: string);
```

C++

```
__fastcall SetProxy(const UnicodeString Proxy);
```

Description

SetProxy is the write implementation for the Proxy (see page 3533) property.

Proxy (see page 3533) lists any proxy servers to use when connecting to the server that hosts the WSDL document.

See Also

Proxy (see page 3533)

15.1.1.2.8 IStreamLoader.SetTimeout Method

Pascal

```
procedure SetTimeout(ATimeOut: Integer);
```

C++

```
__fastcall SetTimeout(int ATimeOut);
```

Description

This is SetTimeout, a member of class IStreamLoader.

15.1.1.2.9 IStreamLoader.SetUserName Method

Sets the value of the UserName (see page 3533) property.

Pascal

```
procedure SetUserName(const UserName: string);
```

C++

```
__fastcall SetUserName(const UnicodeString UserName);
```

Description

SetUserName is the write implementation for the UserName (see page 3533) property.

UserName (see page 3533) is the user name to use when connecting to the server that hosts the WSDL document.

See Also

UserName (see page 3533)

15.1.1.3 IStreamLoader Properties

The properties of the IStreamLoader class are listed here.

Public Properties

	Name	Description
	Password (see page 3532)	Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.
	Proxy (see page 3533)	Lists any proxy servers that are required to connect to a server that hosts the WSDL document.
	Timeout (see page 3533)	This is Timeout, a member of class IStreamLoader.
	UserName (see page 3533)	Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

15.1.1.3.1 IStreamLoader.Password Property

Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

Use Password to provide a password that the stream loader can use if the WSDL document is on a server that requires authentication or if it needs to use a proxy server that requires authentication. Password is the password for the user specified by the UserName (see page 3533) property.

See Also

UserName (see page 3533)

Proxy (see page 3533)

15.1.1.3.2 IStreamLoader.Proxy Property

Lists any proxy servers that are required to connect to a server that hosts the WSDL document.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the stream loader can't resolve the host portion of an URL for the WSDL document locally. If the server that hosts the WSDL document is registered locally or if the WSDL document is a file on the local machine, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

Password (see page 3532)

UserName (see page 3533)

Load (see page 3530)

15.1.1.3.3 IStreamLoader.Timeout Property

Pascal

```
property Timeout: integer;
```

C++

```
__property int Timeout;
```

Description

This is Timeout, a member of class IStreamLoader.

15.1.1.3.4 IStreamLoader.UserName Property

Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

Use UserName to specify the user name that the stream loader can use if the WSDL document is on a server that requires authentication or if it needs to use a proxy server that requires authentication. The password for this user name is specified by the Password (see page 3532) property.

See Also

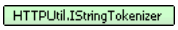
Password (see page 3532)

Proxy (see page 3533)

15.1.2 IStringTokenizer Interface

A string tokenizer breaks a string into tokens.

Class Hierarchy



Pascal

```
IStringTokenizer = interface;
```

C++

```
__interface IStringTokenizer;
```

File

HTTPUtil

Description

IStringTokenizer is implemented by string tokenizers, objects which iterate through the tokens in a string. To obtain a simple string tokenizer, call StringTokenizer (see page 3539).

A tokenizer is created from two strings: a source string and a delimiter string. In the source string, any characters in the delimiter string separate tokens, while any other characters are part of the tokens themselves. Thus in the following calls:

StringTokenizer (see page 3539)('This is a token/So+is+this', '/'); // Delphi example

StringTokenizer (see page 3539)("This is a token/So+is+this', '/'); // C++ example

StringTokenizer (see page 3539) creates a tokenizer that produces two tokens, consisting of the strings "This is a token" and "So+is+this".

Use the nextToken (see page 3535) method to iterate through the tokens.

See Also

StringTokenizer (see page 3539)

15.1.2.1 IStringTokenizer Members

The following tables list the members exposed by IStringTokenizer.

Public Methods

	Name	Description
	getTokenCounts (see page 3535)	Implements countTokens (see page 3536) property.

	hasMoreTokens (see page 3535)	Returns true if more tokens available.
	nextToken (see page 3535)	Returns next available token.

Public Properties

	Name	Description
R	countTokens (see page 3536)	Contains number of tokens in string.

15.1.2.2 IStringTokenizer Methods

The methods of the IStringTokenizer class are listed here.

Public Methods

	Name	Description
	getTokenCounts (see page 3535)	Implements countTokens (see page 3536) property.
	hasMoreTokens (see page 3535)	Returns true if more tokens available.
	nextToken (see page 3535)	Returns next available token.

15.1.2.2.1 IStringTokenizer.getTokenCounts Method

Implements countTokens (see page 3536) property.

Pascal

```
function getTokenCounts: integer;
```

C++

```
__fastcall int getTokenCounts();
```

Description

The getTokenCounts method returns the value of the countTokens (see page 3536) property.

See Also

counttokens (see page 3536)

15.1.2.2.2 IStringTokenizer.hasMoreTokens Method

Returns true if more tokens available.

Pascal

```
function hasMoreTokens: boolean;
```

C++

```
__fastcall boolean hasMoreTokens();
```

Description

The hasMoreTokens method returns true if unretrieved tokens remain, false otherwise.

See Also

nextToken

15.1.2.2.3 IStringTokenizer.nextToken Method

Returns next available token.

Pascal

```
function nextToken: WideString;
```

C++

```
__fastcall BSTR nextToken();
```

Description

The nextToken method returns the next available token. If no more tokens are available (hasMoreTokens (see page 3535) returns false), nextToken returns an empty string.

See Also

nextToken

15.1.2.3 IStringTokenizer Properties

The properties of the IStringTokenizer class are listed here.

Public Properties

	Name	Description
R	countTokens (see page 3536)	Contains number of tokens in string.

15.1.2.3.1 IStringTokenizer.countTokens Property

Contains number of tokens in string.

Pascal

```
property countTokens: integer;
```

C++

```
__property int countTokens;
```

Description

The countTokens property contains the number of tokens in the string.

See Also

AbsoluteIndex

operator_sb

15.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FirstDelimiter (see page 3537)	This is function HTTPUtil.FirstDelimiter.
	FirstDelimiter (see page 3537)	This is function HTTPUtil.FirstDelimiter.
	GetDefaultStreamLoader (see page 3538)	This is function HTTPUtil.GetDefaultStreamLoader.
	HTMLEscape (see page 3538)	This is function HTTPUtil.HTMLEscape.
	StartsWith (see page 3538)	This is function HTTPUtil.StartsWith.
	StringTokenizer (see page 3539)	This is function HTTPUtil.StringTokenizer.
	StringToStringArray (see page 3539)	This is function HTTPUtil.StringToStringArray.

15.2.1 HTTPUtil.FirstDelimiter Function

Pascal

```
function FirstDelimiter(const delimiters: WideString; const Str: WideString): integer;
overload;
```

C++

```
int FirstDelimiter(const BSTR delimiters, const BSTR Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.FirstDelimiter.

15.2.2 HTTPUtil.FirstDelimiter Function

Pascal

```
function FirstDelimiter(const delimiters: string; const Str: String): integer; overload;
```

C++

```
int FirstDelimiter(const UnicodeString delimiters, const UnicodeString Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.FirstDelimiter.

15.2.3 HTTPUtil.GetDefaultStreamLoader Function

Pascal

```
function GetDefaultStreamLoader: IStreamLoader;
```

C++

```
IStreamLoader GetDefaultStreamLoader();
```

File

HTTPUtil

Description

This is function HTTPUtil.GetDefaultStreamLoader.

15.2.4 HTTPUtil.HTMLEscape Function

Pascal

```
function HTMLEscape(const Str: string): string;
```

C++

```
UnicodeString HTMLEscape(const UnicodeString Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.HTMLEscape.

15.2.5 HTTPUtil.StartsWith Function

Pascal

```
function StartsWith(const str: string; const sub: string): Boolean;
```

C++

```
Boolean StartsWith(const UnicodeString str, const UnicodeString sub);
```

File

HTTPUtil

Description

This is function HTTPUtil.StartsWith.

15.2.6 HTTPUtil.StringTokenizer Function

Pascal

```
function StringTokenizer(const str: string; const delim: string): IStringTokenizer;
```

C++

```
IStringTokenizer StringTokenizer(const UnicodeString str, const UnicodeString delim);
```

File

HTTPUtil

Description

This is function HTTPUtil.StringTokenizer.

15.2.7 HTTPUtil.StringToStringArray Function

Pascal

```
function StringToStringArray(const str: string; const delim: string): TStringDynArray;
```

C++

```
TStringDynArray StringToStringArray(const UnicodeString str, const UnicodeString delim);
```

File

HTTPUtil

Description

This is function HTTPUtil.StringToStringArray.

16 IB Namespace

This is namespace IB.

Topics

Name	Description
Classes (see page 3543)	The following table lists classes in this documentation.
Functions (see page 3547)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3551)	The following table lists structs, records, enums in this documentation.
Types (see page 3557)	The following table lists types in this documentation.
Variables (see page 3560)	The following table lists variables in this documentation.
Constants (see page 3561)	The following table lists constants in this documentation.

Classes

	Name	Description
	EIBClientError (see page 3543)	EIBClientError represents a client-specific error in an InterBase Express application.
	EIBError (see page 3543)	EIBError is the exception class for InterBase errors.
	EIBInterBaseError (see page 3545)	EIBInterBaseError is used to raise server-specific errors.
	EIBInterBaseRoleError (see page 3546)	This is class IB.EIBInterBaseRoleError.

Constants

Name	Description
IBBigLocalBufferLength (see page 3561)	This is constant IB.IBBigLocalBufferLength.
IBErrorMessages (see page 3561)	This is constant IB.IBErrorMessages.
IBHugeLocalBufferLength (see page 3562)	This is constant IB.IBHugeLocalBufferLength.
IBLocalBufferLength (see page 3563)	This is constant IB.IBLocalBufferLength.
IBPalette1 (see page 3563)	This is constant IB.IBPalette1.
IBPalette2 (see page 3563)	This is constant IB.IBPalette2.

Functions

	Name	Description
	CheckStatusVector (see page 3547)	Sets or unsets status vector checking.
	GetIBDataBaseErrorMessages (see page 3547)	Returns the current error message setting.
	IBAlloc (see page 3548)	Allocates or de-allocates memory for a given size, and initializes the new memory to zero.
	IBDataBaseError (see page 3548)	Interprets the SQLCode and IBErrorCode status and passes it to the EIBInterBaseError (see page 3545) exception.

	IBError (see page 3548)	Returns the error message for the specified error code.
	SetIBDataBaseErrorMessages (see page 3549)	Enables you to choose the error messages you want to see.
	StatusVector (see page 3549)	Retrieves the thread-specific status vector from thread-local storage.
	StatusVectorArray (see page 3549)	Returns a pointer to the status vector array.
	StatusVectorAsText (see page 3550)	Gets the status vector and returns it as a formatted string.

Structs, Records, Enums

	Name	Description
	TIBClientError (see page 3551)	This is record IB.TIBClientError.
	TIBDataBaseErrorMessage (see page 3555)	This is record IB.TIBDataBaseErrorMessage.
	TTraceFlag (see page 3555)	This is record IB.TTraceFlag.

Types

Name	Description
PStatusVector (see page 3557)	This is type IB.PStatusVector.
TIBDataBaseErrorMessages (see page 3557)	TIBDataBaseErrorMessage (see page 3555) and TIBDataBaseErrorMessages indicate the various strings available to describe an error that an IBX component encountered.
TStatusVector (see page 3558)	This is type IB.TStatusVector.
TTraceFlags (see page 3558)	TTraceFlag (see page 3555) and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.

Variables

Name	Description
IBCS (see page 3560)	This is variable IB.IBCS.

16.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EIBClientError (see page 3543)	EIBClientError represents a client-specific error in an InterBase Express application.
	EIBError (see page 3543)	EIBError is the exception class for InterBase errors.
	EIBInterBaseError (see page 3545)	EIBInterBaseError is used to raise server-specific errors.
	EIBInterBaseRoleError (see page 3546)	This is class IB.EIBInterBaseRoleError.

16.1.1 EIBClientError Class

EIBClientError represents a client-specific error in an InterBase Express application.

Class Hierarchy

Pascal

```
EIBClientError = class(EIBError);
```

C++

```
class EIBClientError : public EIBError;
```

File

IB

Description

Use EIBClientError to raise an exception for client-specific errors. It is raised when an application tries to use a component improperly. For example, an EIBClientError is raised when a component can't access the database because it does not have an IBDatabase (see page 3703) component specified, attempts to edit a read-only dataset, tries to execute an empty query, and so on.

16.1.2 EIBError Class

EIBError is the exception class for InterBase errors.

Class Hierarchy

Pascal

```
EIBError = class(EDatabaseError);
```

C++

```
class EIBError : public EDatabaseError;
```

File

IB

Description

Use EIBError to raise an exception when a component detects an error in the database or in the component implementation. EIBError is the base class for all errors that arise in the use of InterBase Express. Its descendants represent specific types of errors, and include EIBClientError (see page 3543), EIBInterBaseError (see page 3545), and EIBInterbaseRoleError.

16.1.2.1 EIBError Members

The following tables list the members exposed by EIBError.

Public Methods

	Name	Description
	Create (see page 3544)	This is the overview for the Create constructor overload.

Public Properties

	Name	Description
R	IBErrorCode (see page 3545)	Returns the InterBase error code.
R	SQLCode (see page 3545)	Translates an InterBase error code in the error status vector to an SQL error number code.

16.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	EIBError.Create (Long, Long, string) (see page 3544)	Calls the inherited Create and sets the SQLCode (see page 3545) and IBErrorCode (see page 3545)
	EIBError.Create (Long, string) (see page 3544)	Calls the inherited Create and sets the SQLCode (see page 3545) and IBErrorCode (see page 3545)

16.1.2.2.1 EIBError.Create Constructor (Long, Long, string)

Calls the inherited Create and sets the SQLCode (see page 3545) and IBErrorCode (see page 3545)

Pascal

```
constructor Create(ASQLCode: Long; Msg: string); overload;
constructor Create(ASQLCode: Long; AIBErrorCode: Long; Msg: string); overload;
```

C++

```
__fastcall EIBError(Long ASQLCode, UnicodeString Msg);
__fastcall EIBError(Long ASQLCode, Long AIBErrorCode, UnicodeString Msg);
```

Description

Note: Use Create to call to the inherited Create and sets the SQL code and IBErrorCode (see page 3545).

16.1.2.3 EIBError Properties

The properties of the EIBError class are listed here.

Public Properties

	Name	Description
	IBErrorCode (see page 3545)	Returns the InterBase error code.
	SQLCode (see page 3545)	Translates an InterBase error code in the error status vector to an SQL error number code.

16.1.2.3.1 EIBError.IBErrorCode Property

Returns the InterBase error code.

Pascal

```
property IBErrorCode: Long;
```

C++

```
__property Long IBErrorCode;
```

Description

Use IBErrorCode to get the InterBase error code.

16.1.2.3.2 EIBError.SQLCode Property

Translates an InterBase error code in the error status vector to an SQL error number code.

Pascal

```
property SQLCode: Long;
```

C++

```
__property Long SQLCode;
```

Description

Translates an InterBase error code in the error status vector to an SQL error number code.

Delphi syntax:

```
property SQLCode: Long;
```

C++ syntax:

```
__property
        int SQLCode = {read=FSQLCode, nodefault};
```

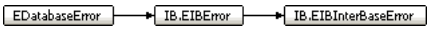
Description

Use SQLCode to translate an InterBase error code in the error status vector to an SQL error number code. Typically, this call is used to populate a program variable with an SQL error number for use in an SQL error-handling routine.

16.1.3 EIBInterBaseError Class

EIBInterBaseError is used to raise server-specific errors.

Class Hierarchy



Pascal

```
EIBInterBaseError = class(EIBError);
```

C++

```
class EIBInterBaseError : public EIBError;
```

File

IB

Description

Use EIBInterBaseError to raise an exception that represents an error code that the InterBase server returns.

16.1.4 EIBInterBaseRoleError Class

Class Hierarchy

Pascal

```
EIBInterBaseRoleError = class(EIBError);
```

C++

```
class EIBInterBaseRoleError : public EIBError;
```

File

IB

Description

This is class IB.EIBInterBaseRoleError.

16.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckStatusVector (see page 3547)	Sets or unsets status vector checking.
	GetIBDataBaseErrorMessages (see page 3547)	Returns the current error message setting.
	IBAlloc (see page 3548)	Allocates or de-allocates memory for a given size, and initializes the new memory to zero.
	IBDataBaseError (see page 3548)	Interprets the SQLCode and IBErrorCode status and passes it to the EIBInterBaseError (see page 3545) exception.
	IBError (see page 3548)	Returns the error message for the specified error code.
	SetIBDataBaseErrorMessages (see page 3549)	Enables you to choose the error messages you want to see.
	StatusVector (see page 3549)	Retrieves the thread-specific status vector from thread-local storage.
	StatusVectorArray (see page 3549)	Returns a pointer to the status vector array.
	StatusVectorAsText (see page 3550)	Gets the status vector and returns it as a formatted string.

16.2.1 IB.CheckStatusVector Function

Sets or unsets status vector checking.

Pascal

```
function CheckStatusVector(ErrorCodes: array of ISC_STATUS): Boolean;
```

C++

```
Boolean CheckStatusVector(array of ISC_STATUS ErrorCodes);
```

File

IB

Description

Set CheckStatusVector to true to enable status vector checking.

16.2.2 IB.GetIBDataBaseErrorMessages Function

Returns the current error message setting.

Pascal

```
function GetIBDataBaseErrorMessages: TIBDataBaseErrorMessages;
```

C++

```
TIBDataBaseErrorMessages GetIBDataBaseErrorMessages();
```

File

IB

Description

Use the GetIBDataBaseErrorMessages function to return the current error message setting.

16.2.3 IB.IBAlloc Function

Allocates or de-allocates memory for a given size, and initializes the new memory to zero.

Pascal

```
procedure IBAlloc(var P; OldSize: Integer; NewSize: Integer);
```

C++

```
IBAlloc( P, int OldSize, int NewSize);
```

File

IB

Description

Use IBAlloc to allocate or de-allocate memory for a given size, and initialize the new memory to zero.

16.2.4 IB.IBDataBaseError Function

Interprets the SQLCode and IBErrorCode status and passes it to the EIBInterBaseError (see page 3545) exception.

Pascal

```
procedure IBDataBaseError;
```

C++

```
IBDataBaseError();
```

File

IB

Description

Use the IBDataBaseError procedure to look at the status vector, interpret the SQLCode and IBErrorCode status, construct the status string, and pass it to the EIBInterBaseError (see page 3545) exception.

16.2.5 IB.IBError Function

Returns the error message for the specified error code.

Pascal

```
procedure IBError(ErrMess: TIBClientError; const Args: array of const);
```

C++

```
IBError(TIBClientError ErrMess, const array of const Args);
```

File

IB

Description

Use the IBError procedure to look up the appropriate error message for a specified error code and raise an EIBClientError (see page 3543) exception.

16.2.6 IB.SetIBDataBaseErrorMessages Function

Enables you to choose the error messages you want to see.

Pascal

```
procedure SetIBDataBaseErrorMessages(Value: TIBDataBaseErrorMessages);
```

C++

```
SetIBDataBaseErrorMessages(TIBDataBaseErrorMessages Value);
```

File

IB

Description

Use the SetIBDataBaseErrorMessages procedure to choose the error messages that you want to use. The available options are defined in TIBDataBaseErrorMessages (see page 3557).

16.2.7 IB.StatusVector Function

Retrieves the thread-specific status vector from thread-local storage.

Pascal

```
function StatusVector: PISC_STATUS;
```

C++

```
PISC_STATUS StatusVector();
```

File

IB

Description

Use the StatusVector function to retrieve the thread-specific status from the thread-local storage.

16.2.8 IB.StatusVectorArray Function

Returns a pointer to the status vector array.

Pascal

```
function StatusVectorArray: PStatusVector;
```

C++

```
PStatusVector StatusVectorArray();
```

File

IB

Description

Use the StatusVectorArray function to return a pointer to the status vector array from the thread-local storage.

16.2.9 IB.StatusVectorAsText Function

Gets the status vector and returns it as a formatted string.

Pascal

```
function StatusVectorAsText: string;
```

C++

```
UnicodeString StatusVectorAsText();
```

File

IB

Description

Use the StatusVector (see page 3549) function to retrieve the status vector and return it as a formatted string.

16.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TIBClientError (see page 3551)	This is record IB.TIBClientError.
	TIBDataBaseErrorMessage (see page 3555)	This is record IB.TIBDataBaseErrorMessage.
	TTraceFlag (see page 3555)	This is record IB.TTraceFlag.

16.3.1 IB.TIBClientError Enumeration

Pascal

```
TIBClientError = (
  ibxeUnknownError,
  ibxeInterBaseMissing,
  ibxeInterBaseInstallMissing,
  ibxeIB60feature,
  ibxeNotSupported,
  ibxeNotPermitted,
  ibxeFileAccessError,
  ibxeConnectionTimeout,
  ibxeCannotSetDatabase,
  ibxeCannotSetTransaction,
  ibxeOperationCancelled,
  ibxeDPBConstantNotSupported,
  ibxeDPBConstantUnknown,
  ibxeTPBConstantNotSupported,
  ibxeTPBConstantUnknown,
  ibxeDatabaseClosed,
  ibxeDatabaseOpen,
  ibxeDatabaseNameMissing,
  ibxeNotInTransaction,
  ibxeInTransaction,
  ibxeTimeoutNegative,
  ibxeNoDatabasesInTransaction,
  ibxeUpdateWrongDB,
  ibxeUpdateWrongTR,
  ibxeDatabaseNotAssigned,
  ibxeTransactionNotAssigned,
  ibxeXSQLDAIndexOutOfRange,
  ibxeXSQLDANameDoesNotExist,
  ibxeEOF,
  ibxeBOF,
  ibxeInvalidStatementHandle,
  ibxeSQLOpen,
  ibxeSQLClosed,
  ibxeDatasetOpen,
  ibxeDatasetClosed,
  ibxeUnknownSQLDataType,
  ibxeInvalidColumnIndex,
  ibxeInvalidParamColumnIndex,
  ibxeInvalidDataConversion,
  ibxeColumnIsNotNullable,
  ibxeBlobCannotBeRead,
```

```
    ibxeBlobCannotBeWritten,
    ibxeEmptyQuery,
    ibxeCannotOpenNonSQLSelect,
    ibxeNoFieldAccess,
    ibxeFieldReadOnly,
    ibxeFieldNotFound,
    ibxeNotEditing,
    ibxeCannotInsert,
    ibxeCannotPost,
    ibxeCannotUpdate,
    ibxeCannotDelete,
    ibxeCannotRefresh,
    ibxeBufferNotSet,
    ibxeCircularReference,
    ibxeSQLParseError,
    ibxeUserAbort,
    ibxeDataSetUniDirectional,
    ibxeCannotCreateSharedResource,
    ibxeWindowsAPIError,
    ibxeColumnListsDontMatch,
    ibxeColumnTypesDontMatch,
    ibxeCantEndSharedTransaction,
    ibxeFieldUnsupportedType,
    ibxeCircularDataLink,
    ibxeEmptySQLStatement,
    ibxeIsASelectStatement,
    ibxeRequiredParamNotSet,
    ibxeNoStoredProcName,
    ibxeIsAExecuteProcedure,
    ibxeUpdateFailed,
    ibxeNotCachedUpdates,
    ibxeNotLiveRequest,
    ibxeNoProvider,
    ibxeNoRecordsAffected,
    ibxeNoTableName,
    ibxeCannotCreatePrimaryIndex,
    ibxeCannotDropSystemIndex,
    ibxeTableNameMismatch,
    ibxeIndexFieldMissing,
    ibxeInvalidCancellation,
    ibxeInvalidEvent,
    ibxeMaximumEvents,
    ibxeNoEventsRegistered,
    ibxeInvalidQueueing,
    ibxeInvalidRegistration,
    ibxeInvalidBatchMove,
    ibxeSQLDialectInvalid,
    ibxeSPBConstantNotSupported,
    ibxeSPBConstantUnknown,
    ibxeServiceActive,
    ibxeServiceInActive,
    ibxeServerNameMissing,
    ibxeQueryParamsError,
    ibxeStartParamsError,
    ibxeOutputParsingError,
    ibxeUseSpecificProcedures,
    ibxeSQLMonitorAlreadyPresent,
    ibxeCantPrintValue,
    ibxeEOFReached,
    ibxeEOFInComment,
    ibxeEOFInString,
    ibxeParamNameExpected,
    ibxeSuccess,
    ibxeDelphiException,
    ibxeNoOptionsSet,
    ibxeNoDestinationDirectory,
```

```
    ibxeNosourceDirectory,
    ibxeNoUninstallFile,
    ibxeOptionNeedsClient,
    ibxeOptionNeedsServer,
    ibxeInvalidOption,
    ibxeInvalidOnErrorResult,
    ibxeInvalidOnStatusResult,
    ibxeDPBConstantUnknownEx,
    ibxeTPBConstantUnknownEx,
    ibxeUnknownPlan,
    ibxeFieldSizeMismatch,
    ibxeEventAlreadyRegistered,
    ibxeStringTooLarge
  );
```

C++

```
  enum TIBClientError {
    ibxeUnknownError,
    ibxeInterBaseMissing,
    ibxeInterBaseInstallMissing,
    ibxeIB60feature,
    ibxeNotSupported,
    ibxeNotPermitted,
    ibxeFileAccessError,
    ibxeConnectionTimeout,
    ibxeCannotSetDatabase,
    ibxeCannotSetTransaction,
    ibxeOperationCancelled,
    ibxeDPBConstantNotSupported,
    ibxeDPBConstantUnknown,
    ibxeTPBConstantNotSupported,
    ibxeTPBConstantUnknown,
    ibxeDatabaseClosed,
    ibxeDatabaseOpen,
    ibxeDatabaseNameMissing,
    ibxeNotInTransaction,
    ibxeInTransaction,
    ibxeTimeoutNegative,
    ibxeNoDatabasesInTransaction,
    ibxeUpdateWrongDB,
    ibxeUpdateWrongTR,
    ibxeDatabaseNotAssigned,
    ibxeTransactionNotAssigned,
    ibxeXSQLDAIndexOutOfRange,
    ibxeXSQLDANameDoesNotExist,
    ibxeEOF,
    ibxeBOF,
    ibxeInvalidStatementHandle,
    ibxeSQLOpen,
    ibxeSQLClosed,
    ibxeDatasetOpen,
    ibxeDatasetClosed,
    ibxeUnknownSQLDataType,
    ibxeInvalidColumnIndex,
    ibxeInvalidParamColumnIndex,
    ibxeInvalidDataConversion,
    ibxeColumnIsNotNullable,
    ibxeBlobCannotBeRead,
    ibxeBlobCannotBeWritten,
    ibxeEmptyQuery,
    ibxeCannotOpenNonSQLSelect,
    ibxeNoFieldAccess,
    ibxeFieldReadOnly,
    ibxeFieldNotFound,
    ibxeNotEditing,
    ibxeCannotInsert,
```

```
    ibxeCannotPost,
    ibxeCannotUpdate,
    ibxeCannotDelete,
    ibxeCannotRefresh,
    ibxeBufferNotSet,
    ibxeCircularReference,
    ibxeSQLParseError,
    ibxeUserAbort,
    ibxeDataSetUniDirectional,
    ibxeCannotCreateSharedResource,
    ibxeWindowsAPIError,
    ibxeColumnListsDontMatch,
    ibxeColumnTypesDontMatch,
    ibxeCantEndSharedTransaction,
    ibxeFieldUnsupportedType,
    ibxeCircularDataLink,
    ibxeEmptySQLStatement,
    ibxeIsASelectStatement,
    ibxeRequiredParamNotSet,
    ibxeNoStoredProcName,
    ibxeIsAExecuteProcedure,
    ibxeUpdateFailed,
    ibxeNotCachedUpdates,
    ibxeNotLiveRequest,
    ibxeNoProvider,
    ibxeNoRecordsAffected,
    ibxeNoTableName,
    ibxeCannotCreatePrimaryIndex,
    ibxeCannotDropSystemIndex,
    ibxeTableNameMismatch,
    ibxeIndexFieldMissing,
    ibxeInvalidCancellation,
    ibxeInvalidEvent,
    ibxeMaximumEvents,
    ibxeNoEventsRegistered,
    ibxeInvalidQueueing,
    ibxeInvalidRegistration,
    ibxeInvalidBatchMove,
    ibxeSQLDialectInvalid,
    ibxeSPBConstantNotSupported,
    ibxeSPBConstantUnknown,
    ibxeServiceActive,
    ibxeServiceInActive,
    ibxeServerNameMissing,
    ibxeQueryParamsError,
    ibxeStartParamsError,
    ibxeOutputParsingError,
    ibxeUseSpecificProcedures,
    ibxeSQLMonitorAlreadyPresent,
    ibxeCantPrintValue,
    ibxeEOFReached,
    ibxeEOFInComment,
    ibxeEOFInString,
    ibxeParamNameExpected,
    ibxeSuccess,
    ibxeDelphiException,
    ibxeNoOptionsSet,
    ibxeNoDestinationDirectory,
    ibxeNosourceDirectory,
    ibxeNoUninstallFile,
    ibxeOptionNeedsClient,
    ibxeOptionNeedsServer,
    ibxeInvalidOption,
    ibxeInvalidOnErrorResult,
    ibxeInvalidOnStatusResult,
    ibxeDPBConstantUnknownEx,
```

```
    ibxeTPBConstantUnknownEx,
    ibxeUnknownPlan,
    ibxeFieldSizeMismatch,
    ibxeEventAlreadyRegistered,
    ibxeStringTooLarge
  };
```

File

IB

Description

This is record IB.TIBClientError.

16.3.2 IB.TIBDataBaseErrorMessage Enumeration

Pascal

```
  TIBDataBaseErrorMessage = (
    ShowSQLCode,
    ShowIBMessage,
    ShowSQLMessage
  );
```

C++

```
  enum TIBDataBaseErrorMessage {
    ShowSQLCode,
    ShowIBMessage,
    ShowSQLMessage
  };
```

File

IB

Description

This is record IB.TIBDataBaseErrorMessage.

16.3.3 IB.TTraceFlag Enumeration

Pascal

```
  TTraceFlag = (
    tfQPrepare,
    tfQExecute,
    tfQFetch,
    tfError,
    tfStmt,
    tfConnect,
    tfTransact,
    tfBlob,
    tfService,
    tfMisc
  );
```

C++

```
  enum TTraceFlag {
    tfQPrepare,
```

```
    tfQExecute,
    tfQFetch,
    tfError,
    tfStmt,
    tfConnect,
    tfTransact,
    tfBlob,
    tfService,
    tfMisc
  };
```

File

IB

Description

This is record IB.TTraceFlag.

16.4 Types

The following table lists types in this documentation.

Types

Name	Description
PStatusVector (see page 3557)	This is type IB.PStatusVector.
TIBDataBaseErrorMessages (see page 3557)	TIBDataBaseErrorMessage (see page 3555) and TIBDataBaseErrorMessages indicate the various strings available to describe an error that an IBX component encountered.
TStatusVector (see page 3558)	This is type IB.TStatusVector.
TTraceFlags (see page 3558)	TTraceFlag (see page 3555) and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.

16.4.1 IB.PStatusVector Type

Pascal

```
PStatusVector = ^TStatusVector;
```

C++

```
^TStatusVector PStatusVector;
```

File

IB

Description

This is type IB.PStatusVector.

16.4.2 IB.TIBDataBaseErrorMessages Type

TIBDataBaseErrorMessage (see page 3555) and TIBDataBaseErrorMessages indicate the various strings available to describe an error that an IBX component encountered.

Pascal

```
TIBDataBaseErrorMessages = set of TIBDataBaseErrorMessage;
```

C++

```
TIBDataBaseErrorMessage TIBDataBaseErrorMessages;
```

File

IB

Description

TIBDataBaseErrorMessages is a set of TIBDataBaseErrorMessage (see page 3555) values. The following table lists the possible values:

Value Meaning

ShowSQLCode The SQL code that the server was trying to execute when the error occurred.

ShowIBMessage The error message InterBase provides for the error.

ShowSQLMessage The error message that describes what part of the SQL code caused the error.

16.4.3 IB.TStatusVector Type

Pascal

```
TStatusVector = array[0..19] of ISC_STATUS;
```

C++

```
array[0..19] of ISC_STATUS TStatusVector;
```

File

IB

Description

This is type IB.TStatusVector.

16.4.4 IB.TTraceFlags Type

TTraceFlag (see page 3555) and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.

Pascal

```
TTraceFlags = set of TTraceFlag;
```

C++

```
TTraceFlag TTraceFlags;
```

File

IB

Description

TTraceFlags is a set of TTraceFlag (see page 3555) values. The following table lists the possible values:

Values Meaning

tfQPrepare Monitor Prepare statements.

tfQExecute Monitor ExecSQL statements.

tfError Monitor server error messages. Such messages may include an error code.

tfStmt Monitor all SQL statements.

tfConnect Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.

tfTransact Monitor transaction statements, such as StartTransaction, Commit, and Rollback.

tfBlob Monitor operations on blob data types.

tfService Monitor messages to one of the IB (see page 3541) services.

tfMisc Monitor any statements not covered by other flag options.

16.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
IBCS (see page 3560)	This is variable IB.IBCS.

16.5.1 IB.IBCS Variable

Pascal

```
IBCS: TRTLCriticalSection;
```

C++

```
TRTLCriticalSection IBCS;
```

File

IB

Description

This is variable IB.IBCS.

16.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IBBigLocalBufferLength (see page 3561)	This is constant IB.IBBigLocalBufferLength.
IBErrorMessages (see page 3561)	This is constant IB.IBErrorMessages.
IBHugeLocalBufferLength (see page 3562)	This is constant IB.IBHugeLocalBufferLength.
IBLocalBufferLength (see page 3563)	This is constant IB.IBLocalBufferLength.
IBPalette1 (see page 3563)	This is constant IB.IBPalette1.
IBPalette2 (see page 3563)	This is constant IB.IBPalette2.

16.6.1 IB.IBBigLocalBufferLength Constant

Pascal

```
IBBigLocalBufferLength = IBLocalBufferLength * 2;
```

C++

```
IBBigLocalBufferLength = IBLocalBufferLength * 2;
```

File

IB

Description

This is constant IB.IBBigLocalBufferLength.

16.6.2 IB.IBErrorMessages Constant

Pascal

```
IBErrorMessages: array[TIBClientError] of string = ( SUnknownError, SInterBaseMissing,
SInterBaseInstallMissing, SIB60feature, SNotSupported, SNotPermitted, SFileAccessError,
SConnectionTimeout, SCannotSetDatabase, SCannotSetTransaction, SOperationCancelled,
SDPBConstantNotSupported, SDPBConstantUnknown, STPBConstantNotSupported, STPBConstantUnknown,
SDatabaseClosed, SDatabaseOpen, SDatabaseNameMissing, SNotInTransaction, SInTransaction,
STimeoutNegative, SNoDatabasesInTransaction, SUpdateWrongDB, SUpdateWrongTR,
SDatabaseNotAssigned, STransactionNotAssigned, SXSQLDAIndexOutOfRange,
SXSQLDANameDoesNotExist, SEOF, SBOF, SInvalidStatementHandle, SSQLOpen, SSQLClosed,
SDatasetOpen, SDatasetClosed, SUnknownSQLDataType, SInvalidColumnIndex,
SInvalidParamColumnIndex, SInvalidDataConversion, SColumnIsNotNullable, SBlobCannotBeRead,
SBlobCannotBeWritten, SEmptyQuery, SCannotOpenNonSQLSelect, SNoFieldAccess, SFieldReadOnly,
SFieldNotFound, SNotEditing, SCannotInsert, SCannotPost, SCannotUpdate, SCannotDelete,
SCannotRefresh, SBufferNotSet, SCircularReference, SSQLParseError, SUserAbort,
SDataSetUniDirectional, SCannotCreateSharedResource, SWindowsAPIError, SColumnListsDontMatch,
SColumnTypesDontMatch, SCantEndSharedTransaction, SFieldUnsupportedType, SCircularDataLink,
SEmptySQLStatement, SIsASelectStatement, SRequiredParamNotSet, SNoStoredProcName,
SIsAExecuteProcedure, SUpdateFailed, SNotCachedUpdates, SNotLiveRequest, SNoProvider,
SNoRecordsAffected, SNoTableName, SCannotCreatePrimaryIndex, SCannotDropSystemIndex,
```

```
STableNameMismatch, SIndexFieldMissing, SInvalidCancellation, SInvalidEvent, SMaximumEvents,
SNoEventsRegistered, SInvalidQueueing, SInvalidRegistration, SInvalidBatchMove,
SSQLDialectInvalid, SSPBConstantNotSupported, SSPBConstantUnknown, SServiceActive,
SServiceInActive, SServerNameMissing, SQueryParamsError, SStartParamsError,
SOutputParsingError, SUseSpecificProcedures, SSQLMonitorAlreadyPresent, SCantPrintValue,
SEOFReached, SEOFInComment, SEOFInString, SParamNameExpected, SSuccess, SDelphiException,
SNoOptionsSet, SNoDestinationDirectory, SNosourceDirectory, SNoUninstallFile,
SOptionNeedsClient, SOptionNeedsServer, SInvalidOption, SInvalidOnErrorResult,
SInvalidOnStatusResult, SDPBConstantUnknownEx, STPBConstantUnknownEx, SUnknownPlan,
SFieldSizeMismatch, SEventAlreadyRegistered, SStringTooLarge );
```

C++

```
array[TIBClientError] of string IBErrorMessages = ( SUnknownError, SInterBaseMissing,
SInterBaseInstallMissing, SIB60feature, SNotSupported, SNotPermitted, SFileAccessError,
SConnectionTimeout, SCannotSetDatabase, SCannotSetTransaction, SOperationCancelled,
SDPBConstantNotSupported, SDPBConstantUnknown, STPBConstantNotSupported, STPBConstantUnknown,
SDatabaseClosed, SDatabaseOpen, SDatabaseNameMissing, SNotInTransaction, SInTransaction,
STimeoutNegative, SNoDatabasesInTransaction, SUpdateWrongDB, SUpdateWrongTR,
SDatabaseNotAssigned, STransactionNotAssigned, SXSQLDAIndexOutOfRange,
SXSQLDANameDoesNotExist, SEOF, SBOF, SInvalidStatementHandle, SSQLOpen, SSQLClosed,
SDatasetOpen, SDatasetClosed, SUnknownSQLDataType, SInvalidColumnIndex,
SInvalidParamColumnIndex, SInvalidDataConversion, SColumnIsNotNullable, SBlobCannotBeRead,
SBlobCannotBeWritten, SEmptyQuery, SCannotOpenNonSQLSelect, SNoFieldAccess, SFieldReadOnly,
SFieldNotFound, SNotEditing, SCannotInsert, SCannotPost, SCannotUpdate, SCannotDelete,
SCannotRefresh, SBufferNotSet, SCircularReference, SSQLParseError, SUserAbort,
SDataSetUniDirectional, SCannotCreateSharedResource, SWindowsAPIError, SColumnListsDontMatch,
SColumnTypesDontMatch, SCantEndSharedTransaction, SFieldUnsupportedType, SCircularDataLink,
SEmptySQLStatement, SIsASelectStatement, SRequiredParamNotSet, SNoStoredProcName,
SIsAExecuteProcedure, SUpdateFailed, SNotCachedUpdates, SNotLiveRequest, SNoProvider,
SNoRecordsAffected, SNoTableName, SCannotCreatePrimaryIndex, SCannotDropSystemIndex,
STableNameMismatch, SIndexFieldMissing, SInvalidCancellation, SInvalidEvent, SMaximumEvents,
SNoEventsRegistered, SInvalidQueueing, SInvalidRegistration, SInvalidBatchMove,
SSQLDialectInvalid, SSPBConstantNotSupported, SSPBConstantUnknown, SServiceActive,
SServiceInActive, SServerNameMissing, SQueryParamsError, SStartParamsError,
SOutputParsingError, SUseSpecificProcedures, SSQLMonitorAlreadyPresent, SCantPrintValue,
SEOFReached, SEOFInComment, SEOFInString, SParamNameExpected, SSuccess, SDelphiException,
SNoOptionsSet, SNoDestinationDirectory, SNosourceDirectory, SNoUninstallFile,
SOptionNeedsClient, SOptionNeedsServer, SInvalidOption, SInvalidOnErrorResult,
SInvalidOnStatusResult, SDPBConstantUnknownEx, STPBConstantUnknownEx, SUnknownPlan,
SFieldSizeMismatch, SEventAlreadyRegistered, SStringTooLarge );
```

File

IB

Description

This is constant IB.IBErrorMessages.

16.6.3 IB.IBHugeLocalBufferLength Constant

Pascal

```
IBHugeLocalBufferLength = IBBigLocalBufferLength * 20;
```

C++

```
IBHugeLocalBufferLength = IBBigLocalBufferLength * 20;
```

File

IB

Description

This is constant IB.IBHugeLocalBufferLength.

16.6.4 IB.IBLocalBufferLength Constant

Pascal

```
IBLocalBufferLength = 512;
```

C++

```
IBLocalBufferLength = 512;
```

File

IB

Description

This is constant IB.IBLocalBufferLength.

16.6.5 IB.IBPalette1 Constant

Pascal

```
IBPalette1 = 'InterBase';
```

C++

```
IBPalette1 = 'InterBase';
```

File

IB

Description

This is constant IB.IBPalette1.

16.6.6 IB.IBPalette2 Constant

Pascal

```
IBPalette2 = 'InterBase Admin';
```

C++

```
IBPalette2 = 'InterBase Admin';
```

File

IB

Description

This is constant IB.IBPalette2.

17 IBBatchMove Namespace

This is namespace IBBatchMove.

Topics

Name	Description
Classes (see page 3566)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3578)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TIBBatchMove (see page 3566)	TIBBatchMove enables applications to perform database operations on groups of records or entire tables.

Structs, Records, Enums

	Name	Description
	TBatchMode (see page 3578)	This is record IBBatchMove.TBatchMode.

17.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBBatchMove (see page 3566)	TIBBatchMove enables applications to perform database operations on groups of records or entire tables.

17.1.1 TIBBatchMove Class

TIBBatchMove enables applications to perform database operations on groups of records or entire tables.

Class Hierarchy

Pascal

```
TIBBatchMove = class(TComponent);
```

C++

```
class TIBBatchMove : public TComponent;
```

File

IBBatchMove

Description

Set the properties of the TIBBatchMove object to

- Add the records in a dataset to a database table.
- Delete the records in a dataset from a database table.
- Copy a dataset to create a new database table or overwrite an existing table.

Call the Execute (see page 3568) method to perform the actual operation. TIBTable (see page 4142) objects can also perform the same batch operations, but only with the options specified by the default TIBBatchMove property values.

17.1.1.1 TIBBatchMove Members

The following tables list the members exposed by TIBBatchMove.

Protected Methods

	Name	Description
	Notification (see page 3569)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Create (see page 3567)	Creates an instance of TIBBatchMove (see page 3566).
	Destroy (see page 3568)	Destroys an instance of TIBBatchMove (see page 3566).

	Execute (see page 3568)	Performs the batch operation specified by Mode (see page 3574).

Public Properties

	Name	Description
R	ChangedCount (see page 3571)	Indicates the number of records from the Destination (see page 3572) table that are altered as a result of the batch operation.
R	KeyViolCount (see page 3572)	Reports the number of records that could not be replaced, added, or deleted from Destination (see page 3572) because of integrity or key violations.
R	MovedCount (see page 3574)	Reports the number of records form the Source (see page 3576) which were applied to the destination.
R	ProblemCount (see page 3575)	Indicates the number of records which could not be added to Destination (see page 3572) without loss of data due to a field type mismatch.

Published Properties

	Name	Description
	AbortOnKeyViol (see page 3570)	Specifies whether the batch operation is terminated immediately when integrity or key violations occur.
	AbortOnProblem (see page 3570)	Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 3572).
	ChangedTableName (see page 3571)	Specifies the name of a local Interbase table that is created to hold copies of all records from the destination table that are changed by the batch operation.
	CommitCount (see page 3571)	Specifies how many records are batch moved before a commit occurs.
	Destination (see page 3572)	Specifies the TIBTable (see page 4142) object for the database table that is the destination of the batch operation.
	KeyViolTableName (see page 3573)	Specifies the name of an InterBase table that will be created to contain all records from the Source (see page 3576) that cannot participate in the batch operation because of integrity or key violations.
	Mappings (see page 3573)	Specifies the column mappings for a batch operation.
	Mode (see page 3574)	Specifies what the TIBBatchMove (see page 3566) object does when the Execute (see page 3568) method is called.
	ProblemTableName (see page 3575)	Specifies the name of a Interbase table that will be created to contain all records from the Source (see page 3576) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 3572) table.
	RecordCount (see page 3576)	Specifies the maximum number of records that are applied to the Destination (see page 3572) when Execute (see page 3568) is called.
	Source (see page 3576)	Specifies the dataset that is the source of the batch operation.
	Transliterate (see page 3577)	Specifies whether the data in the Source (see page 3576) records should be converted form the locale of the Source (see page 3576) to the locale of the Destination (see page 3572) when the Execute (see page 3568) method is called.

17.1.1.2 TIBBatchMove.Create Constructor

Creates an instance of TIBBatchMove (see page 3566).

Pascal

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

C++

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

Description

Call Create to create an instance of TIBBatchMove (see page 3566) for performing a batch operation. Create allocates the memory for the TIBBatchMove (see page 3566) object and

- Sets AbortOnKeyViol (see page 3570) to True.
- Sets AbortOnProblem (see page 3570) to True.
- Sets Transliterate (see page 3577) to True.

- Creates a TStringList object to hold the Mappings (see page 3573).

The property settings made by Create are the values used when a TIBTable (see page 4142) object performs batch operations using its BatchMove method.

17.1.1.3 TIBBatchMove.Destroy Destructor

Destroys an instance of TIBBatchMove (see page 3566).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBBatchMove();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBBatchMove (see page 3566) object is not already freed and only then calls Destroy.

Before calling the inherited destructor, Destroy frees the TStringList object that implements the Mappings (see page 3573) property.

17.1.1.4 TIBBatchMove Methods

The methods of the TIBBatchMove class are listed here.

Protected Methods

	Name	Description
	Notification (see page 3569)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Execute (see page 3568)	Performs the batch operation specified by Mode (see page 3574).

17.1.1.4.1 TIBBatchMove.Execute Method

Performs the batch operation specified by Mode (see page 3574).

Pascal

```
procedure Execute;
```

C++

```
__fastcall Execute();
```

Description

After setting properties to indicate what batch operation should be performed and how, call Execute to perform the operation. As a minimum, the Mode (see page 3574), Source (see page 3576), and Destination (see page 3572) properties must be set.

After calling Execute, the values of the ChangedCount (see page 3571), KeyViolCount (see page 3572), MovedCount (see page 3574), ProblemCount (see page 3575), and RecordCount (see page 3576) properties will become available. These properties will give an indication of what happened as a result of the call to Execute. Additional information about the batch operation will be available through the creation of a changed table, a keyviol table, and/or a problems table if it was requested.

17.1.1.4.2 TIBBatchMove.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

17.1.1.5 TIBBatchMove Properties

The properties of the TIBBatchMove class are listed here.

Public Properties

	Name	Description
R	ChangedCount (see page 3571)	Indicates the number of records from the Destination (see page 3572) table that are altered as a result of the batch operation.
R	KeyViolCount (see page 3572)	Reports the number of records that could not be replaced, added, or deleted from Destination (see page 3572) because of integrity or key violations.
R	MovedCount (see page 3574)	Reports the number of records form the Source (see page 3576) which were applied to the destination.
R	ProblemCount (see page 3575)	Indicates the number of records which could not be added to Destination (see page 3572) without loss of data due to a field type mismatch.

Published Properties

	Name	Description
	AbortOnKeyViol (see page 3570)	Specifies whether the batch operation is terminated immediately when integrity or key violations occur.
	AbortOnProblem (see page 3570)	Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 3572).
	ChangedTableName (see page 3571)	Specifies the name of a local Interbase table that is created to hold copies of all records from the destination table that are changed by the batch operation.
	CommitCount (see page 3571)	Specifies how many records are batch moved before a commit occurs.
	Destination (see page 3572)	Specifies the TIBTable (see page 4142) object for the database table that is the destination of the batch operation.

	KeyViolTableName (see page 3573)	Specifies the name of an InterBase table that will be created to contain all records from the Source (see page 3576) that cannot participate in the batch operation because of integrity or key violations.
	Mappings (see page 3573)	Specifies the column mappings for a batch operation.
	Mode (see page 3574)	Specifies what the TIBBatchMove (see page 3566) object does when the Execute (see page 3568) method is called.
	ProblemTableName (see page 3575)	Specifies the name of a Interbase table that will be created to contain all records from the Source (see page 3576) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 3572) table.
	RecordCount (see page 3576)	Specifies the maximum number of records that are applied to the Destination (see page 3572) when Execute (see page 3568) is called.
	Source (see page 3576)	Specifies the dataset that is the source of the batch operation.
	Transliterate (see page 3577)	Specifies whether the data in the Source (see page 3576) records should be converted form the locale of the Source (see page 3576) to the locale of the Destination (see page 3572) when the Execute (see page 3568) method is called.

17.1.1.5.1 TIBBatchMove.AbortOnKeyViol Property

Specifies whether the batch operation is terminated immediately when integrity or key violations occur.

Pascal

```
property AbortOnKeyViol: Boolean;
```

C++

```
__property Boolean AbortOnKeyViol;
```

Description

Set AbortOnKeyViol to true (the default) to cancel the batch operation when it causes integrity or key violations in the destination table. Set AbortOnKeyViol to false to cause records that would cause integrity or key violations to be posted to a key violations table instead. Provide a KeyViolTableName (see page 3573) when setting AbortOnKeyViol to false. If there is no KeyViolTableName (see page 3573), AbortOnKeyViol may still be set to false, in which case problem records are counted but not saved in a table.

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

17.1.1.5.2 TIBBatchMove.AbortOnProblem Property

Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 3572).

Pascal

```
property AbortOnProblem: Boolean;
```

C++

```
__property Boolean AbortOnProblem;
```

Description

Set AbortOnProblem to true to cancel the batch operation when types of the fields in the Source (see page 3576) table records do not match the types of the fields in the Destination (see page 3572) table records. Set AbortOnProblem to false to cause

the mismatched fields to be 'trimmed'. When AbortOnProblem is false, set the ProblemTableName (see page 3575) to have all the untrimmed versions of records that were trimmed saved to a separate table.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TIBBatchMove (see page 3566) to raise an exception.

Trimming does not occur when the Mode (see page 3574) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table that do not exactly match the values from the source table may be deleted.

17.1.1.5.3 TIBBatchMove.ChangedCount Property

Indicates the number of records from the Destination (see page 3572) table that are altered as a result of the batch operation.

Pascal

```
property ChangedCount: Longint;
```

C++

```
__property Longint ChangedCount;
```

Description

Use ChangedCount to learn the number of records in the Destination (see page 3572) table that are updated when the Mode (see page 3574) is batUpdate or batAppendUpdate, or to learn the number that are deleted when the Mode (see page 3574) is batDelete.

Set the ChangedTableName (see page 3571) property to save the unchanged versions of the altered records to a separate table. ChangedCount is then the number of records in that separate table.

17.1.1.5.4 TIBBatchMove.ChangedTableName Property

Specifies the name of a local Interbase table that is created to hold copies of all records from the destination table that are changed by the batch operation.

Pascal

```
property ChangedTableName: TFileName;
```

C++

```
__property TFileName ChangedTableName;
```

Description

Set ChangedTableName to create a table that saves all unchanged versions of records that are changed by the batch operation. ChangedTableName must specify the name of a Interbase table. Setting ChangedTableName ensures that no information in the Destination (see page 3572) table gets lost when the records are altered or deleted.

When the Mode (see page 3574) property is batUpdate or batAppendUpdate, the new table will hold all records that were updated. When the Mode (see page 3574) property is batDelete, the new table will hold all records that were deleted.

The ChangedCount (see page 3571) property specifies the number of records in the new table.

17.1.1.5.5 TIBBatchMove.CommitCount Property

Specifies how many records are batch moved before a commit occurs.

Pascal

```
property CommitCount: Integer;
```

C++

```
__property int CommitCount;
```

Description

Use CommitCount to chunk batch operations into smaller operations. CommitCount allows an application to accommodate server transaction logs that are not big enough to handle the entire batch. Setting CommitCount to 0 will cause chunking into the number of records that will fit in 32K bytes.

Note: If the Execute (see page 3568) method is called within an explicit transaction on the database containing the tables, CommitCount is ignored.

17.1.1.5.6 TIBBatchMove.Destination Property

Specifies the TIBTable (see page 4142) object for the database table that is the destination of the batch operation.

Pascal

```
property Destination: TIBTable;
```

C++

```
__property TIBTable Destination;
```

Description

Use Destination to indicate the database table that will be created or altered by the batch operation. The batch operation affects the Destination differently depending on the Mode (see page 3574):

Mode (see page 3574) Effect on Destination

- batAppend Records are added to the destination table. No pre-existing records are altered. The destination table must already exist.
- batUpdate Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source records. No new records are added. The destination table must already exist and must have an index defined.
- batAppendUpdate Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source. Source (see page 3576) records that do not match key fields are added to the destination table. The destination table must already exist and must have an index defined.
- batDelete Records in the destination table that match the records in the source are deleted. The destination table must already exist and must have an index defined.
- batCopy The destination table is created to match the records in the source. If Destination already exists, it is replaced by the copy of the source.

17.1.1.5.7 TIBBatchMove.KeyViolCount Property

Reports the number of records that could not be replaced, added, or deleted from Destination (see page 3572) because of integrity or key violations.

Pascal

```
property KeyViolCount: Longint;
```

C++

```
__property Longint KeyViolCount;
```

Description

Use KeyViolCount to learn the number of records form the Source (see page 3576) table that could not be applied to the Destination (see page 3572) table because of integrity or key violations. To obtain a table of the offending records from the Source (see page 3576) table, specify a KeyViolTableName (see page 3573).

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

Note: If AbortOnKeyViol (see page 3570) is true, KeyViolCount will never be greater than one, because the first violation will cause the batch operation to terminate.

17.1.1.5.8 TIBBatchMove.KeyViolTableName Property

Specifies the name of an InterBase table that will be created to contain all records from the Source (see page 3576) that cannot participate in the batch operation because of integrity or key violations.

Pascal

```
property KeyViolTableName: TFileName;
```

C++

```
__property TFileName KeyViolTableName;
```

Description

Use KeyViolTableName to obtain a table of records that could not be applied to the Destination (see page 3572) table because of integrity or key violations. KeyViolCount (see page 3572) has the number of records in the table that is created.

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

Note: If AbortOnKeyViol (see page 3570) is true, then there will be at most one record in this table because the operation aborts with that first record.

17.1.1.5.9 TIBBatchMove.Mappings Property

Specifies the column mappings for a batch operation.

Pascal

```
property Mappings: TStrings;
```

C++

```
__property TStrings Mappings;
```

Description

Set Mappings to specify the correspondence between fields in the Source (see page 3576) and fields in the Destination (see page 3572). By default TIBBatchMove (see page 3566) matches fields based on their position in the source and destination tables. That is, the first column in the source is matched with the first column in the destination, and so on. Mappings enables an application to override this default.

Mappings is a list of column mappings (one per line) in one of two forms. To map the column ColName in the source table to the column of the same name in the destination table, use:

```
ColName
```

To map the column named SourceColName in the source table to the column named DestColName in the destination table, use:

```
DestColName=SourceColName
```

When adding or appending records, fields in Destination (see page 3572) which have no entry in Mappings will be set to NULL. When copying a dataset, fields in Destination (see page 3572) which have no entry in Mappings will not appear as columns in the copy of the table.

If source and destination column data types are not the same, TIBBatchMove (see page 3566) can either cancel the batch operation or perform a 'best fit.' If AbortOnProblem (see page 3570) is true, type mismatches will terminate the batch move. Otherwise, the field values from the Source (see page 3576) will be converted, if possible, to be compatible with the Destination (see page 3572).

17.1.1.5.10 TIBBatchMove.Mode Property

Specifies what the TIBBatchMove (see page 3566) object does when the Execute (see page 3568) method is called.

Pascal

```
property Mode: TBatchMode;
```

C++

```
__property TBatchMode Mode;
```

Description

Use Mode to indicate whether the TIBBatchMove (see page 3566) object should add records, replace records, delete records, or copy the Source (see page 3576). These are the possible values for Mode:

Value	Meaning
batAppend	Append the records in the source to the destination table. The destination table must already exist, and the two tables must not have records with duplicate keys. This is the default mode.
batUpdate	Replace records in the destination table with matching records from the source table. The destination table must exist and must have an index defined to match records.
batAppendUpdate	If a matching record exists in the destination table, replace it. Otherwise, append records to the destination table. The destination table must exist and must have an index defined to match records.
batCopy	Create (see page 3567) the destination table based on the structure of the source table. If the destination already exists, the operation will delete it, and replace it with the new copy of the source.
batDelete	Delete records in the destination table that match records in the source table. The destination table must already exist and must have an index defined.

17.1.1.5.11 TIBBatchMove.MovedCount Property

Reports the number of records form the Source (see page 3576) which were applied to the destination.

Pascal

```
property MovedCount: Longint;
```

C++

```
__property Longint MovedCount;
```

Description

Read MovedCount to learn the number of records from the Source (see page 3576) that were read during the Execute (see page 3568) method. The MovedCount value includes any records which caused key or integrity violations, or which were trimmed.

17.1.1.5.12 TIBBatchMove.ProblemCount Property

Indicates the number of records which could not be added to Destination (see page 3572) without loss of data due to a field type mismatch.

Pascal

```
property ProblemCount: Longint;
```

C++

```
__property Longint ProblemCount;
```

Description

Read ProblemCount to learn the number of records from the Source (see page 3576) that had field values that could not be mapped to destination fields without being 'trimmed'. If a ProblemTableName (see page 3575) is specified, ProblemCount is the number of records in that table.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TIBBatchMove (see page 3566) to raise an exception.

Trimming does not occur when the Mode (see page 3574) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table may be deleted that do not exactly match the values from the source table.

Note: If AbortOnProblem (see page 3570) is true, then ProblemCount will be at most one, because the operation aborts when the problem occurs.

17.1.1.5.13 TIBBatchMove.ProblemTableName Property

Specifies the name of a Interbase table that will be created to contain all records from the Source (see page 3576) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 3572) table.

Pascal

```
property ProblemTableName: TFileName;
```

C++

```
__property TFileName ProblemTableName;
```

Description

Set ProblemTableName to create a table that holds records from the Source (see page 3576) that contain fields that had to be trimmed to be applied to the Destination (see page 3572). ProblemCount (see page 3575) will have the number of records placed in the new table. If ProblemTableName is not specified, the data in the record is still trimmed when applied to the destination table, but there will be no easy way to tell which fields were trimmed.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TIBBatchMove (see page 3566) to raise an exception.

Trimming does not occur when the Mode (see page 3574) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table may be deleted that do not exactly match the values from the source table.

Note: If AbortOnProblem (see page 3570) is true, then there will be at most one record in this table because the operation aborts with the first record that has a problem.

17.1.1.5.14 TIBBatchMove.RecordCount Property

Specifies the maximum number of records that are applied to the Destination (see page 3572) when Execute (see page 3568) is called.

Pascal

```
property RecordCount: Longint;
```

C++

```
__property Longint RecordCount;
```

Description

Use RecordCount to limit the number of Source (see page 3576) records that are applied to the Destination (see page 3572). If zero, all records are added, subtracted, or copied, beginning with the first record in Source (see page 3576). If RecordCount is greater than zero, up to RecordCount records are applied to the Destination (see page 3572), starting with the current record in the dataset. If RecordCount exceeds the number of records remaining in Source (see page 3576), no wraparound occurs; the operation is terminated.

17.1.1.5.15 TIBBatchMove.Source Property

Specifies the dataset that is the source of the batch operation.

Pascal

```
property Source: TIBCustomDataSet;
```

C++

```
__property TIBCustomDataSet Source;
```

Description

Use Source to indicate the dataset that contains the records that will be added to, subtracted from, or copied to the Destination (see page 3572). If Source is a TIBQuery component, the query is executed before the move is executed. If a filter is currently in effect on Source, the filter conditions are respected when the batch move executes. The Mode (see page 3574) property indicates how the Source records are applied to the Destination (see page 3572):

Mode (see page 3574)	Effect on Destination (see page 3572)
batAppend	batAppend Records are added to the destination table.
batUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source records.

batAppendUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source. Source records that do not match key fields are added to the destination table.
batDelete	Records in the destination table that match the records in the source are deleted.
batCopy	Destination (see page 3572) table is created to match the records in the source.

17.1.1.5.16 TIBBatchMove.Transliterate Property

Specifies whether the data in the Source (see page 3576) records should be converted form the locale of the Source (see page 3576) to the locale of the Destination (see page 3572) when the Execute (see page 3568) method is called.

Pascal

```
property Transliterate: Boolean;
```

C++

```
__property Boolean Transliterate;
```

Description

Set Transliterate to true when the Source (see page 3576) dataset and the Destination (see page 3572) table use different language drivers and the data may contain extended ASCII characters. Set Transliterate to false to avoid the overhead of the character set conversion when both the Source (see page 3576) and the Destination (see page 3572) use the same language driver or the data does not contain extended ASCII characters.

17.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBatchMode (see page 3578)	This is record IBBatchMove.TBatchMode.

17.2.1 IBBatchMove.TBatchMode Enumeration

Pascal

```
TBatchMode = (
  batAppend,
  batUpdate,
  batAppendUpdate,
  batDelete,
  batCopy
);
```

C++

```
enum TBatchMode {
  batAppend,
  batUpdate,
  batAppendUpdate,
  batDelete,
  batCopy
};
```

File

IBBatchMove

Description

This is record IBBatchMove.TBatchMode.

18 IBBlob Namespace

This is namespace IBBlob.

Topics

Name	Description
Classes (see page 3580)	The following table lists classes in this documentation.
Functions (see page 3594)	The following table lists functions in this documentation.
Constants (see page 3596)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIBBlobStream (see page 3580)	TIBBlobStream is a stream object that lets applications read from or write to field objects that represent Blob fields.

Constants

Name	Description
DefaultBlobSegmentSize (see page 3596)	This is constant IBBlob.DefaultBlobSegmentSize.

Functions

	Name	Description
	GetBlobInfo (see page 3594)	This is function IBBlob.GetBlobInfo.
	ReadBlob (see page 3594)	This is function IBBlob.ReadBlob.
	WriteBlob (see page 3595)	This is function IBBlob.WriteBlob.

18.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBBlobStream (see page 3580)	TIBBlobStream is a stream object that lets applications read from or write to field objects that represent Blob fields.

18.1.1 TIBBlobStream Class

TIBBlobStream is a stream object that lets applications read from or write to field objects that represent Blob fields.

Class Hierarchy

Pascal

```
TIBBlobStream = class(TStream);
```

C++

```
class TIBBlobStream : public TStream;
```

File

IBBlob

Description

Use TIBBlobStream to access or modify the value of a Blob field object. Blob field objects are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. Blob fields use Blob streams to implement many of their data access properties and methods.

TIBBlobStream allows objects that have no specialized knowledge of how data is stored in a Blob field to read or write such data by employing the uniform stream interface.

To use a Blob stream, create an instance of TIBBlobStream, use the methods of the stream to read or write the data, and then free the Blob stream. Do not use the same instance of TBlobStream to access data from more than one record. Instead, create a new TIBBlobStream object every time you need to read or write Blob data on a new record.

18.1.1.1 TIBBlobStream Members

The following tables list the members exposed by TIBBlobStream.

Protected Methods

	Name	Description
	CloseBlob (see page 3584)	This is CloseBlob, a member of class TIBBlobStream.
	CreateBlob (see page 3584)	This is CreateBlob, a member of class TIBBlobStream.
	EnsureBlobInitialized (see page 3584)	This is EnsureBlobInitialized, a member of class TIBBlobStream.
	GetBlobInfo (see page 3585)	This is GetBlobInfo, a member of class TIBBlobStream.

	GetDatabase (see page 3585)	This is GetDatabase, a member of class TIBBlobStream.
	GetDBHandle (see page 3585)	This is GetDBHandle, a member of class TIBBlobStream.
	GetTransaction (see page 3585)	This is GetTransaction, a member of class TIBBlobStream.
	GetTRHandle (see page 3586)	This is GetTRHandle, a member of class TIBBlobStream.
	OpenBlob (see page 3586)	This is OpenBlob, a member of class TIBBlobStream.
	SetBlobID (see page 3588)	This is SetBlobID, a member of class TIBBlobStream.
	SetDatabase (see page 3588)	This is SetDatabase, a member of class TIBBlobStream.
	SetMode (see page 3589)	This is SetMode, a member of class TIBBlobStream.
	SetTransaction (see page 3589)	This is SetTransaction, a member of class TIBBlobStream.

Public Methods

	Name	Description
	Call (see page 3583)	Returns an error message based on the error code.
	Cancel (see page 3583)	Cancels all changes to the BLOB field, reverting to its state when the Blob stream was created.
	CheckReadable (see page 3583)	Indicates whether or not a Blob is readable.
	CheckWritable (see page 3584)	Indicates whether or not a Blob is write-able.
	Create (see page 3581)	Creates an instance of TIBBlobStream (see page 3580).
V	Destroy (see page 3582)	Destroys an instance of TIBBlobStream (see page 3580).
	Finalize (see page 3585)	Creates a Blob on the InterBase server and writes the data from the BlobStream to it.
	LoadFromFile (see page 3586)	Loads a Blob from a file to the field.
	LoadFromStream (see page 3586)	Loads a Blob from a stream into the field.
V	Read (see page 3587)	Reads up to Count bytes from the current position in the field's data into Buffer.
	SaveToFile (see page 3587)	Saves the contents of the Blob field to a file.
	SaveToStream (see page 3587)	Saves the contents of the BLOB field to a stream.
V	Seek (see page 3588)	Resets the current position of the TIBBlobStream (see page 3580) object.
V	SetSize (see page 3589)	Set the new total size of the Blob.
	Truncate (see page 3589)	Discards all data in the Blob field from the current position on.
V	Write (see page 3590)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

Public Properties

	Name	Description
	BlobID (see page 3590)	Sets or returns the Blob ID.
R	BlobMaxSegmentSize (see page 3591)	Returns the maximum segment size.
R	BlobNumSegments (see page 3591)	Returns the total number of segments in the Blob.
R	BlobSize (see page 3591)	Returns the total size of the Blob.
R	BlobType (see page 3591)	Returns the Blob type.
	Database (see page 3592)	Sets or returns the associated database.
R	DBHandle (see page 3592)	Indicates the database handle.
R	Handle (see page 3592)	Indicates the Blob handle.
	Mode (see page 3592)	Sets or returns the Blob stream mode type.
R	Modified (see page 3593)	Indicates whether the Blob field has been modified.
	Transaction (see page 3593)	Gets or sets the associated transaction.
R	TRHandle (see page 3593)	Indicates the transaction handle.

18.1.1.2 TIBBlobStream.Create Constructor

Creates an instance of TIBBlobStream (see page 3580).

Pascal

```
constructor Create;
```

C++

```
__fastcall TIBBlobStream();
```

Description

Call (see page 3583) Create to obtain an instance of TIBBlobStream (see page 3580) for reading from or writing to a specific TBlobField object.

18.1.1.3 TIBBlobStream.Destroy Destructor

Destroys an instance of TIBBlobStream (see page 3580).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBBlobStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TIBBlobStream (see page 3580) object is not already freed and only then calls Destroy.

Destroy the TIBBlobStream (see page 3580) object by calling Free when it is no longer needed for reading from or writing to the BLOB field.

Destroy triggers an OnDataChange event if the Blob stream was used to overwrite or modify the data in the field. Destroy then frees any buffers that were allocated to handle the data.

18.1.1.4 TIBBlobStream Methods

The methods of the TIBBlobStream class are listed here.

Protected Methods

	Name	Description
	CloseBlob (see page 3584)	This is CloseBlob, a member of class TIBBlobStream.
	CreateBlob (see page 3584)	This is CreateBlob, a member of class TIBBlobStream.
	EnsureBlobInitialized (see page 3584)	This is EnsureBlobInitialized, a member of class TIBBlobStream.
	GetBlobInfo (see page 3585)	This is GetBlobInfo, a member of class TIBBlobStream.
	GetDatabase (see page 3585)	This is GetDatabase, a member of class TIBBlobStream.
	GetDBHandle (see page 3585)	This is GetDBHandle, a member of class TIBBlobStream.
	GetTransaction (see page 3585)	This is GetTransaction, a member of class TIBBlobStream.
	GetTRHandle (see page 3586)	This is GetTRHandle, a member of class TIBBlobStream.
	OpenBlob (see page 3586)	This is OpenBlob, a member of class TIBBlobStream.
	SetBlobID (see page 3588)	This is SetBlobID, a member of class TIBBlobStream.
	SetDatabase (see page 3588)	This is SetDatabase, a member of class TIBBlobStream.
	SetMode (see page 3589)	This is SetMode, a member of class TIBBlobStream.
	SetTransaction (see page 3589)	This is SetTransaction, a member of class TIBBlobStream.

Public Methods

	Name	Description
	Call (see page 3583)	Returns an error message based on the error code.
	Cancel (see page 3583)	Cancels all changes to the BLOB field, reverting to its state when the Blob stream was created.

	CheckReadable (see page 3583)	Indicates whether or not a Blob is readable.
	CheckWritable (see page 3584)	Indicates whether or not a Blob is write-able.
	Finalize (see page 3585)	Creates a Blob on the InterBase server and writes the data from the BlobStream to it.
	LoadFromFile (see page 3586)	Loads a Blob from a file to the field.
	LoadFromStream (see page 3586)	Loads a Blob from a stream into the field.
	Read (see page 3587)	Reads up to Count bytes from the current position in the field's data into Buffer.
	SaveToFile (see page 3587)	Saves the contents of the Blob field to a file.
	SaveToStream (see page 3587)	Saves the contents of the BLOB field to a stream.
	Seek (see page 3588)	Resets the current position of the TIBBlobStream (see page 3580) object.
	SetSize (see page 3589)	Set the new total size of the Blob.
	Truncate (see page 3589)	Discards all data in the Blob field from the current position on.
	Write (see page 3590)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

18.1.1.4.1 TIBBlobStream.Call Method

Returns an error message based on the error code.

Pascal

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

C++

```
__fastcall ISC_STATUS Call(ISC_STATUS ErrCode, Boolean RaiseError);
```

Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

18.1.1.4.2 TIBBlobStream.Cancel Method

Cancels all changes to the BLOB field, reverting to its state when the Blob stream was created.

Pascal

```
procedure Cancel;
```

C++

```
__fastcall Cancel();
```

Description

Call (see page 3583) Cancel to back out all changes to the Blob field using this Blob stream. After a call to Cancel, the Modified (see page 3593) property is reset to false.

18.1.1.4.3 TIBBlobStream.CheckReadable Method

Indicates whether or not a Blob is readable.

Pascal

```
procedure CheckReadable;
```

C++

```
__fastcall CheckReadable();
```

Description

Call (see page 3583) CheckReadable to determine whether or not a Blob is readable. This method raises an exception if the Blob is not readable.

18.1.1.4.4 TIBBlobStream.CheckWritable Method

Indicates whether or not a Blob is write-able.

Pascal

```
procedure CheckWritable;
```

C++

```
__fastcall CheckWritable();
```

Description

Call (see page 3583) CheckWritable to determine whether or not a Blob is writeable. This method raises an exception if the Blob is not writeable.

18.1.1.4.5 TIBBlobStream.CloseBlob Method

Pascal

```
procedure CloseBlob;
```

C++

```
__fastcall CloseBlob();
```

Description

This is CloseBlob, a member of class TIBBlobStream.

18.1.1.4.6 TIBBlobStream.CreateBlob Method

Pascal

```
procedure CreateBlob;
```

C++

```
__fastcall CreateBlob();
```

Description

This is CreateBlob, a member of class TIBBlobStream.

18.1.1.4.7 TIBBlobStream.EnsureBlobInitialized Method

Pascal

```
procedure EnsureBlobInitialized;
```

C++

```
__fastcall EnsureBlobInitialized();
```

Description

This is EnsureBlobInitialized, a member of class TIBBlobStream.

18.1.1.4.8 TIBBlobStream.Finalize Method

Creates a Blob on the InterBase server and writes the data from the BlobStream to it.

Pascal

```
procedure Finalize;
```

C++

```
__fastcall Finalize();
```

Description

Finalize creates a Blob on the InterBase server and writes the data from the BlobStream to the Blob and then closes the Blob.

18.1.1.4.9 TIBBlobStream.GetBlobInfo Method

Pascal

```
procedure GetBlobInfo;
```

C++

```
__fastcall GetBlobInfo();
```

Description

This is GetBlobInfo, a member of class TIBBlobStream.

18.1.1.4.10 TIBBlobStream.GetDatabase Method

Pascal

```
function GetDatabase: TIBDatabase;
```

C++

```
__fastcall TIBDatabase GetDatabase();
```

Description

This is GetDatabase, a member of class TIBBlobStream.

18.1.1.4.11 TIBBlobStream.GetDBHandle Method

Pascal

```
function GetDBHandle: PISC_DB_HANDLE;
```

C++

```
__fastcall PISC_DB_HANDLE GetDBHandle();
```

Description

This is GetDBHandle, a member of class TIBBlobStream.

18.1.1.4.12 TIBBlobStream.GetTransaction Method

Pascal

```
function GetTransaction: TIBTransaction;
```

C++

```
__fastcall TIBTransaction GetTransaction();
```

Description

This is GetTransaction, a member of class TIBBlobStream.

18.1.1.4.13 TIBBlobStream.GetTRHandle Method

Pascal

```
function GetTRHandle: PISC_TR_HANDLE;
```

C++

```
__fastcall PISC_TR_HANDLE GetTRHandle();
```

Description

This is GetTRHandle, a member of class TIBBlobStream.

18.1.1.4.14 TIBBlobStream.LoadFromFile Method

Loads a Blob from a file to the field.

Pascal

```
procedure LoadFromFile(Filename: string);
```

C++

```
__fastcall LoadFromFile(UnicodeString Filename);
```

Description

Use LoadFromFile to load the contents of a file into a Blob field. Specify the name of the file to load into the field as the value of the FileName parameter.

18.1.1.4.15 TIBBlobStream.LoadFromStream Method

Loads a Blob from a stream into the field.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to copy the contents of a stream into the Blob field. Specify the stream from which the field's value is copied as the value of the Stream parameter.

18.1.1.4.16 TIBBlobStream.OpenBlob Method

Pascal

```
procedure OpenBlob;
```

C++

```
__fastcall OpenBlob();
```

Description

This is OpenBlob, a member of class TIBBlobStream.

18.1.1.4.17 TIBBlobStream.Read Method

Reads up to Count bytes from the current position in the field's data into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Call (see page 3583) Read to read data from the Blob field when the number of bytes in the field's data is not known. Buffer must have at least Count bytes allocated to hold the data that was read from the field.

Read transfers up to Count bytes from the Blob data into Buffer, starting in the current position, and then advances the current position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read checks the Transliterate property of the field, and converts the data into ANSI from the character set specified by the dataset if Transliterate is true.

All the other data-reading methods of a Blob stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

Do not call Read when the TIBBlobStream (see page 3580) was created in bmWrite mode.

18.1.1.4.18 TIBBlobStream.SaveToFile Method

Saves the contents of the Blob field to a file.

Pascal

```
procedure SaveToFile(Filename: string);
```

C++

```
__fastcall SaveToFile(UnicodeString Filename);
```

Description

Use SaveToFile to save the contents of the Blob field to a file. Specify the name of the file as the value of the FileName parameter.

18.1.1.4.19 TIBBlobStream.SaveToStream Method

Saves the contents of the BLOB field to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to copy the contents of a Blob field to a stream. Specify the name of the stream to which the field's value is

saved as the value of the Stream parameter.

18.1.1.4.20 TIBBlobStream.Seek Method

Resets the current position of the TIBBlobStream (see page 3580) object.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position within the Blob data by the indicated offset. Seek allows an application to read from or write to a particular location within the Blob data.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	Offset is from the beginning of the Blob data Seek moves to the position Offset Offset must be >= 0
soFromCurrent	Offset is from the current position in the Blob data Seek moves to Position + Offset
soFromEnd	Offset is from the end of the Blob data Offset must be <= 0 to indicate a number of bytes before the end of the Blob

Seek returns the new value of the Position property, the new current position in the Blob data.

18.1.1.4.21 TIBBlobStream.SetBlobID Method

Pascal

```
procedure SetBlobID(Value: TISC_QUAD);
```

C++

```
__fastcall SetBlobID(TISC_QUAD Value);
```

Description

This is SetBlobID, a member of class TIBBlobStream.

18.1.1.4.22 TIBBlobStream.SetDatabase Method

Pascal

```
procedure SetDatabase(Value: TIBDatabase);
```

C++

```
__fastcall SetDatabase(TIBDatabase Value);
```

Description

This is SetDatabase, a member of class TIBBlobStream.

18.1.1.4.23 TIBBlobStream.SetMode Method

Pascal

```
procedure SetMode(Value: TBlobStreamMode);
```

C++

```
__fastcall SetMode(TBlobStreamMode Value);
```

Description

This is SetMode, a member of class TIBBlobStream.

18.1.1.4.24 TIBBlobStream.SetSize Method

Set the new total size of the Blob.

Pascal

```
procedure SetSize(NewSize: Long); override;
```

C++

```
virtual __fastcall SetSize(Long NewSize);
```

Description

Call (see page 3583) SetSize to set the new total size of the Blob.

18.1.1.4.25 TIBBlobStream.SetTransaction Method

Pascal

```
procedure SetTransaction(Value: TIBTransaction);
```

C++

```
__fastcall SetTransaction(TIBTransaction Value);
```

Description

This is SetTransaction, a member of class TIBBlobStream.

18.1.1.4.26 TIBBlobStream.Truncate Method

Discards all data in the Blob field from the current position on.

Pascal

```
procedure Truncate;
```

C++

```
__fastcall Truncate();
```

Description

Use Truncate to limit the size of the Blob data. Calling Truncate when the current position is 0 will clear the contents of the Blob field.

Do not call Truncate when the TIBBlobStream (see page 3580) was created in bmRead mode.

18.1.1.4.27 TIBBlobStream.Write Method

Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write Count bytes to the Blob field, starting at the current position. The Write method for TIBBlobStream (see page 3580) always writes the entire Count bytes, as Blob data does not necessarily include a termination character. Thus, Write is equivalent to the WriteBuffer method.

Write checks the Transliterate property of the field, and converts the data from ANSI into the character set of the dataset if Transliterate is true.

All the other data-writing methods of a Blob stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

Do not call Write when the TIBBlobStream (see page 3580) was created in bmRead mode.

18.1.1.5 TIBBlobStream Properties

The properties of the TIBBlobStream class are listed here.

Public Properties

	Name	Description
	BlobID (see page 3590)	Sets or returns the Blob ID.
R	BlobMaxSegmentSize (see page 3591)	Returns the maximum segment size.
R	BlobNumSegments (see page 3591)	Returns the total number of segments in the Blob.
R	BlobSize (see page 3591)	Returns the total size of the Blob.
R	BlobType (see page 3591)	Returns the Blob type.
	Database (see page 3592)	Sets or returns the associated database.
R	DBHandle (see page 3592)	Indicates the database handle.
R	Handle (see page 3592)	Indicates the Blob handle.
	Mode (see page 3592)	Sets or returns the Blob stream mode type.
R	Modified (see page 3593)	Indicates whether the Blob field has been modified.
	Transaction (see page 3593)	Gets or sets the associated transaction.
R	TRHandle (see page 3593)	Indicates the transaction handle.

18.1.1.5.1 TIBBlobStream.BlobID Property

Sets or returns the Blob ID.

Pascal

```
property BlobID: TISC_QUAD;
```

C++

```
__property TISC_QUAD BlobID;
```

Description

Use BlobID to set or return the 64-bit system-defined Blob ID, which is stored in a field in the table and points to the first segment of the Blob or to a page of pointers to Blob fragments

18.1.1.5.2 TIBBlobStream.BlobMaxSegmentSize Property

Returns the maximum segment size.

Pascal

```
property BlobMaxSegmentSize: Long;
```

C++

```
__property Long BlobMaxSegmentSize;
```

Description

Use BlobMaxSegmentSize to return the length of the longest Blob segment.

18.1.1.5.3 TIBBlobStream.BlobNumSegments Property

Returns the total number of segments in the Blob.

Pascal

```
property BlobNumSegments: Long;
```

C++

```
__property Long BlobNumSegments;
```

Description

Use BlobNumSegments to return the total number of segments in the Blob.

18.1.1.5.4 TIBBlobStream.BlobSize Property

Returns the total size of the Blob.

Pascal

```
property BlobSize: Long;
```

C++

```
__property Long BlobSize;
```

Description

Use BlobSize to return the total size of the Blob in bytes.

18.1.1.5.5 TIBBlobStream.BlobType Property

Returns the Blob type.

Pascal

```
property BlobType: Short;
```

C++

```
__property Short BlobType;
```

Description

Use BlobType to return the Blob type; either 0 for segmented, or 1 for stream.

18.1.1.5.6 TIBBlobStream.Database Property

Sets or returns the associated database.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to set or return the associated database.

18.1.1.5.7 TIBBlobStream.DBHandle Property

Indicates the database handle.

Pascal

```
property DBHandle: PISC_DB_HANDLE;
```

C++

```
__property PISC_DB_HANDLE DBHandle;
```

Description

Use DBHandle to return the database handle.

18.1.1.5.8 TIBBlobStream.Handle Property

Indicates the Blob handle.

Pascal

```
property Handle: TISC_BLOB_HANDLE;
```

C++

```
__property TISC_BLOB_HANDLE Handle;
```

Description

Use Handle to return the Blob handle.

18.1.1.5.9 TIBBlobStream.Mode Property

Sets or returns the Blob stream mode type.

Pascal

```
property Mode: TBlobStreamMode;
```

C++

```
__property TBlobStreamMode Mode;
```

Description

Use Mode to set or read the Blob stream mode type. The Mode specifies whether the stream is used for reading BLOB values, writing BLOB values, or both.

18.1.1.5.10 TIBBlobStream.Modified Property

Indicates whether the Blob field has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Modified is true when the value to a Blob field has changed. If the value of the Blob field is set using the properties of TBlobField, or using a TIBBlobStream (see page 3580) object, Modified is automatically set to true.

18.1.1.5.11 TIBBlobStream.Transaction Property

Gets or sets the associated transaction.

Pascal

```
property Transaction: TIBTransaction;
```

C++

```
__property TIBTransaction Transaction;
```

Description

Use Transaction to set or read the associated transaction.

18.1.1.5.12 TIBBlobStream.TRHandle Property

Indicates the transaction handle.

Pascal

```
property TRHandle: PISC_TR_HANDLE;
```

C++

```
__property PISC_TR_HANDLE TRHandle;
```

Description

Use TRHandle to obtain the transaction handle.

18.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetBlobInfo (see page 3594)	This is function IBBlob.GetBlobInfo.
	ReadBlob (see page 3594)	This is function IBBlob.ReadBlob.
	WriteBlob (see page 3595)	This is function IBBlob.WriteBlob.

18.2.1 IBBlob.GetBlobInfo Function

Pascal

```
procedure GetBlobInfo(hBlobHandle: PISC_BLOB_HANDLE; var NumSegments: Long; var
MaxSegmentSize: Long; var TotalSize: Long; var BlobType: Short);
```

C++

```
GetBlobInfo(PISC_BLOB_HANDLE hBlobHandle, Long NumSegments, Long MaxSegmentSize, Long
TotalSize, Short BlobType);
```

File

IBBlob

Description

This is function IBBlob.GetBlobInfo.

18.2.2 IBBlob.ReadBlob Function

Pascal

```
procedure ReadBlob(hBlobHandle: PISC_BLOB_HANDLE; Buffer: PChar; BlobSize: Long);
```

C++

```
ReadBlob(PISC_BLOB_HANDLE hBlobHandle, const char * Buffer, Long BlobSize);
```

File

IBBlob

Description

This is function IBBlob.ReadBlob.

18.2.3 IBBlob.WriteBlob Function

Pascal

```
procedure WriteBlob(hBlobHandle: PISC_BLOB_HANDLE; Buffer: PChar; BlobSize: Long);
```

C++

```
WriteBlob(PISC_BLOB_HANDLE hBlobHandle, const char * Buffer, Long BlobSize);
```

File

IBBlob

Description

This is function IBBlob.WriteBlob.

18.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DefaultBlobSegmentSize (see page 3596)	This is constant IBBlob.DefaultBlobSegmentSize.

18.3.1 IBBlob.DefaultBlobSegmentSize Constant

Pascal

```
DefaultBlobSegmentSize = 16 * 1024;
```

C++

```
DefaultBlobSegmentSize = 16 * 1024;
```

File

IBBlob

Description

This is constant IBBlob.DefaultBlobSegmentSize.

19 IBCustomDataSet Namespace

This is namespace IBCustomDataSet.

Topics

Name	Description
Classes (see page 3599)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3689)	The following table lists structs, records, enums in this documentation.
Types (see page 3694)	The following table lists types in this documentation.
Variables (see page 3699)	The following table lists variables in this documentation.
Constants (see page 3700)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIBBCDField (see page 3599)	TIBBCDField represents a numeric field as a Currency type..
	TIBCustomDataSet (see page 3602)	TIBCustomDataSet is the base class for all datasets that represent data fetched using InterBase Express.
	TIBDataLink (see page 3651)	TIBDataLink is a helper class used by data-aware objects to coordinate the actions of TIBDataSet (see page 3654) and to respond to data events.
	TIBDataSet (see page 3654)	TIBDataSet executes InterBase SQL statements.
	TIBDataSetUpdateObject (see page 3674)	TIBDataSetUpdateObject is the abstract base class for update objects used to update otherwise ununpdateable queries when cached updates are enabled.
	TIBDSBlobStream (see page 3678)	TIBDSBlobStream reads from and writes to BLOB fields on behalf of an IB (see page 3541) Dataset.
	TIBGeneratorField (see page 3682)	TIBGeneratorField calls the server to generate values for a field in an IB (see page 3541) dataset.
	TIBStringField (see page 3686)	TIBStringField allows for strings in excess of 8196 bytes of data.

Constants

Name	Description
BufferCacheSize (see page 3700)	This is constant IBCustomDataSet.BufferCacheSize.
DefaultFieldClasses (see page 3700)	This is constant IBCustomDataSet.DefaultFieldClasses.
UniCache (see page 3701)	This is constant IBCustomDataSet.UniCache.

Structs, Records, Enums

	Name	Description
	TCachedUpdateStatus (see page 3689)	TCachedUpdateStatus indicates the status of a record in a dataset that has cached updates.
	TFieldData (see page 3690)	This is record IBCustomDataSet.TFieldData.

	TIBDBKey (see page 3690)	TIBDBKey stores the key for a row in an InterBase table.
	TIBGeneratorApplyEvent (see page 3691)	TIBGeneratorApplyEvent indicates when a generator should create field values.
	TIBUpdateAction (see page 3691)	TIBUpdateAction indicates the action that a dataset should take after an event handler has responded to an update error.
	TLiveMode (see page 3692)	This is record IBCustomDataSet.TLiveMode.
	TRecordData (see page 3693)	This is record IBCustomDataSet.TRecordData.

Types

Name	Description
PBlobDataArray (see page 3694)	This is type IBCustomDataSet.PBlobDataArray.
PDateTime (see page 3694)	This is type IBCustomDataSet.PDateTime.
PFieldData (see page 3695)	This is type IBCustomDataSet.PFieldData.
PIBDBKey (see page 3695)	This is type IBCustomDataSet.PIBDBKey.
PRecordData (see page 3695)	This is type IBCustomDataSet.PRecordData.
TBlobDataArray (see page 3696)	This is type IBCustomDataSet.TBlobDataArray.
TIBUpdateErrorEvent (see page 3696)	TIBUpdateErrorEvent is the type of event handlers that respond when an IBX dataset encounters an error applying a cached update.
TIBUpdateRecordEvent (see page 3696)	TIBUpdateRecordEvent is the type of event handlers that respond when an IBX dataset applies a cached update.
TIBUpdateRecordTypes (see page 3697)	TIBUpdateRecordTypes is a subset of all possible record types in a dataset that caches updates.
TLiveModes (see page 3697)	TLiveModes and TLiveModes indicate what operations an application can perform on an IBX dataset.

Variables

Name	Description
CreateProviderProc (see page 3699)	This is variable IBCustomDataSet.CreateProviderProc.

19.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBBCDField (see page 3599)	TIBBCDField represents a numeric field as a Currency type..
	TIBCustomDataSet (see page 3602)	TIBCustomDataSet is the base class for all datasets that represent data fetched using InterBase Express.
	TIBDataLink (see page 3651)	TIBDataLink is a helper class used by data-aware objects to coordinate the actions of TIBDataSet (see page 3654) and to respond to data events.
	TIBDataSet (see page 3654)	TIBDataSet executes InterBase SQL statements.
	TIBDataSetUpdateObject (see page 3674)	TIBDataSetUpdateObject is the abstract base class for update objects used to update otherwise unupdateable queries when cached updates are enabled.
	TIBDSBlobStream (see page 3678)	TIBDSBlobStream reads from and writes to BLOB fields on behalf of an IB (see page 3541) Dataset.
	TIBGeneratorField (see page 3682)	TIBGeneratorField calls the server to generate values for a field in an IB (see page 3541) dataset.
	TIBStringField (see page 3686)	TIBStringField allows for strings in excess of 8196 bytes of data.

19.1.1 TIBBCDField Class

TIBBCDField represents a numeric field as a Currency type..

Class Hierarchy

Pascal

```
TIBBCDField = class(TBCDField);
```

C++

```
class TIBBCDField : public TBCDField;
```

File

IBCustomDataSet

Description

TIBBCDField encapsulates the Currency type. Despite the name, TIBBCDField does not represent a BCD value. Rather, TIBBCDFields represent InterBase Numeric(x, y) fields, where x < 18 and y <= 4.

19.1.1.1 TIBBCDField Members

The following tables list the members exposed by TIBBCDField.

Protected Methods

	Name	Description
	CheckTypeSize (see page 3600)	This is CheckTypeSize, a member of class TIBBCDField.
	GetAsCurrency (see page 3601)	This is GetAsCurrency, a member of class TIBBCDField.

	GetAsString (see page 3601)	This is GetAsString, a member of class TIBBCDField.
	GetAsVariant (see page 3601)	This is GetAsVariant, a member of class TIBBCDField.
	GetDataSize (see page 3601)	This is GetDataSize, a member of class TIBBCDField.

Public Methods

	Name	Description
	Create (see page 3600)	Creates an instance of a TIBBCDField (see page 3599) object.

Published Properties

	Name	Description
	Size (see page 3602)	This is Size, a member of class TIBBCDField.

19.1.1.2 TIBBCDField.Create Constructor

Creates an instance of a TIBBCDField (see page 3599) object.

Pascal

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

C++

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

Description

Create sets the datatype to ftBCD and the size to 8.

19.1.1.3 TIBBCDField Methods

The methods of the TIBBCDField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 3600)	This is CheckTypeSize, a member of class TIBBCDField.
	GetAsCurrency (see page 3601)	This is GetAsCurrency, a member of class TIBBCDField.
	GetAsString (see page 3601)	This is GetAsString, a member of class TIBBCDField.
	GetAsVariant (see page 3601)	This is GetAsVariant, a member of class TIBBCDField.
	GetDataSize (see page 3601)	This is GetDataSize, a member of class TIBBCDField.

19.1.1.3.1 TIBBCDField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TIBBCDField.

19.1.1.3.2 TIBBCDField.GetAsCurrency Method

Pascal

```
function GetAsCurrency: Currency; override;
```

C++

```
virtual __fastcall Currency GetAsCurrency();
```

Description

This is GetAsCurrency, a member of class TIBBCDField.

19.1.1.3.3 TIBBCDField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall UnicodeString GetAsString();
```

Description

This is GetAsString, a member of class TIBBCDField.

19.1.1.3.4 TIBBCDField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TIBBCDField.

19.1.1.3.5 TIBBCDField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TIBBCDField.

19.1.1.4 TIBBCDField Properties

The properties of the TIBBCDField class are listed here.

Published Properties

	Name	Description
	Size (see page 3602)	This is Size, a member of class TIBBCDField.

19.1.1.4.1 TIBBCDField.Size Property

Pascal

```
property Size;
```

C++

```
__property Size;
```

Description

This is Size, a member of class TIBBCDField.

19.1.2 TIBCustomDataSet Class

TIBCustomDataSet is the base class for all datasets that represent data fetched using InterBase Express.

Class Hierarchy

Pascal

```
TIBCustomDataSet = class(TDataset);
```

C++

```
class TIBCustomDataSet : public TDataset;
```

File

IBCustomDataSet

Description

TIBCustomDataSet is a dataset object that defines InterBase Express (IBX) functionality for a dataset. Applications never use TIBCustomDataSet objects directly. Instead they use the descendants of TIBCustomDataSet, such as TIBDataSet (see page 3654), TIBQuery, TIBStoredProc, and TIBTable, which inherit its dataset-related properties and methods.

See Also

Getting Started with InterBase Express

19.1.2.1 TIBCustomDataSet Members

The following tables list the members exposed by TIBCustomDataSet.

Protected Events

	Name	Description
	AfterDatabaseDisconnect (see page 3649)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3649)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3649)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3649)	Occurs immediately before a transaction that includes this dataset's query ends.

	DatabaseFree (see page 3650)	Occurs when the database associated with this dataset is freed.
	TransactionFree (see page 3651)	Occurs when the transaction that includes this dataset's query is freed.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.
	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.
V	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
V	Disconnect (see page 3613)	This is Disconnect, a member of class TIBCustomDataSet.
V	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
V	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
V	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
V	DoOnNewRecord (see page 3614)	This is DoOnNewRecord, a member of class TIBCustomDataSet.
V	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
V	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
V	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
V	GetCanModify (see page 3615)	This is GetCanModify, a member of class TIBCustomDataSet.
V	GetDataSource (see page 3616)	This is GetDataSource, a member of class TIBCustomDataSet.
V	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
V	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
V	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
V	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
V	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
V	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
V	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
V	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
V	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.
V	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
V	InternalClose (see page 3619)	This is InternalClose, a member of class TIBCustomDataSet.
V	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
V	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
V	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
V	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
V	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
V	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
V	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
V	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
V	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
V	InternalOpen (see page 3621)	This is InternalOpen, a member of class TIBCustomDataSet.
V	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
V	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
V	InternalRefresh (see page 3622)	This is InternalRefresh, a member of class TIBCustomDataSet.
V	InternalRefreshRow (see page 3622)	This is InternalRefreshRow, a member of class TIBCustomDataSet.
V	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
V	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.
V	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.

	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.
	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
	PSGetTableName (see page 3627)	This is PSGetTableName, a member of class TIBCustomDataSet.
	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.
	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.

Protected Properties

	Name	Description
	BufferChunks (see page 3638)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3638)	Specifies whether cached updates are enabled for a dataset.
	DeleteSQL (see page 3639)	Holds the SQL statement used to delete rows from the dataset.
	GeneratorField (see page 3640)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3640)	Holds the SQL statement used to insert rows into the dataset.
R	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
R	LiveMode (see page 3641)	Indicates what types of edits, if any, can be made to the data in the dataset.
	ModifySQL (see page 3641)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	ParamCheck (see page 3642)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
R	Params (see page 3643)	Specifies the parameters to use for any parameterized queries.
R	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.
R	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
R	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
R	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
R	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	RefreshSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
R	SelectStmtHandle (see page 3645)	Gets the handle for the SELECT statement.
R	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
R	StatementType (see page 3646)	Returns the statement type of the QSelect (see page 3644) query.
	UniDirectional (see page 3647)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Methods

	Name	Description
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.
V	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.
	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
V	CompareBookmarks (see page 3611)	Compares two bookmarks.
V	Create (see page 3606)	Creates an instance of a TDataSet component.
V	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
V	Destroy (see page 3606)	Destroys the instance of a dataset component.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
V	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
V	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.
V	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
V	Locate (see page 3624)	Searches the dataset for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
V	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.
V	Post (see page 3626)	Writes a modified record to the database server.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
V	UpdateStatus (see page 3632)	Reports the update status for the current record.

Public Properties

	Name	Description
R	DBHandle (see page 3639)	Specifies the database handle for the dataset.
R	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
R	RowsAffected (see page 3645)	Indicates the number of rows operated on by the latest query execution.
R	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateObject (see page 3647)	Specifies the update object component used to update a read-only result set.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Events

	Name	Description
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	AfterCancel (see page 3634)	This is AfterCancel, a member of class TIBCustomDataSet.
	AfterClose (see page 3634)	This is AfterClose, a member of class TIBCustomDataSet.
	AfterDelete (see page 3634)	This is AfterDelete, a member of class TIBCustomDataSet.
	AfterEdit (see page 3634)	This is AfterEdit, a member of class TIBCustomDataSet.
	AfterInsert (see page 3635)	This is AfterInsert, a member of class TIBCustomDataSet.
	AfterOpen (see page 3635)	This is AfterOpen, a member of class TIBCustomDataSet.

	AfterPost (see page 3635)	This is AfterPost, a member of class TIBCustomDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3635)	This is AfterScroll, a member of class TIBCustomDataSet.
	AutoCalcFields (see page 3636)	This is AutoCalcFields, a member of class TIBCustomDataSet.
	BeforeCancel (see page 3636)	This is BeforeCancel, a member of class TIBCustomDataSet.
	BeforeClose (see page 3636)	This is BeforeClose, a member of class TIBCustomDataSet.
	BeforeDelete (see page 3636)	This is BeforeDelete, a member of class TIBCustomDataSet.
	BeforeEdit (see page 3637)	This is BeforeEdit, a member of class TIBCustomDataSet.
	BeforeInsert (see page 3637)	This is BeforeInsert, a member of class TIBCustomDataSet.
	BeforeOpen (see page 3637)	This is BeforeOpen, a member of class TIBCustomDataSet.
	BeforePost (see page 3637)	This is BeforePost, a member of class TIBCustomDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3638)	This is BeforeScroll, a member of class TIBCustomDataSet.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3641)	This is OnCalcFields, a member of class TIBCustomDataSet.
	OnDeleteError (see page 3642)	This is OnDeleteError, a member of class TIBCustomDataSet.
	OnEditError (see page 3642)	This is OnEditError, a member of class TIBCustomDataSet.
	OnNewRecord (see page 3642)	This is OnNewRecord, a member of class TIBCustomDataSet.
	OnPostError (see page 3642)	This is OnPostError, a member of class TIBCustomDataSet.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.

19.1.2.2 TIBCustomDataSet.Create Constructor

Creates an instance of a TDataSet component.

Pascal

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

C++

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

Description

Call Create to instantiate a dataset component at runtime. Ordinarily applications instantiate dataset descendants, such as TIBTable, TIBQuery, TIBDataSet (see page 3654), or TIBStoredProc, rather than TIBCustomDataSet (see page 3602). These instantiated objects are handled automatically.

On the other hand, applications that create specialized dataset components, such as custom components, may need to instantiate a TIBCustomDataSet (see page 3602) component by calling Create. Create:

- Calls the inherited Create for TDataSet.
- Creates the five query objects.
- Creates the base object to hold the reference to the database and transaction.
- Associates database and transaction related events to the base object component.

19.1.2.3 TIBCustomDataSet.Destroy Destructor

Destroys the instance of a dataset component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBCustomDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, the application should call Free, which verifies that the dataset component is not already freed before calling Destroy.

Destroy performs the following tasks:

- Frees the associated query objects.
- Frees the base object.
- Frees the associated internal resources.

19.1.2.4 TIBCustomDataSet Methods

The methods of the TIBCustomDataSet class are listed here.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.
	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.
V	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
V	Disconnect (see page 3613)	This is Disconnect, a member of class TIBCustomDataSet.
V	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
V	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
V	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
V	DoOnNewRecord (see page 3614)	This is DoOnNewRecord, a member of class TIBCustomDataSet.
V	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
V	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
V	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
V	GetCanModify (see page 3615)	This is GetCanModify, a member of class TIBCustomDataSet.
V	GetDataSource (see page 3616)	This is GetDataSource, a member of class TIBCustomDataSet.
V	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
V	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
V	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
V	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
V	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
V	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
V	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
V	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
V	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.

	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
	InternalClose (see page 3619)	This is InternalClose, a member of class TIBCustomDataSet.
	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
	InternalOpen (see page 3621)	This is InternalOpen, a member of class TIBCustomDataSet.
	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
	InternalRefresh (see page 3622)	This is InternalRefresh, a member of class TIBCustomDataSet.
	InternalRefreshRow (see page 3622)	This is InternalRefreshRow, a member of class TIBCustomDataSet.
	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.
	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.
	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.
	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
	PSGetTableName (see page 3627)	This is PSGetTableName, a member of class TIBCustomDataSet.
	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.
	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.

Public Methods

	Name	Description
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.
	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.
	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
	CompareBookmarks (see page 3611)	Compares two bookmarks.
	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.

	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
	Locate (see page 3624)	Searches the dataset for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.
	Post (see page 3626)	Writes a modified record to the database server.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
	UpdateStatus (see page 3632)	Reports the update status for the current record.

19.1.2.4.1 TIBCustomDataSet.ActivateConnection Method

Pascal

```
procedure ActivateConnection;
```

C++

```
__fastcall ActivateConnection();
```

Description

This is ActivateConnection, a member of class TIBCustomDataSet.

19.1.2.4.2 TIBCustomDataSet.ActivateTransaction Method

Pascal

```
function ActivateTransaction: Boolean;
```

C++

```
__fastcall Boolean ActivateTransaction();
```

Description

This is ActivateTransaction, a member of class TIBCustomDataSet.

19.1.2.4.3 TIBCustomDataSet.AllocRecordBuffer Method

Pascal

```
function AllocRecordBuffer: PChar; override;
```

C++

```
virtual __fastcall const char * AllocRecordBuffer();
```

Description

This is AllocRecordBuffer, a member of class TIBCustomDataSet.

19.1.2.4.4 TIBCustomDataSet.ApplyUpdates Method

Writes a dataset's pending cached updates to the database.

Pascal

```
procedure ApplyUpdates;
```

C++

```
__fastcall ApplyUpdates();
```

Description

Call ApplyUpdates to write a dataset's pending cached updates to a database. This method passes cached data to the database for storage, but the changes are not committed to the database. An application must explicitly call the TIBTransaction (see page 3746) component's Commit method to commit the changes to the database if the write is successful, or call the TIBTransaction (see page 3746) component's Rollback method to undo the changes if there is an error.

Note: The preferred method for updating datasets is to call a database component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The application is responsible for committing or rolling back the transaction.

19.1.2.4.5 TIBCustomDataSet.BookmarkValid Method

Indicates the validity of a specified bookmark.

Pascal

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

C++

```
virtual __fastcall Boolean BookmarkValid(TBookmark Bookmark);
```

Description

This function returns true if the specified bookmark can be used to identify a record in the dataset, false otherwise.

19.1.2.4.6 TIBCustomDataSet.CachedUpdateStatus Method

Returns the status of the cached updates.

Pascal

```
function CachedUpdateStatus: TCachedUpdateStatus;
```

C++

```
__fastcall TCachedUpdateStatus CachedUpdateStatus();
```

Description

Call CachedUpdateStatus to return the cached update status of the current record in the dataset.

19.1.2.4.7 TIBCustomDataSet.CancelUpdates Method

Clears all pending cached updates from the cache.

Pascal

```
procedure CancelUpdates;
```

C++

```
__fastcall CancelUpdates();
```

Description

Call CancelUpdates to clear all pending cached updates from the cache.

When a dataset is closed, or the CachedUpdates (see page 3638) property is set to false, CancelUpdates is called automatically.

Note: To undo changes to a single record, call RevertRecord (see page 3629).

19.1.2.4.8 TIBCustomDataSet.CheckDatasetClosed Method

Pascal

```
procedure CheckDatasetClosed;
```

C++

```
__fastcall CheckDatasetClosed();
```

Description

This is CheckDatasetClosed, a member of class TIBCustomDataSet.

19.1.2.4.9 TIBCustomDataSet.CheckDatasetOpen Method

Pascal

```
procedure CheckDatasetOpen;
```

C++

```
__fastcall CheckDatasetOpen();
```

Description

This is CheckDatasetOpen, a member of class TIBCustomDataSet.

19.1.2.4.10 TIBCustomDataSet.CheckNotUniDirectional Method

Pascal

```
procedure CheckNotUniDirectional;
```

C++

```
__fastcall CheckNotUniDirectional();
```

Description

This is CheckNotUniDirectional, a member of class TIBCustomDataSet.

19.1.2.4.11 TIBCustomDataSet.ClearCalcFields Method

Pascal

```
procedure ClearCalcFields(Buffer: PChar); override;
```

C++

```
virtual __fastcall ClearCalcFields(const char * Buffer);
```

Description

This is ClearCalcFields, a member of class TIBCustomDataSet.

19.1.2.4.12 TIBCustomDataSet.CompareBookmarks Method

Compares two bookmarks.

Pascal

```
function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark): Integer; override;
```

C++

```
virtual __fastcall int CompareBookmarks(TBookmark Bookmark1, TBookmark Bookmark2);
```

Description

Call CompareBookmarks to compare one bookmark to another. CompareBookmarks returns an integer value indicating the difference between the two bookmarks. CompareBookmarks returns a 0 (zero) if the two bookmarks are the same, a -1 (negative one) if bookmark one is less than bookmark two, or a 1 (positive one) if bookmark one is greater than bookmark two. CompareBookmarks also returns a zero if one of the two bookmarks is nil (Delphi) or NULL (C++).

Bookmark1 and Bookmark2 are the two TBookmark values to compare.

19.1.2.4.13 TIBCustomDataSet.ConstraintsStored Method

Pascal

```
function ConstraintsStored: Boolean;
```

C++

```
__fastcall Boolean ConstraintsStored();
```

Description

This is ConstraintsStored, a member of class TIBCustomDataSet.

19.1.2.4.14 TIBCustomDataSet.CreateBlobStream Method

Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading data from or writing data to a binary large object (BLOB) field. The Field parameter must specify a TBlobField component from the Fields property array. The Mode parameter specifies whether the stream will be used for reading, writing, or updating the contents of the field.

Note: The stream that this method returns belongs to the dataset. Do not free the stream: the dataset handles all allocation and freeing of streams that this method creates.

19.1.2.4.15 TIBCustomDataSet.CreateFields Method

Pascal

```
procedure CreateFields; override;
```

C++

```
virtual __fastcall CreateFields();
```

Description

This is CreateFields, a member of class TIBCustomDataSet.

19.1.2.4.16 TIBCustomDataSet.Current Method

Returns the descriptor area that holds information about the current query.

Pascal

```
function Current: TIBXSQLDA;
```

C++

```
__fastcall TIBXSQLDA Current();
```

Description

Most applications do not need to call the Current method. It returns the extended descriptor area for the dataset's SQL statement, which can be used to retrieve properties of the SQL statement.

19.1.2.4.17 TIBCustomDataSet.DeactivateTransaction Method

Pascal

```
procedure DeactivateTransaction;
```

C++

```
__fastcall DeactivateTransaction();
```

Description

This is DeactivateTransaction, a member of class TIBCustomDataSet.

19.1.2.4.18 TIBCustomDataSet.Disconnect Method

Pascal

```
procedure Disconnect; virtual;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TIBCustomDataSet.

19.1.2.4.19 TIBCustomDataSet.DoBeforeDelete Method

Pascal

```
procedure DoBeforeDelete; override;
```

C++

```
virtual __fastcall DoBeforeDelete();
```

Description

This is DoBeforeDelete, a member of class TIBCustomDataSet.

19.1.2.4.20 TIBCustomDataSet.DoBeforeEdit Method

Pascal

```
procedure DoBeforeEdit; override;
```

C++

```
virtual __fastcall DoBeforeEdit();
```

Description

This is DoBeforeEdit, a member of class TIBCustomDataSet.

19.1.2.4.21 TIBCustomDataSet.DoBeforeInsert Method

Pascal

```
procedure DoBeforeInsert; override;
```

C++

```
virtual __fastcall DoBeforeInsert();
```

Description

This is DoBeforeInsert, a member of class TIBCustomDataSet.

19.1.2.4.22 TIBCustomDataSet.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; override;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TIBCustomDataSet.

19.1.2.4.23 TIBCustomDataSet.FetchAll Method

Retrieves all records from the current cursor position to the end of the file and stores them locally.

Pascal

```
procedure FetchAll;
```

C++

```
__fastcall FetchAll();
```

Description

Call FetchAll to reduce network traffic when using cached updates. FetchAll calls CheckBrowseMode to post any pending changes, and then retrieves all records from the current cursor position to the end of the file, and stores them locally. Ordinarily when cached updates are enabled, a transaction retrieves only as much data as it needs for return purposes.

Note: Using FetchAll is not always appropriate. For example, when an application accesses a database used by many simultaneous clients and there is a high degree of contention for updating the same records, fetching all records at once may not be advantageous because some fetched records may be changed by other applications. Always weigh the advantages of reduced network traffic against the need for reduced record contention.

19.1.2.4.24 TIBCustomDataSet.FreeRecordBuffer Method

Pascal

```
procedure FreeRecordBuffer(var Buffer: PChar); override;
```

C++

```
virtual __fastcall FreeRecordBuffer(const char * Buffer);
```

Description

This is FreeRecordBuffer, a member of class TIBCustomDataSet.

19.1.2.4.25 TIBCustomDataSet.GetActiveBuf Method

Pascal

```
function GetActiveBuf: PChar;
```

C++

```
__fastcall const char * GetActiveBuf();
```

Description

This is GetActiveBuf, a member of class TIBCustomDataSet.

19.1.2.4.26 TIBCustomDataSet.GetBookmarkData Method

Pascal

```
procedure GetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall GetBookmarkData(const char * Buffer, void * Data);
```

Description

This is GetBookmarkData, a member of class TIBCustomDataSet.

19.1.2.4.27 TIBCustomDataSet.GetBookmarkFlag Method

Pascal

```
function GetBookmarkFlag(Buffer: PChar): TBookmarkFlag; override;
```

C++

```
virtual __fastcall TBookmarkFlag GetBookmarkFlag(const char * Buffer);
```

Description

This is GetBookmarkFlag, a member of class TIBCustomDataSet.

19.1.2.4.28 TIBCustomDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TIBCustomDataSet.

19.1.2.4.29 TIBCustomDataSet.GetCurrentRecord Method

Retrieves the current record into a buffer.

Pascal

```
function GetCurrentRecord(Buffer: PChar): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCurrentRecord(const char * Buffer);
```

Description

Most applications should not need to call GetCurrentRecord. TDataSet automatically allocates a buffer for the active record.

Call GetCurrentRecord to copy the current record into a buffer allocated by the application. Buffer must be at least as big as the record size indicated by the RecordSize property.

19.1.2.4.30 TIBCustomDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TIBCustomDataSet.

19.1.2.4.31 TIBCustomDataSet.GetFieldClass Method

Pascal

```
function GetFieldClass(FieldType: TFieldType): TFieldClass; override;
```

C++

```
virtual __fastcall TFieldClass GetFieldClass(TFieldType FieldType);
```

Description

This is GetFieldClass, a member of class TIBCustomDataSet.

19.1.2.4.32 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TIBCustomDataSet.GetFieldData (TField, Pointer) (see page 3617)	Reads the field data into a buffer.
	TIBCustomDataSet.GetFieldData (TField, Pointer, Boolean) (see page 3617)	Reads the field data into a buffer.

19.1.2.4.32.1 TIBCustomDataSet.GetFieldData Method (TField, Pointer)

Reads the field data into a buffer.

Pascal

```
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; overload;
function GetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean): Boolean;
override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
```

Description

Reads the field data into a buffer.

Delphi syntax:

```
        function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; overload;
    function GetFieldData(Field: TField; Buffer: Pointer): Boolean; overload; override;
    function GetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean): Boolean;
overload; override;
```

C++ syntax:

```
        virtual bool __fastcall GetFieldData(Db::TField* Field, void * Buffer);
    virtual bool __fastcall GetFieldData(Db::TField* Field, void * Buffer, bool NativeFormat);
    virtual bool __fastcall GetFieldData(int FieldNo, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number. The Buffer parameter is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). NativeFormat indicates whether the dataset fetches the field in Delphi's native format for the field type. When NativeFormat is false, the dataset must convert the field value to the native type. This allows the field to handle data from different types of datasets (IBX, BDE-based, and so on) in a uniform manner.

GetFieldData returns true if the data was successfully written to Buffer.

19.1.2.4.33 TIBCustomDataSet.GetRecNo Method

Pascal

```
function GetRecNo: Integer; override;
```

C++

```
virtual __fastcall int GetRecNo();
```

Description

This is GetRecNo, a member of class TIBCustomDataSet.

19.1.2.4.34 TIBCustomDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; override;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TIBCustomDataSet.

19.1.2.4.35 TIBCustomDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TIBCustomDataSet.

19.1.2.4.36 TIBCustomDataSet.GetRecordSize Method

Pascal

```
function GetRecordSize: Word; override;
```

C++

```
virtual __fastcall Word GetRecordSize();
```

Description

This is GetRecordSize, a member of class TIBCustomDataSet.

19.1.2.4.37 TIBCustomDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InitRecord(const char * Buffer);
```

Description

This is InitRecord, a member of class TIBCustomDataSet.

19.1.2.4.38 TIBCustomDataSet.InternalAddRecord Method

Pascal

```
procedure InternalAddRecord(Buffer: Pointer; Append: Boolean); override;
```

C++

```
virtual __fastcall InternalAddRecord(void * Buffer, Boolean Append);
```

Description

This is InternalAddRecord, a member of class TIBCustomDataSet.

19.1.2.4.39 TIBCustomDataSet.InternalBatchInput Method

Pascal

```
procedure InternalBatchInput(InputObject: TIBBatchInput); virtual;
```

C++

```
virtual __fastcall InternalBatchInput(TIBBatchInput InputObject);
```

Description

This is InternalBatchInput, a member of class TIBCustomDataSet.

19.1.2.4.40 TIBCustomDataSet.InternalBatchOutput Method

Pascal

```
procedure InternalBatchOutput(OutputObject: TIBBatchOutput); virtual;
```

C++

```
virtual __fastcall InternalBatchOutput(TIBBatchOutput OutputObject);
```

Description

This is InternalBatchOutput, a member of class TIBCustomDataSet.

19.1.2.4.41 TIBCustomDataSet.InternalCancel Method

Pascal

```
procedure InternalCancel; override;
```

C++

```
virtual __fastcall InternalCancel();
```

Description

This is InternalCancel, a member of class TIBCustomDataSet.

19.1.2.4.42 TIBCustomDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TIBCustomDataSet.

19.1.2.4.43 TIBCustomDataSet.InternalDelete Method

Pascal

```
procedure InternalDelete; override;
```

C++

```
virtual __fastcall InternalDelete();
```

Description

This is InternalDelete, a member of class TIBCustomDataSet.

19.1.2.4.44 TIBCustomDataSet.InternalExecQuery Method

Pascal

```
procedure InternalExecQuery; virtual;
```

C++

```
virtual __fastcall InternalExecQuery();
```

Description

This is InternalExecQuery, a member of class TIBCustomDataSet.

19.1.2.4.45 TIBCustomDataSet.InternalFirst Method

Pascal

```
procedure InternalFirst; override;
```

C++

```
virtual __fastcall InternalFirst();
```

Description

This is InternalFirst, a member of class TIBCustomDataSet.

19.1.2.4.46 TIBCustomDataSet.InternalGotoBookmark Method

Pascal

```
procedure InternalGotoBookmark(Bookmark: Pointer); override;
```

C++

```
virtual __fastcall InternalGotoBookmark(void * Bookmark);
```

Description

This is InternalGotoBookmark, a member of class TIBCustomDataSet.

19.1.2.4.47 TIBCustomDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TIBCustomDataSet.

19.1.2.4.48 TIBCustomDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TIBCustomDataSet.

19.1.2.4.49 TIBCustomDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TIBCustomDataSet.

19.1.2.4.50 TIBCustomDataSet.InternalInsert Method

Pascal

```
procedure InternalInsert; override;
```

C++

```
virtual __fastcall InternalInsert();
```

Description

This is InternalInsert, a member of class TIBCustomDataSet.

19.1.2.4.51 TIBCustomDataSet.InternalLast Method

Pascal

```
procedure InternalLast; override;
```

C++

```
virtual __fastcall InternalLast();
```

Description

This is InternalLast, a member of class TIBCustomDataSet.

19.1.2.4.52 TIBCustomDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TIBCustomDataSet.

19.1.2.4.53 TIBCustomDataSet.InternalPost Method

Pascal

```
procedure InternalPost; override;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TIBCustomDataSet.

19.1.2.4.54 TIBCustomDataSet.InternalPrepare Method

Pascal

```
procedure InternalPrepare; virtual;
```

C++

```
virtual __fastcall InternalPrepare();
```

Description

This is InternalPrepare, a member of class TIBCustomDataSet.

19.1.2.4.55 TIBCustomDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TIBCustomDataSet.

19.1.2.4.56 TIBCustomDataSet.InternalRefreshRow Method

Pascal

```
procedure InternalRefreshRow; virtual;
```

C++

```
virtual __fastcall InternalRefreshRow();
```

Description

This is InternalRefreshRow, a member of class TIBCustomDataSet.

19.1.2.4.57 TIBCustomDataSet.InternalSetParamsFromCursor Method

Pascal

```
procedure InternalSetParamsFromCursor; virtual;
```

C++

```
virtual __fastcall InternalSetParamsFromCursor();
```

Description

This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.

19.1.2.4.58 TIBCustomDataSet.InternalSetToRecord Method

Pascal

```
procedure InternalSetToRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalSetToRecord(const char * Buffer);
```

Description

This is InternalSetToRecord, a member of class TIBCustomDataSet.

19.1.2.4.59 TIBCustomDataSet.InternalUnPrepare Method

Pascal

```
procedure InternalUnPrepare; virtual;
```

C++

```
virtual __fastcall InternalUnPrepare();
```

Description

This is InternalUnPrepare, a member of class TIBCustomDataSet.

19.1.2.4.60 TIBCustomDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TIBCustomDataSet.

19.1.2.4.61 TIBCustomDataSet.IsSequenced Method

IsSequenced indicates whether sequence numbers are available for database records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

IsSequenced indicates whether sequence numbers are available for database records.

Delphi syntax:

```
function IsSequenced: Boolean; override;
```

C++ syntax:

```
virtual bool __fastcall IsSequenced(void);
```

Description

Call IsSequenced to determine whether database records can be located by sequence numbers. When IsSequenced is true, the dataset can navigate directly to a specific record by setting the RecNo property. If IsSequenced is false, the only way to navigate to a specific record is to start at the beginning and count records.

19.1.2.4.62 TIBCustomDataSet.Loaded Method

Pascal

```
procedure Loaded; override;
```

C++

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

Description

This is Loaded, a member of class TIBCustomDataSet.

19.1.2.4.63 TIBCustomDataSet.Locate Method

Searches the dataset for a specified record and makes that record the current record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call Locate to search a dataset for a specific record and position the cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. For example:

```
        with CustTable do
    Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']),
[loPartialKey]);
       TLocateOptions Opts;
    Opts.Clear();
    Opts << loPartialKey;
```

```
Variant locvalues[3];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
locvalues[2] = Variant("408-431-1000");
CustTable->Locate("Company;Contact;Phone", VarArrayOf(locvalues, 2), Opts);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns true if it finds a matching record, and makes that record the current one. Otherwise Locate returns false.

Locate uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, Locate uses the index. Otherwise Locate creates a filter for the search.

19.1.2.4.64 TIBCustomDataSet.LocateNext Method

Searches the dataset for the record after a specified record and makes that record the current record.

Pascal

```
function LocateNext(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions): Boolean;
```

C++

```
__fastcall Boolean LocateNext(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call LocateNext to search a dataset for a record after the current cursor position

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine.

The Options parameter is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then LocateNext ignores case when matching fields. If Options contains the loPartialKey setting, then LocateNext allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

LocateNext returns true if it finds a matching record, and makes that record the current one. Otherwise LocateNext returns false.

LocateNext uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, LocateNext uses the index. Otherwise LocateNext creates a filter for the search.

19.1.2.4.65 TIBCustomDataSet.Lookup Method

Retrieves field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const UnicodeString KeyFields, const Variant KeyValues,
```

```
const UnicodeString ResultFields);
```

Description

Call Lookup to retrieve values for specified fields from a record that matches search criteria. KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant array containing the values to match in the key fields. To specify multiple search values, pass KeyValues as a variant array as an argument, or construct a variant array on the fly using the VarArrayOf routine.

ResultFields is a string containing a semicolon-delimited list of field names whose values should be returned from the matching record.

Lookup returns a variant array containing the values from the fields specified in ResultFields.

If the search fields in KeyFields are indexed, Lookup uses the index. Otherwise Lookup creates a filter for the search.

19.1.2.4.66 TIBCustomDataSet.Post Method

Writes a modified record to the database server.

Pascal

```
procedure Post; override;
```

C++

```
virtual __fastcall Post();
```

Description

Post writes any changes to the current record to the database server. Dataset methods that change the dataset state, such as Edit, Insert, or Append, or that move from one record to another, such as First, Last, Next, and Prior automatically call Post.

Before posting the current record, Post checks the GeneratorField (see page 3640) property to determine whether it must generate any field values before posting the record. After changing the generated field, if necessary, Post triggers a BeforePost (see page 3637) event, writes the modified record to the server, and then triggers an AfterPost (see page 3635) event.

19.1.2.4.67 TIBCustomDataSet.PSEndTransaction Method

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

This is PSEndTransaction, a member of class TIBCustomDataSet.

19.1.2.4.68 TIBCustomDataSet.PSExecuteStatement Method

Pascal

```
function PSExecuteStatement(const ASQL: string; AParams: TParams; ResultSet: Pointer = nil):
Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const UnicodeString ASQL, TParams AParams, void *
ResultSet = nil);
```

Description

This is PSExecuteStatement, a member of class TIBCustomDataSet.

19.1.2.4.69 TIBCustomDataSet.PSGetQuoteChar Method

Pascal

```
function PSGetQuoteChar: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetQuoteChar();
```

Description

This is PSGetQuoteChar, a member of class TIBCustomDataSet.

19.1.2.4.70 TIBCustomDataSet.PSGetTableName Method

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetTableName();
```

Description

This is PSGetTableName, a member of class TIBCustomDataSet.

19.1.2.4.71 TIBCustomDataSet.PSGetUpdateException Method

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

This is PSGetUpdateException, a member of class TIBCustomDataSet.

19.1.2.4.72 TIBCustomDataSet.PSInTransaction Method

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

This is PSInTransaction, a member of class TIBCustomDataSet.

19.1.2.4.73 TIBCustomDataSet.PSIsSQLBased Method

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

This is PSIsSQLBased, a member of class TIBCustomDataSet.

19.1.2.4.74 TIBCustomDataSet.PSIsSQLSupported Method

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

This is PSIsSQLSupported, a member of class TIBCustomDataSet.

19.1.2.4.75 TIBCustomDataSet.PSReset Method

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

This is PSReset, a member of class TIBCustomDataSet.

19.1.2.4.76 TIBCustomDataSet.PSStartTransaction Method

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

This is PSStartTransaction, a member of class TIBCustomDataSet.

19.1.2.4.77 TIBCustomDataSet.PSUpdateRecord Method

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

This is PSUpdateRecord, a member of class TIBCustomDataSet.

19.1.2.4.78 TIBCustomDataSet.RecordModified Method

Sets the record to modified or unmodified.

Pascal

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

C++

```
__fastcall RecordModified(Boolean Value);
```

Description

Call RecordModified to mark a record as modified or unmodified.

Note: This method is for internal use.

19.1.2.4.79 TIBCustomDataSet.ReQuery Method

Pascal

```
procedure ReQuery;
```

C++

```
__fastcall ReQuery();
```

Description

This is ReQuery, a member of class TIBCustomDataSet.

19.1.2.4.80 TIBCustomDataSet.RevertRecord Method

Restores the current record in the dataset to an unmodified state when cached updates are enabled.

Pascal

```
procedure RevertRecord;
```

C++

```
__fastcall RevertRecord();
```

Description

Call RevertRecord to undo changes made to the current record when cached updates are enabled.

Note: To undo all changes to all pending updates in the cache, call CancelUpdates (see page 3610).

19.1.2.4.81 TIBCustomDataSet.SetActive Method

Pascal

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

C++

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

Description

This is SetActive, a member of class TIBCustomDataSet.

19.1.2.4.82 TIBCustomDataSet.SetBookmarkData Method

Pascal

```
procedure SetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall SetBookmarkData(const char * Buffer, void * Data);
```

Description

This is SetBookmarkData, a member of class TIBCustomDataSet.

19.1.2.4.83 TIBCustomDataSet.SetBookmarkFlag Method

Pascal

```
procedure SetBookmarkFlag(Buffer: PChar; Value: TBookmarkFlag); override;
```

C++

```
virtual __fastcall SetBookmarkFlag(const char * Buffer, TBookmarkFlag Value);
```

Description

This is SetBookmarkFlag, a member of class TIBCustomDataSet.

19.1.2.4.84 TIBCustomDataSet.SetCachedUpdates Method

Pascal

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

C++

```
__fastcall SetCachedUpdates(Boolean Value);
```

Description

This is SetCachedUpdates, a member of class TIBCustomDataSet.

19.1.2.4.85 TIBCustomDataSet.SetDataSource Method

Pascal

```
procedure SetDataSource(Value: TDataSource);
```

C++

```
__fastcall SetDataSource(TDataSource Value);
```

Description

This is SetDataSource, a member of class TIBCustomDataSet.

19.1.2.4.86 SetFieldData Method

This is the overview for the SetFieldData method overload.

Overload List

	Name	Description
	TIBCustomDataSet.SetFieldData (TField, Pointer) (see page 3631)	This is SetFieldData, a member of class TIBCustomDataSet.
	TIBCustomDataSet.SetFieldData (TField, Pointer, Boolean) (see page 3631)	This is SetFieldData, a member of class TIBCustomDataSet.

19.1.2.4.86.1 TIBCustomDataSet.SetFieldData Method (TField, Pointer)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TIBCustomDataSet.

19.1.2.4.86.2 TIBCustomDataSet.SetFieldData Method (TField, Pointer, Boolean)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean); override;
overload;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
```

Description

This is SetFieldData, a member of class TIBCustomDataSet.

19.1.2.4.87 TIBCustomDataSet.SetRecNo Method

Pascal

```
procedure SetRecNo(Value: Integer); override;
```

C++

```
virtual __fastcall SetRecNo(int Value);
```

Description

This is SetRecNo, a member of class TIBCustomDataSet.

19.1.2.4.88 TIBCustomDataSet.SQLType Method

Returns the type of query to be executed.

Pascal

```
function SQLType: TIBSQLTypes;
```

C++

```
__fastcall TIBSQLTypes SQLType();
```

Description

Use the SQLType to determine the type of query to be executed. This is the same as the SQLType property of the query specified by QSelect (see page 3644).

19.1.2.4.89 TIBCustomDataSet.Undelete Method

Restores a record deleted from the dataset.

Pascal

```
procedure Undelete;
```

C++

```
__fastcall Undelete();
```

Description

Call Undelete to restore a record deleted or uninserted from the dataset.

19.1.2.4.90 TIBCustomDataSet.UpdateStatus Method

Reports the update status for the current record.

Pascal

```
function UpdateStatus: TUpdateStatus; override;
```

C++

```
virtual __fastcall TUpdateStatus UpdateStatus();
```

Description

Call UpdateStatus to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

19.1.2.5 TIBCustomDataSet Properties

The properties of the TIBCustomDataSet class are listed here.

Protected Properties

	Name	Description
	BufferChunks (see page 3638)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3638)	Specifies whether cached updates are enabled for a dataset.
	DeleteSQL (see page 3639)	Holds the SQL statement used to delete rows from the dataset.
	GeneratorField (see page 3640)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3640)	Holds the SQL statement used to insert rows into the dataset.
R	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
R	LiveMode (see page 3641)	Indicates what types of edits, if any, can be made to the data in the dataset.
	ModifySQL (see page 3641)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	ParamCheck (see page 3642)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
R	Params (see page 3643)	Specifies the parameters to use for any parameterized queries.
R	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.

	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	RefreshSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
	SelectStmtHandle (see page 3645)	Gets the handle for the SELECT statement.
	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
	StatementType (see page 3646)	Returns the statement type of the QSelect (see page 3644) query.
	UniDirectional (see page 3647)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Properties

	Name	Description
	DBHandle (see page 3639)	Specifies the database handle for the dataset.
	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
	RowsAffected (see page 3645)	Indicates the number of rows operated on by the latest query execution.
	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateObject (see page 3647)	Specifies the update object component used to update a read-only result set.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Properties

	Name	Description
	AfterCancel (see page 3634)	This is AfterCancel, a member of class TIBCustomDataSet.
	AfterClose (see page 3634)	This is AfterClose, a member of class TIBCustomDataSet.
	AfterDelete (see page 3634)	This is AfterDelete, a member of class TIBCustomDataSet.
	AfterEdit (see page 3634)	This is AfterEdit, a member of class TIBCustomDataSet.
	AfterInsert (see page 3635)	This is AfterInsert, a member of class TIBCustomDataSet.
	AfterOpen (see page 3635)	This is AfterOpen, a member of class TIBCustomDataSet.
	AfterPost (see page 3635)	This is AfterPost, a member of class TIBCustomDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3635)	This is AfterScroll, a member of class TIBCustomDataSet.
	AutoCalcFields (see page 3636)	This is AutoCalcFields, a member of class TIBCustomDataSet.
	BeforeCancel (see page 3636)	This is BeforeCancel, a member of class TIBCustomDataSet.
	BeforeClose (see page 3636)	This is BeforeClose, a member of class TIBCustomDataSet.
	BeforeDelete (see page 3636)	This is BeforeDelete, a member of class TIBCustomDataSet.
	BeforeEdit (see page 3637)	This is BeforeEdit, a member of class TIBCustomDataSet.
	BeforeInsert (see page 3637)	This is BeforeInsert, a member of class TIBCustomDataSet.
	BeforeOpen (see page 3637)	This is BeforeOpen, a member of class TIBCustomDataSet.
	BeforePost (see page 3637)	This is BeforePost, a member of class TIBCustomDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3638)	This is BeforeScroll, a member of class TIBCustomDataSet.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3641)	This is OnCalcFields, a member of class TIBCustomDataSet.
	OnDeleteError (see page 3642)	This is OnDeleteError, a member of class TIBCustomDataSet.

	OnEditError (see page 3642)	This is OnEditError, a member of class TIBCustomDataSet.
	OnNewRecord (see page 3642)	This is OnNewRecord, a member of class TIBCustomDataSet.
	OnPostError (see page 3642)	This is OnPostError, a member of class TIBCustomDataSet.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.

19.1.2.5.1 TIBCustomDataSet.AfterCancel Property

Pascal

```
property AfterCancel;
```

C++

```
__property AfterCancel;
```

Description

This is AfterCancel, a member of class TIBCustomDataSet.

19.1.2.5.2 TIBCustomDataSet.AfterClose Property

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

Description

This is AfterClose, a member of class TIBCustomDataSet.

19.1.2.5.3 TIBCustomDataSet.AfterDelete Property

Pascal

```
property AfterDelete;
```

C++

```
__property AfterDelete;
```

Description

This is AfterDelete, a member of class TIBCustomDataSet.

19.1.2.5.4 TIBCustomDataSet.AfterEdit Property

Pascal

```
property AfterEdit;
```

C++

```
__property AfterEdit;
```

Description

This is AfterEdit, a member of class TIBCustomDataSet.

19.1.2.5.5 TIBCustomDataSet.AfterInsert Property

Pascal

```
property AfterInsert;
```

C++

```
__property AfterInsert;
```

Description

This is AfterInsert, a member of class TIBCustomDataSet.

19.1.2.5.6 TIBCustomDataSet.AfterOpen Property

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

Description

This is AfterOpen, a member of class TIBCustomDataSet.

19.1.2.5.7 TIBCustomDataSet.AfterPost Property

Pascal

```
property AfterPost;
```

C++

```
__property AfterPost;
```

Description

This is AfterPost, a member of class TIBCustomDataSet.

19.1.2.5.8 TIBCustomDataSet.AfterRefresh Property

Pascal

```
property AfterRefresh;
```

C++

```
__property AfterRefresh;
```

Description

This is AfterRefresh, a member of class TIBCustomDataSet.

19.1.2.5.9 TIBCustomDataSet.AfterScroll Property

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

Description

This is AfterScroll, a member of class TIBCustomDataSet.

19.1.2.5.10 TIBCustomDataSet.AutoCalcFields Property

Pascal

```
property AutoCalcFields;
```

C++

```
__property AutoCalcFields;
```

Description

This is AutoCalcFields, a member of class TIBCustomDataSet.

19.1.2.5.11 TIBCustomDataSet.BeforeCancel Property

Pascal

```
property BeforeCancel;
```

C++

```
__property BeforeCancel;
```

Description

This is BeforeCancel, a member of class TIBCustomDataSet.

19.1.2.5.12 TIBCustomDataSet.BeforeClose Property

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

Description

This is BeforeClose, a member of class TIBCustomDataSet.

19.1.2.5.13 TIBCustomDataSet.BeforeDelete Property

Pascal

```
property BeforeDelete;
```

C++

```
__property BeforeDelete;
```

Description

This is BeforeDelete, a member of class TIBCustomDataSet.

19.1.2.5.14 TIBCustomDataSet.BeforeEdit Property

Pascal

```
property BeforeEdit;
```

C++

```
__property BeforeEdit;
```

Description

This is BeforeEdit, a member of class TIBCustomDataSet.

19.1.2.5.15 TIBCustomDataSet.BeforeInsert Property

Pascal

```
property BeforeInsert;
```

C++

```
__property BeforeInsert;
```

Description

This is BeforeInsert, a member of class TIBCustomDataSet.

19.1.2.5.16 TIBCustomDataSet.BeforeOpen Property

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

Description

This is BeforeOpen, a member of class TIBCustomDataSet.

19.1.2.5.17 TIBCustomDataSet.BeforePost Property

Pascal

```
property BeforePost;
```

C++

```
__property BeforePost;
```

Description

This is BeforePost, a member of class TIBCustomDataSet.

19.1.2.5.18 TIBCustomDataSet.BeforeRefresh Property

Pascal

```
property BeforeRefresh;
```

C++

```
__property BeforeRefresh;
```

Description

This is BeforeRefresh, a member of class TIBCustomDataSet.

19.1.2.5.19 TIBCustomDataSet.BeforeScroll Property

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

Description

This is BeforeScroll, a member of class TIBCustomDataSet.

19.1.2.5.20 TIBCustomDataSet.BufferChunks Property

Specifies the dataset buffer chunk size.

Pascal

```
property BufferChunks: Integer;
```

C++

```
__property int BufferChunks;
```

Description

Use BufferChunks to get or set the number of records for which the dataset allocates buffer space at any time. When the dataset's buffer is full, trying to fetch an additional record causes the dataset to reallocate the buffer so that it has enough memory to hold an additional BufferChunks records.

Note: When Unidirectional is true, BufferChunks has no meaning.

19.1.2.5.21 TIBCustomDataSet.CachedUpdates Property

Specifies whether cached updates are enabled for a dataset.

Pascal

```
property CachedUpdates: Boolean;
```

C++

```
__property Boolean CachedUpdates;
```

Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is true, cached updates are enabled. If CachedUpdates is false, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Note: **Note**: Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset.

19.1.2.5.22 TIBCustomDataSet.Database Property

Identifies the database component for which this dataset represents one or more tables.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to access the properties, events, and methods of the database component associated with this dataset.

See Also

Creating and Deleting Tables

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

19.1.2.5.23 TIBCustomDataSet.DBHandle Property

Specifies the database handle for the dataset.

Pascal

```
property DBHandle: PISC_DB_HANDLE;
```

C++

```
__property PISC_DB_HANDLE DBHandle;
```

Description

Check DBHandle to determine the database handle for the dataset.

19.1.2.5.24 TIBCustomDataSet.DeleteSQL Property

Holds the SQL statement used to delete rows from the dataset.

Pascal

```
property DeleteSQL: TStrings;
```

C++

```
__property TStrings DeleteSQL;
```

Description

Use DeleteSQL to delete rows in the dataset.

19.1.2.5.25 TIBCustomDataSet.ForcedRefresh Property

Specifies whether the dataset should refresh its data every time it posts a record.

Pascal

```
property ForcedRefresh: Boolean;
```

C++

```
__property Boolean ForcedRefresh;
```

Description

Set ForcedRefresh to true to force the dataset to refresh its data every time it posts a record. When ForcedRefresh is false (the default), the data is only refreshed when you call the Refresh method, or when the dataset needs to fetch the value of a default or internally calculated field.

19.1.2.5.26 TIBCustomDataSet.GeneratorField Property

Specifies whether the dataset should refresh its data every time it posts a record.

Pascal

```
property GeneratorField: TIBGeneratorField;
```

C++

```
__property TIBGeneratorField GeneratorField;
```

Description

Use GeneratorField to specify a generator that automatically provides the value for one of the fields in the dataset. Use the properties of the TIBGeneratorField (see page 3682) object to indicate which field's value should be set and how that value should be calculated.

Warning: Do not set the value of GeneratorField. To change the value of this property, change the properties of the TIBGeneratorField (see page 3682) object that is the value of this property.

19.1.2.5.27 TIBCustomDataSet.InsertSQL Property

Holds the SQL statement used to insert rows into the dataset.

Pascal

```
property InsertSQL: TStrings;
```

C++

```
__property TStrings InsertSQL;
```

Description

Use InsertSQL to insert rows into the dataset.

19.1.2.5.28 TIBCustomDataSet.InternalPrepared Property

Indicates whether the dataset has prepared its queries for execution.

Pascal

```
property InternalPrepared: Boolean;
```

C++

```
__property Boolean InternalPrepared;
```

Description

TIBCustomDataSet (see page 3602) uses InternalPrepared internally to determine when it needs to prepare its set of queries

for execution. The dataset checks InternalPrepared before executing a query, and prepares the query if InternalPrepared is false.

19.1.2.5.29 TIBCustomDataSet.LiveMode Property

Indicates what types of edits, if any, can be made to the data in the dataset.

Pascal

```
property LiveMode: TLiveModes;
```

C++

```
__property TLiveModes LiveMode;
```

Description

Read LiveMode to determine what types of edits should be allowed. LiveMode is set automatically to restrict the capabilities of the dataset to reflect the user's rights.

Note: For read-only datasets, such as TIBQuery, LiveMode is always an empty set.

19.1.2.5.30 TIBCustomDataSet.ModifySQL Property

Provides the ability to access the SQL object encapsulating the ModifySQL statement.

Pascal

```
property ModifySQL: TStrings;
```

C++

```
__property TStrings ModifySQL;
```

Description

Use ModifySQL to access the SQL object that encapsulates the ModifySQL statement.

19.1.2.5.31 TIBCustomDataSet.ObjectView Property

Pascal

```
property ObjectView;
```

C++

```
__property ObjectView;
```

Description

This is ObjectView, a member of class TIBCustomDataSet.

19.1.2.5.32 TIBCustomDataSet.OnCalcFields Property

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

Description

This is OnCalcFields, a member of class TIBCustomDataSet.

19.1.2.5.33 TIBCustomDataSet.OnDeleteError Property

Pascal

```
property OnDeleteError;
```

C++

```
__property OnDeleteError;
```

Description

This is OnDeleteError, a member of class TIBCustomDataSet.

19.1.2.5.34 TIBCustomDataSet.OnEditError Property

Pascal

```
property OnEditError;
```

C++

```
__property OnEditError;
```

Description

This is OnEditError, a member of class TIBCustomDataSet.

19.1.2.5.35 TIBCustomDataSet.OnNewRecord Property

Pascal

```
property OnNewRecord;
```

C++

```
__property OnNewRecord;
```

Description

This is OnNewRecord, a member of class TIBCustomDataSet.

19.1.2.5.36 TIBCustomDataSet.OnPostError Property

Pascal

```
property OnPostError;
```

C++

```
__property OnPostError;
```

Description

This is OnPostError, a member of class TIBCustomDataSet.

19.1.2.5.37 TIBCustomDataSet.ParamCheck Property

Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether the Params (see page 3643) property is cleared and regenerated if an application modifies the SQL in this dataset's query. By default ParamCheck is true, meaning that the Params (see page 3643) property is automatically regenerated at runtime. When ParamCheck is true, the proper number of parameters is guaranteed to be generated for the current SQL statement.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement that are not parameters for the TIBCustomDataSet (see page 3602). For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TIBCustomDataSet (see page 3602) executing the DDL statement.

19.1.2.5.38 TIBCustomDataSet.Params Property

Specifies the parameters to use for any parameterized queries.

Pascal

```
property Params: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA Params;
```

Description

Use Params to set up the parameters for the SQL object specified by the QSelect (see page 3644) property. Params is the same as the SQLParams (see page 3646) property.

19.1.2.5.39 TIBCustomDataSet.Plan Property

Indicates the PLAN portion of the dataset's SELECT query.

Pascal

```
property Plan: String;
```

C++

```
__property UnicodeString Plan;
```

Description

Read the Plan property to determine the PLAN clause in the query specified by QSelect (see page 3644).

19.1.2.5.40 TIBCustomDataSet.QDelete Property

Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.

Pascal

```
property QDelete: TIBSQL;
```

C++

```
__property TIBSQL QDelete;
```

Description

Use QDelete to access the SQL object that encapsulates the Delete SQL statement.

19.1.2.5.41 TIBCustomDataSet.QInsert Property

Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.

Pascal

```
property QInsert: TIBSQL;
```

C++

```
__property TIBSQL QInsert;
```

Description

Use QInsert to access the SQL object that encapsulates the InsertSQL (see page 3640) statement.

19.1.2.5.42 TIBCustomDataSet.QModify Property

Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.

Pascal

```
property QModify: TIBSQL;
```

C++

```
__property TIBSQL QModify;
```

Description

Use QModify to access the SQL object that encapsulates the ModifySQL (see page 3641) statement.

19.1.2.5.43 TIBCustomDataSet.QRefresh Property

Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.

Pascal

```
property QRefresh: TIBSQL;
```

C++

```
__property TIBSQL QRefresh;
```

Description

Use QRefresh to access the SQL object that encapsulates the RefreshSQL (see page 3645) statement.

19.1.2.5.44 TIBCustomDataSet.QSelect Property

Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.

Pascal

```
property QSelect: TIBSQL;
```

C++

```
__property TIBSQL QSelect;
```

Description

Use QSelect to access the SQL object that encapsulates the SelectSQL (see page 3645) statement.

19.1.2.5.45 TIBCustomDataSet.RefreshSQL Property

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

Pascal

```
property RefreshSQL: TStrings;
```

C++

```
__property TStrings RefreshSQL;
```

Description

Use RefreshSQL to access the SQL object that encapsulates the RefreshSQL statement.

19.1.2.5.46 TIBCustomDataSet.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the dataset represents a query that does not return a result set, you can't use the RecordCount property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is -1 if the query could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

19.1.2.5.47 TIBCustomDataSet.SelectSQL Property

Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.

Pascal

```
property SelectSQL: TStrings;
```

C++

```
__property TStrings SelectSQL;
```

Description

Use SelectSQL to access the SQL object that encapsulates the SelectSQL statement.

19.1.2.5.48 TIBCustomDataSet.SelectStmtHandle Property

Gets the handle for the SELECT statement.

Pascal

```
property SelectStmtHandle: TISC_STMT_HANDLE;
```

C++

```
__property TISC_STMT_HANDLE SelectStmtHandle;
```

Description

Use SelectStmtHandle to get the handle for the SELECT statement.

19.1.2.5.49 TIBCustomDataSet.SQLParams Property

Specifies the parameters to use for any parameterized queries.

Pascal

```
property SQLParams: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA SQLParams;
```

Description

Use SQLParams to set up the parameters for the SQL object specified by the QSelect (see page 3644) property. SQLParams is the same as the Params (see page 3643) property.

19.1.2.5.50 TIBCustomDataSet.StatementType Property

Returns the statement type of the QSelect (see page 3644) query.

Pascal

```
property StatementType: TIBSQLTypes;
```

C++

```
__property TIBSQLTypes StatementType;
```

Description

Use StatementType to determine the statement type of a QSelect (see page 3644) query.

19.1.2.5.51 TIBCustomDataSet.Transaction Property

Identifies the transaction under which the query executes.

Pascal

```
property Transaction: TIBTransaction;
```

C++

```
__property TIBTransaction Transaction;
```

Description

Use Transaction to determine the transaction that contains any query this dataset executes. Set the properties of this transaction object to control how the transaction executes, or set the Transaction property to cause this dataset's queries to be included in an existing transaction.

19.1.2.5.52 TIBCustomDataSet.TRHandle Property

Specifies the transaction handle for the dataset.

Pascal

```
property TRHandle: PISC_TR_HANDLE;
```

C++

```
__property PISC_TR_HANDLE TRHandle;
```

Description

Check TRHandle to determine the transaction handle for the dataset.

19.1.2.5.53 TIBCustomDataSet.UniDirectional Property

Determines whether bidirectional cursors are enabled for the dataset.

Pascal

```
property UniDirectional: Boolean;
```

C++

```
__property Boolean UniDirectional;
```

Description

Use UniDirectional to determine whether bidirectional cursors are enabled for this dataset.

Note: Do not confuse the UniDirectional property with the IsUniDirectional property inherited from TDataSet. IsUniDirectional indicates whether the dataset is a dbExpress dataset, which, in addition to using a unidirectional cursor, does not buffer records. UniDirectional is only concerned with whether the cursor permits backward or random navigation.

19.1.2.5.54 TIBCustomDataSet.UpdateMode Property

Indicates how records are located when updating records.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

UpdateMode is not currently implemented in IB (see page 3541) datasets. It is reserved for future use.

19.1.2.5.55 TIBCustomDataSet.UpdateObject Property

Specifies the update object component used to update a read-only result set.

Pascal

```
property UpdateObject: TIBDataSetUpdateObject;
```

C++

```
__property TIBDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TIBDataSetUpdateObject (see page 3674) component to use in an application that must be able to update a read-only result set.

In some cases, such as when using TIBQuery, the dataset does not automatically update records. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

TIBDataSetUpdateObject (see page 3674) is especially useful for complicated updates from an OnUpdateRecord (see page 3651) event.

19.1.2.5.56 TIBCustomDataSet.UpdateRecordTypes Property

Specifies the type of records visible in a dataset when cached updates are enabled.

Pascal

```
property UpdateRecordTypes: TIBUpdateRecordTypes;
```

C++

```
__property TIBUpdateRecordTypes UpdateRecordTypes;
```

Description

Use UpdateRecordTypes to specify the records that are visible in a dataset when cached updates are enabled. UpdateRecordTypes is a set that can have the following values:

By default, a dataset is created with an UpdateRecordTypes set of cusModified, cusInserted, or cusUnmodified, meaning that all existing, edited, or inserted records are visible to the user.

An application that must cycle through a dataset to undelete records may change UpdateRecordTypes as part of an undelete method, so that deleted records are 'visible' long enough to restore them to their previously undeleted conditions.

Similarly, an application that must cycle through a dataset to uninsert records may change UpdateRecordTypes as part of an uninsert method, so that uninserted records are 'visible' long enough to restore them to their previously inserted conditions.

An application might also use UpdateRecordTypes like a filter to temporarily limit visible records to those added or inserted by the user during the current session.

19.1.2.5.57 TIBCustomDataSet.UpdatesPending Property

Indicates whether the cached updates buffer contains records that are not yet applied.

Pascal

```
property UpdatesPending: Boolean;
```

C++

```
__property Boolean UpdatesPending;
```

Description

Examine UpdatesPending to check the status of the cached updates buffer. If UpdatesPending is true, then there are edited, deleted, or inserted records to apply to the database. If UpdatesPending is false, there are no records in the cache.

19.1.2.6 TIBCustomDataSet Events

The events of the TIBCustomDataSet class are listed here.

Protected Events

	Name	Description
	AfterDatabaseDisconnect (see page 3649)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3649)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3649)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3649)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 3650)	Occurs when the database associated with this dataset is freed.

	TransactionFree (see page 3651)	Occurs when the transaction that includes this dataset's query is freed.

Published Events

	Name	Description
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.

19.1.2.6.1 TIBCustomDataSet.AfterDatabaseDisconnect Event

Occurs immediately after the connection to the associated database is closed.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to respond when the dataset closes its connection to the database specified by the Database (see page 3639) property.

19.1.2.6.2 TIBCustomDataSet.AfterTransactionEnd Event

Occurs immediately after a transaction that includes this dataset's query ends.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to respond when the transaction specified by the Transaction (see page 3646) property ends.

19.1.2.6.3 TIBCustomDataSet.BeforeDatabaseDisconnect Event

Occurs immediately before the connection to the associated database is closed.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to respond when the dataset is about to close its connection to the database specified by the Database (see page 3639) property.

19.1.2.6.4 TIBCustomDataSet.BeforeTransactionEnd Event

Occurs immediately before a transaction that includes this dataset's query ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to respond when the transaction specified by the Transaction (see page 3646) property is about to end.

19.1.2.6.5 TIBCustomDataSet.DatabaseFree Event

Occurs when the database associated with this dataset is freed.

Pascal

```
property DatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent DatabaseFree;
```

Description

Write a DatabaseFree event handler to respond when the database specified by the Database (see page 3639) property is freed.

19.1.2.6.6 TIBCustomDataSet.OnUpdateError Event

Occurs if an exception is generated when cached updates are applied to a database.

Pascal

```
property OnUpdateError: TIBUpdateErrorEvent;
```

C++

```
__property TIBUpdateErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

Because there is a delay between the time a record is first cached and the time cached updates are applied, there is a possibility that another application may change one or more of the same records in the database before the cached changes can be applied. DataSet is the name of the dataset to which updates are applied.

The error handler can use the OldValue and NewValue properties of TField to evaluate error conditions, and set NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

Note: If a call to ApplyUpdates (see page 3609) raises an exception and ApplyUpdates (see page 3609) is not called within the context of a try..except (Delphi) or try..catch (C++) block, an error message is returned. If an OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is returned twice. To prevent re-return, set UpdateAction to uaAbort in the error handler.

Warning: **Important:** The code in an OnUpdateError handler must not call any methods that make a different record the current one.

19.1.2.6.7 TIBCustomDataSet.OnUpdateRecord Event

Occurs when cached updates are applied to a record.

Pascal

```
property OnUpdateRecord: TIBUpdateRecordEvent;
```

C++

```
__property TIBUpdateRecordEvent OnUpdateRecord;
```

Description

Write an OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

Note: The code in an OnUpdateRecord handler must not call any methods that make a different record the current one.

19.1.2.6.8 TIBCustomDataSet.TransactionFree Event

Occurs when the transaction that includes this dataset's query is freed.

Pascal

```
property TransactionFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent TransactionFree;
```

Description

Write a TransactionFree event handler to respond when the transaction specified by the Transaction (see page 3646) property is freed.

19.1.3 TIBDataLink Class

TIBDataLink is a helper class used by data-aware objects to coordinate the actions of TIBDataSet (see page 3654) and to respond to data events.

Class Hierarchy

Pascal

```
TIBDataLink = class(TDetailDataLink);
```

C++

```
class TIBDataLink : public TDetailDataLink;
```

File

IBCustomDataSet

Description

Use TIBDataLink or one of its descendants in any data-aware object that implements a DataSource property to represent its link to a dataset or that needs to respond to data events.

The constructor of the data-aware object should call the constructor of TIBDataLink, and initialize any relevant properties. The data-aware object can then link to a TDataSource by using the DataSource property of the TIBDataLink.

Data-aware objects that link to a single field in a dataset should use a TFieldDataLink instead.

19.1.3.1 TIBDataLink Members

The following tables list the members exposed by TIBDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 3653)	This is ActiveChanged, a member of class TIBDataLink.
	CheckBrowseMode (see page 3653)	This is CheckBrowseMode, a member of class TIBDataLink.
	GetDetailDataSet (see page 3653)	This is GetDetailDataSet, a member of class TIBDataLink.
	RecordChanged (see page 3654)	This is RecordChanged, a member of class TIBDataLink.

Public Methods

	Name	Description
	Create (see page 3652)	Creates an instance of TIBDataLink (see page 3651).
	Destroy (see page 3652)	Destroys an instance of TIBDataLink (see page 3651).

19.1.3.2 TIBDataLink.Create Constructor

Creates an instance of TIBDataLink (see page 3651).

Pascal

```
constructor Create(ADataSet: TIBCustomDataSet);
```

C++

```
__fastcall TIBDataLink(TIBCustomDataSet ADataSet);
```

Description

Create is called from the constructor of any data-aware object that uses a TIBDataLink (see page 3651) to implement its DataSource property.

After calling the inherited constructor, Create initializes the BufferCount property to 1. Data-aware objects that use a TIBDataLink (see page 3651) object to manage their link to a DataSource should change the BufferCount property to the number or records they represent, after calling the inherited constructor.

19.1.3.3 TIBDataLink.Destroy Destructor

Destroys an instance of TIBDataLink (see page 3651).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBDataLink();
```

Description

Do not call Destroy directly in an application. Instead, call the Free method. Free verifies that the TIBDataLink (see page 3651) object is not already freed and only then calls Destroy.

Before calling the inherited destructor, Destroy removes any reference to the TIBDataLink (see page 3651) from the data source object.

The TIBDataLink (see page 3651) object should be destroyed in the destructor of its Owner, where that Owner calls Create (see page 3652) from its constructor.

19.1.3.4 TIBDataLink Methods

The methods of the TIBDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 3653)	This is ActiveChanged, a member of class TIBDataLink.
	CheckBrowseMode (see page 3653)	This is CheckBrowseMode, a member of class TIBDataLink.
	GetDetailDataSet (see page 3653)	This is GetDetailDataSet, a member of class TIBDataLink.
	RecordChanged (see page 3654)	This is RecordChanged, a member of class TIBDataLink.

19.1.3.4.1 TIBDataLink.ActiveChanged Method

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

This is ActiveChanged, a member of class TIBDataLink.

19.1.3.4.2 TIBDataLink.CheckBrowseMode Method

Pascal

```
procedure CheckBrowseMode; override;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

This is CheckBrowseMode, a member of class TIBDataLink.

19.1.3.4.3 TIBDataLink.GetDetailDataSet Method

Pascal

```
function GetDetailDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

This is GetDetailDataSet, a member of class TIBDataLink.

19.1.3.4.4 TIBDataLink.RecordChanged Method

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

This is RecordChanged, a member of class TIBDataLink.

19.1.4 TIBDataSet Class

TIBDataSet executes InterBase SQL statements.

Class Hierarchy

Pascal

```
TIBDataSet = class(TIBCustomDataSet);
```

C++

```
class TIBDataSet : public TIBCustomDataSet;
```

File

IBCustomDataSet

Description

Use TIBDataSet to execute InterBase SQL statements. Each TIBDataSet component has properties for a statement to fetch data (typically an SQL SELECT statement), a statement to modify those records, one for inserting records, one for deleting records, and one to refresh the dataset.

TIBDataSet buffers the result set, making it completely scrollable. Since TIBDataSet is a descendant of TDataSet, it works well with all data-aware components.

See Also

Types of Datasets

Getting Started with InterBase Express

19.1.4.1 TIBDataSet Members

The following tables list the members exposed by TIBDataSet.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.

	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.
V	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
V	Disconnect (see page 3613)	This is Disconnect, a member of class TIBCustomDataSet.
V	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
V	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
V	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
V	DoOnNewRecord (see page 3614)	This is DoOnNewRecord, a member of class TIBCustomDataSet.
V	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
V	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
V	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
V	GetCanModify (see page 3615)	This is GetCanModify, a member of class TIBCustomDataSet.
V	GetDataSource (see page 3616)	This is GetDataSource, a member of class TIBCustomDataSet.
V	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
V	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
V	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
V	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
V	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
V	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
V	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
V	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
V	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.
V	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
V	InternalClose (see page 3619)	This is InternalClose, a member of class TIBCustomDataSet.
V	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
V	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
V	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
V	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
V	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
V	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
V	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
V	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
V	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
V	InternalOpen (see page 3659)	This is InternalOpen, a member of class TIBDataSet.
V	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
V	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
V	InternalRefresh (see page 3622)	This is InternalRefresh, a member of class TIBCustomDataSet.
V	InternalRefreshRow (see page 3622)	This is InternalRefreshRow, a member of class TIBCustomDataSet.
V	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
V	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.
V	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.
V	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.
V	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
V	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
V	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
V	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
V	PSGetTableName (see page 3627)	This is PSGetTableName, a member of class TIBCustomDataSet.
V	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
V	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
V	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.

	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
	PSSetCommandText (see page 3660)	This is PSSetCommandText, a member of class TIBDataSet.
	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 3660)	This is SetFiltered, a member of class TIBDataSet.
	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.

Protected Properties

	Name	Description
	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.
	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Methods

	Name	Description
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.
	BatchInput (see page 3659)	Executes the parameterized query in SQL for input in the referenced input object.
	BatchOutput (see page 3659)	Outputs the selected query in SQL to the referenced OutputObject.
	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.
	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
	CompareBookmarks (see page 3611)	Compares two bookmarks.
	Create (see page 3606)	Creates an instance of a TDataSet component.
	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
	Destroy (see page 3606)	Destroys the instance of a dataset component.
	ExecSQL (see page 3659)	Executes the SQL statement that the SelectSQL (see page 3671) property specifies when it is not a SELECT statement.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.
	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
	Locate (see page 3624)	Searches the dataset for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.

	ParamByName (see page 3660)	Returns a specified parameter of the SELECT query that fetches this dataset's data.
V	Post (see page 3626)	Writes a modified record to the database server.
	Prepare (see page 3660)	Prepares all queries in the dataset to be executed.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
	UnPrepare (see page 3661)	Resets the state of a dataset's internal queries.
V	UpdateStatus (see page 3632)	Reports the update status for the current record.

Public Properties

	Name	Description
R	DBHandle (see page 3639)	Specifies the database handle for the dataset.
	LiveMode (see page 3668)	Indicates what types of edits, if any, can be made to the data in the dataset.
	Params (see page 3670)	Provides the ability to specify values for a parameterized query.
R	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
R	Prepared (see page 3671)	Determines whether or not a set of dataset queries is prepared for execution.
R	RowsAffected (see page 3645)	Indicates the number of rows operated on by the latest query execution.
	SelectStmtHandle (see page 3671)	Gets the handle for the SELECT statement.
	StatementType (see page 3672)	Returns the statement type of the QSelect query.
R	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 3673)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3673)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3673)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3674)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 3674)	Occurs when the database associated with this dataset is freed.
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.
	TransactionFree (see page 3674)	Occurs when the transaction that includes this dataset's query is freed.

Published Properties

	Name	Description
	Active (see page 3662)	This is Active, a member of class TIBDataSet.
	AfterCancel (see page 3662)	This is AfterCancel, a member of class TIBDataSet.
	AfterClose (see page 3662)	This is AfterClose, a member of class TIBDataSet.
	AfterDelete (see page 3663)	This is AfterDelete, a member of class TIBDataSet.
	AfterEdit (see page 3663)	This is AfterEdit, a member of class TIBDataSet.
	AfterInsert (see page 3663)	This is AfterInsert, a member of class TIBDataSet.
	AfterOpen (see page 3663)	This is AfterOpen, a member of class TIBDataSet.
	AfterPost (see page 3663)	This is AfterPost, a member of class TIBDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3664)	This is AfterScroll, a member of class TIBDataSet.
	AutoCalcFields (see page 3664)	This is AutoCalcFields, a member of class TIBDataSet.
	BeforeCancel (see page 3664)	This is BeforeCancel, a member of class TIBDataSet.
	BeforeClose (see page 3664)	This is BeforeClose, a member of class TIBDataSet.
	BeforeDelete (see page 3665)	This is BeforeDelete, a member of class TIBDataSet.

	BeforeEdit (see page 3665)	This is BeforeEdit, a member of class TIBDataSet.
	BeforeInsert (see page 3665)	This is BeforeInsert, a member of class TIBDataSet.
	BeforeOpen (see page 3665)	This is BeforeOpen, a member of class TIBDataSet.
	BeforePost (see page 3665)	This is BeforePost, a member of class TIBDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3666)	This is BeforeScroll, a member of class TIBDataSet.
	BufferChunks (see page 3666)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3666)	Specifies whether cached updates are enabled for a dataset.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	DataSource (see page 3667)	This is DataSource, a member of class TIBDataSet.
	DeleteSQL (see page 3667)	Holds the SQL statement used to delete rows from the dataset.
	Filtered (see page 3667)	Specifies whether or not filtering is active for the dataset.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	GeneratorField (see page 3667)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3668)	Holds the SQL statement used to insert rows into the dataset.
	ModifySQL (see page 3668)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3669)	This is OnCalcFields, a member of class TIBDataSet.
	OnDeleteError (see page 3669)	This is OnDeleteError, a member of class TIBDataSet.
	OnEditError (see page 3669)	This is OnEditError, a member of class TIBDataSet.
	OnFilterRecord (see page 3669)	This is OnFilterRecord, a member of class TIBDataSet.
	OnNewRecord (see page 3669)	This is OnNewRecord, a member of class TIBDataSet.
	OnPostError (see page 3670)	This is OnPostError, a member of class TIBDataSet.
	ParamCheck (see page 3670)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
	RefreshSQL (see page 3671)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3671)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.
	UniDirectional (see page 3672)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 3672)	Specifies the update object component used to update a read-only result set.

19.1.4.2 TIBDataSet Methods

The methods of the TIBDataSet class are listed here.

Protected Methods

	Name	Description
	InternalOpen (see page 3659)	This is InternalOpen, a member of class TIBDataSet.
	PSSetCommandText (see page 3660)	This is PSSetCommandText, a member of class TIBDataSet.
	SetFiltered (see page 3660)	This is SetFiltered, a member of class TIBDataSet.

Public Methods

	Name	Description
	BatchInput (see page 3659)	Executes the parameterized query in SQL for input in the referenced input object.
	BatchOutput (see page 3659)	Outputs the selected query in SQL to the referenced OutputObject.
	ExecSQL (see page 3659)	Executes the SQL statement that the SelectSQL (see page 3671) property specifies when it is not a SELECT statement.
	ParamByName (see page 3660)	Returns a specified parameter of the SELECT query that fetches this dataset's data.
	Prepare (see page 3660)	Prepares all queries in the dataset to be executed.
	UnPrepare (see page 3661)	Resets the state of a dataset's internal queries.

19.1.4.2.1 TIBDataSet.BatchInput Method

Executes the parameterized query in SQL for input in the referenced input object.

Pascal

```
procedure BatchInput(InputObject: TIBBatchInput);
```

C++

```
__fastcall BatchInput(TIBBatchInput InputObject);
```

Description

Call BatchInput to execute the parameterized query in SQL for input in the referenced input object.

19.1.4.2.2 TIBDataSet.BatchOutput Method

Outputs the selected query in SQL to the referenced OutputObject.

Pascal

```
procedure BatchOutput(OutputObject: TIBBatchOutput);
```

C++

```
__fastcall BatchOutput(TIBBatchOutput OutputObject);
```

Description

Call BatchOutput to output the selected query in SQL to the referenced OutputObject.

19.1.4.2.3 TIBDataSet.ExecSQL Method

Executes the SQL statement that the SelectSQL (see page 3671) property specifies when it is not a SELECT statement.

Pascal

```
procedure ExecSQL;
```

C++

```
__fastcall ExecSQL();
```

Description

Call ExecSQL to execute the main query for this dataset when it is not a SELECT statement. If the value of the SelectSQL (see page 3671) property is not set, or if it is set to a SELECT statement, ExecSQL raises an EIBError (see page 3543) exception.

19.1.4.2.4 TIBDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TIBDataSet.

19.1.4.2.5 TIBDataSet.ParamByName Method

Returns a specified parameter of the SELECT query that fetches this dataset's data.

Pascal

```
function ParamByName(Idx: String): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR ParamByName(UnicodeString Idx);
```

Description

Call ParamByName to retrieve the object that represents a parameter of the query that fetches this dataset's data. Use the TIBXSQLVAR (see page 4086) object this method returns to get or set the parameter value.

Idx is the name of the parameter.

19.1.4.2.6 TIBDataSet.Prepare Method

Prepares all queries in the dataset to be executed.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call Prepare to prepare all queries in the dataset to be executed.

19.1.4.2.7 TIBDataSet.PSSetCommandText Method

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const UnicodeString CommandText);
```

Description

This is PSSetCommandText, a member of class TIBDataSet.

19.1.4.2.8 TIBDataSet.SetFiltered Method

Pascal

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

C++

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

Description

This is SetFiltered, a member of class TIBDataSet.

19.1.4.2.9 TIBDataSet.UnPrepare Method

Resets the state of a dataset's internal queries.

Pascal

```
procedure UnPrepare;
```

C++

```
__fastcall UnPrepare();
```

Description

Note: Call UnPrepare to reset the state of a dataset's internal queries.

19.1.4.3 TIBDataSet Properties

The properties of the TIBDataSet class are listed here.

Public Properties

	Name	Description
	LiveMode (see page 3668)	Indicates what types of edits, if any, can be made to the data in the dataset.
	Params (see page 3670)	Provides the ability to specify values for a parameterized query.
R	Prepared (see page 3671)	Determines whether or not a set of dataset queries is prepared for execution.
	SelectStmtHandle (see page 3671)	Gets the handle for the SELECT statement.
	StatementType (see page 3672)	Returns the statement type of the QSelect query.

Published Properties

	Name	Description
	Active (see page 3662)	This is Active, a member of class TIBDataSet.
	AfterCancel (see page 3662)	This is AfterCancel, a member of class TIBDataSet.
	AfterClose (see page 3662)	This is AfterClose, a member of class TIBDataSet.
	AfterDelete (see page 3663)	This is AfterDelete, a member of class TIBDataSet.
	AfterEdit (see page 3663)	This is AfterEdit, a member of class TIBDataSet.
	AfterInsert (see page 3663)	This is AfterInsert, a member of class TIBDataSet.
	AfterOpen (see page 3663)	This is AfterOpen, a member of class TIBDataSet.
	AfterPost (see page 3663)	This is AfterPost, a member of class TIBDataSet.
	AfterScroll (see page 3664)	This is AfterScroll, a member of class TIBDataSet.
	AutoCalcFields (see page 3664)	This is AutoCalcFields, a member of class TIBDataSet.
	BeforeCancel (see page 3664)	This is BeforeCancel, a member of class TIBDataSet.
	BeforeClose (see page 3664)	This is BeforeClose, a member of class TIBDataSet.
	BeforeDelete (see page 3665)	This is BeforeDelete, a member of class TIBDataSet.
	BeforeEdit (see page 3665)	This is BeforeEdit, a member of class TIBDataSet.
	BeforeInsert (see page 3665)	This is BeforeInsert, a member of class TIBDataSet.
	BeforeOpen (see page 3665)	This is BeforeOpen, a member of class TIBDataSet.
	BeforePost (see page 3665)	This is BeforePost, a member of class TIBDataSet.
	BeforeScroll (see page 3666)	This is BeforeScroll, a member of class TIBDataSet.
	BufferChunks (see page 3666)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3666)	Specifies whether cached updates are enabled for a dataset.
	DataSource (see page 3667)	This is DataSource, a member of class TIBDataSet.
	DeleteSQL (see page 3667)	Holds the SQL statement used to delete rows from the dataset.
	Filtered (see page 3667)	Specifies whether or not filtering is active for the dataset.
	GeneratorField (see page 3667)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3668)	Holds the SQL statement used to insert rows into the dataset.

	ModifySQL (see page 3668)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	OnCalcFields (see page 3669)	This is OnCalcFields, a member of class TIBDataSet.
	OnDeleteError (see page 3669)	This is OnDeleteError, a member of class TIBDataSet.
	OnEditError (see page 3669)	This is OnEditError, a member of class TIBDataSet.
	OnFilterRecord (see page 3669)	This is OnFilterRecord, a member of class TIBDataSet.
	OnNewRecord (see page 3669)	This is OnNewRecord, a member of class TIBDataSet.
	OnPostError (see page 3670)	This is OnPostError, a member of class TIBDataSet.
	ParamCheck (see page 3670)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
	RefreshSQL (see page 3671)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3671)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
	UniDirectional (see page 3672)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 3672)	Specifies the update object component used to update a read-only result set.

19.1.4.3.1 TIBDataSet.Active Property

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

This is Active, a member of class TIBDataSet.

19.1.4.3.2 TIBDataSet.AfterCancel Property

Pascal

```
property AfterCancel;
```

C++

```
__property AfterCancel;
```

Description

This is AfterCancel, a member of class TIBDataSet.

19.1.4.3.3 TIBDataSet.AfterClose Property

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

Description

This is AfterClose, a member of class TIBDataSet.

19.1.4.3.4 TIBDataSet.AfterDelete Property

Pascal

```
property AfterDelete;
```

C++

```
__property AfterDelete;
```

Description

This is AfterDelete, a member of class TIBDataSet.

19.1.4.3.5 TIBDataSet.AfterEdit Property

Pascal

```
property AfterEdit;
```

C++

```
__property AfterEdit;
```

Description

This is AfterEdit, a member of class TIBDataSet.

19.1.4.3.6 TIBDataSet.AfterInsert Property

Pascal

```
property AfterInsert;
```

C++

```
__property AfterInsert;
```

Description

This is AfterInsert, a member of class TIBDataSet.

19.1.4.3.7 TIBDataSet.AfterOpen Property

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

Description

This is AfterOpen, a member of class TIBDataSet.

19.1.4.3.8 TIBDataSet.AfterPost Property

Pascal

```
property AfterPost;
```

C++

```
__property AfterPost;
```

Description

This is AfterPost, a member of class TIBDataSet.

19.1.4.3.9 TIBDataSet.AfterScroll Property

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

Description

This is AfterScroll, a member of class TIBDataSet.

19.1.4.3.10 TIBDataSet.AutoCalcFields Property

Pascal

```
property AutoCalcFields;
```

C++

```
__property AutoCalcFields;
```

Description

This is AutoCalcFields, a member of class TIBDataSet.

19.1.4.3.11 TIBDataSet.BeforeCancel Property

Pascal

```
property BeforeCancel;
```

C++

```
__property BeforeCancel;
```

Description

This is BeforeCancel, a member of class TIBDataSet.

19.1.4.3.12 TIBDataSet.BeforeClose Property

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

Description

This is BeforeClose, a member of class TIBDataSet.

19.1.4.3.13 TIBDataSet.BeforeDelete Property

Pascal

```
property BeforeDelete;
```

C++

```
__property BeforeDelete;
```

Description

This is BeforeDelete, a member of class TIBDataSet.

19.1.4.3.14 TIBDataSet.BeforeEdit Property

Pascal

```
property BeforeEdit;
```

C++

```
__property BeforeEdit;
```

Description

This is BeforeEdit, a member of class TIBDataSet.

19.1.4.3.15 TIBDataSet.BeforeInsert Property

Pascal

```
property BeforeInsert;
```

C++

```
__property BeforeInsert;
```

Description

This is BeforeInsert, a member of class TIBDataSet.

19.1.4.3.16 TIBDataSet.BeforeOpen Property

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

Description

This is BeforeOpen, a member of class TIBDataSet.

19.1.4.3.17 TIBDataSet.BeforePost Property

Pascal

```
property BeforePost;
```

C++

```
__property BeforePost;
```

Description

This is BeforePost, a member of class TIBDataSet.

19.1.4.3.18 TIBDataSet.BeforeScroll Property

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

Description

This is BeforeScroll, a member of class TIBDataSet.

19.1.4.3.19 TIBDataSet.BufferChunks Property

Specifies the dataset buffer chunk size.

Pascal

```
property BufferChunks: Integer;
```

C++

```
__property int BufferChunks;
```

Description

Use BufferChunks to get or set the number of records for which the dataset allocates buffer space at any time. When the dataset's buffer is full, trying to fetch an additional record causes the dataset to reallocate the buffer so that it has enough memory to hold an additional BufferChunks records.

Note: When Unidirectional is true, BufferChunks has no meaning.

19.1.4.3.20 TIBDataSet.CachedUpdates Property

Specifies whether cached updates are enabled for a dataset.

Pascal

```
property CachedUpdates: Boolean;
```

C++

```
__property Boolean CachedUpdates;
```

Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is true, cached updates are enabled. If CachedUpdates is false, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Note: **Note**: Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset.

19.1.4.3.21 TIBDataSet.DataSource Property

Pascal

```
property DataSource;
```

C++

```
__property DataSource;
```

Description

This is DataSource, a member of class TIBDataSet.

19.1.4.3.22 TIBDataSet.DeleteSQL Property

Holds the SQL statement used to delete rows from the dataset.

Pascal

```
property DeleteSQL: TStrings;
```

C++

```
__property TStrings DeleteSQL;
```

Description

Use DeleteSQL to delete rows in the dataset.

19.1.4.3.23 TIBDataSet.Filtered Property

Specifies whether or not filtering is active for the dataset.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. Otherwise Filtered is false. To apply filter conditions specified in the Filter property or the OnFilterRecord (see page 3669) event handler, set Filtered to true.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

19.1.4.3.24 TIBDataSet.GeneratorField Property

Specifies whether the dataset should refresh its data every time it posts a record.

Pascal

```
property GeneratorField: TIBGeneratorField;
```

C++

```
__property TIBGeneratorField GeneratorField;
```

Description

Use GeneratorField to specify a generator that automatically provides the value for one of the fields in the dataset. Use the properties of the TIBGeneratorField object to indicate which field's value should be set and how that value should be calculated.

Warning: Do not set the value of GeneratorField. To change the value of this property, change the properties of the TIBGeneratorField object that is the value of this property.

19.1.4.3.25 TIBDataSet.InsertSQL Property

Holds the SQL statement used to insert rows into the dataset.

Pascal

```
property InsertSQL: TStrings;
```

C++

```
__property TStrings InsertSQL;
```

Description

Use InsertSQL to insert rows into the dataset.

19.1.4.3.26 TIBDataSet.LiveMode Property

Indicates what types of edits, if any, can be made to the data in the dataset.

Pascal

```
property LiveMode: TLiveModes;
```

C++

```
__property TLiveModes LiveMode;
```

Description

Read LiveMode to determine what types of edits should be allowed. LiveMode is set automatically to restrict the capabilities of the dataset to reflect the user's rights.

Note: For read-only datasets, such as TIBQuery, LiveMode is always an empty set.

19.1.4.3.27 TIBDataSet.ModifySQL Property

Provides the ability to access the SQL object encapsulating the ModifySQL statement.

Pascal

```
property ModifySQL: TStrings;
```

C++

```
__property TStrings ModifySQL;
```

Description

Use ModifySQL to access the SQL object that encapsulates the ModifySQL statement.

19.1.4.3.28 TIBDataSet.OnCalcFields Property

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

Description

This is OnCalcFields, a member of class TIBDataSet.

19.1.4.3.29 TIBDataSet.OnDeleteError Property

Pascal

```
property OnDeleteError;
```

C++

```
__property OnDeleteError;
```

Description

This is OnDeleteError, a member of class TIBDataSet.

19.1.4.3.30 TIBDataSet.OnEditError Property

Pascal

```
property OnEditError;
```

C++

```
__property OnEditError;
```

Description

This is OnEditError, a member of class TIBDataSet.

19.1.4.3.31 TIBDataSet.OnFilterRecord Property

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

This is OnFilterRecord, a member of class TIBDataSet.

19.1.4.3.32 TIBDataSet.OnNewRecord Property

Pascal

```
property OnNewRecord;
```

C++

```
__property OnNewRecord;
```

Description

This is OnNewRecord, a member of class TIBDataSet.

19.1.4.3.33 TIBDataSet.OnPostError Property

Pascal

```
property OnPostError;
```

C++

```
__property OnPostError;
```

Description

This is OnPostError, a member of class TIBDataSet.

19.1.4.3.34 TIBDataSet.ParamCheck Property

Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether the Params property is cleared and regenerated if an application modifies the SQL in this dataset's query. By default ParamCheck is true, meaning that the Params property is automatically regenerated at runtime. When ParamCheck is true, the proper number of parameters is guaranteed to be generated for the current SQL statement.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement that are not parameters for the TIBCustomDataSet. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TIBCustomDataSet executing the DDL statement.

19.1.4.3.35 TIBDataSet.Params Property

Provides the ability to specify values for a parameterized query.

Pascal

```
property Params: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA Params;
```

Description

Use Params to specify values for a parameterized query.

For example:

```
DataSet1.Params[0].AsInteger := 24
 DataSet1.Params.ByName['Field2'].AsString := 'foo'
    DataSet1->Params->Items[0]->AsInteger = 24;
 DataSet1->Params->ByName["Field2"]->AsString = "foo";
```

19.1.4.3.36 TIBDataSet.Prepared Property

Determines whether or not a set of dataset queries is prepared for execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Examine Prepared to determine if a set of queries is already prepared for execution. If Prepared is true, the set of queries is prepared, and if Prepared is false, the set of queries is not prepared. While a set of queries need not be prepared before execution, execution performance is enhanced if the set of queries is prepared beforehand, particularly if it is a parameterized set of queries that is executed more than once using the same parameter values.

19.1.4.3.37 TIBDataSet.RefreshSQL Property

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

Pascal

```
property RefreshSQL: TStrings;
```

C++

```
__property TStrings RefreshSQL;
```

Description

Use RefreshSQL to access the SQL object that encapsulates the RefreshSQL statement.

19.1.4.3.38 TIBDataSet.SelectSQL Property

Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.

Pascal

```
property SelectSQL: TStrings;
```

C++

```
__property TStrings SelectSQL;
```

Description

Use SelectSQL to access the SQL object that encapsulates the SelectSQL statement.

19.1.4.3.39 TIBDataSet.SelectStmtHandle Property

Gets the handle for the SELECT statement.

Pascal

```
property SelectStmtHandle: TISC_STMT_HANDLE;
```

C++

```
__property TISC_STMT_HANDLE SelectStmtHandle;
```

Description

Use SelectStmtHandle to get the handle for the SELECT statement.

19.1.4.3.40 TIBDataSet.StatementType Property

Returns the statement type of the QSelect query.

Pascal

```
property StatementType: TIBSQLTypes;
```

C++

```
__property TIBSQLTypes StatementType;
```

Description

Use StatementType to determine the statement type of a QSelect query.

19.1.4.3.41 TIBDataSet.UniDirectional Property

Determines whether bidirectional cursors are enabled for the dataset.

Pascal

```
property UniDirectional: Boolean;
```

C++

```
__property Boolean UniDirectional;
```

Description

Use UniDirectional to determine whether bidirectional cursors are enabled for this dataset.

Note: Do not confuse the UniDirectional property with the IsUniDirectional property inherited from TDataSet. IsUniDirectional indicates whether the dataset is a dbExpress dataset, which, in addition to using a unidirectional cursor, does not buffer records. UniDirectional is only concerned with whether the cursor permits backward or random navigation.

19.1.4.3.42 TIBDataSet.UpdateObject Property

Specifies the update object component used to update a read-only result set.

Pascal

```
property UpdateObject: TIBDataSetUpdateObject;
```

C++

```
__property TIBDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TIBDataSetUpdateObject component to use in an application that must be able to update a read-only result set.

In some cases, such as when using TIBQuery, the dataset does not automatically update records. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

TIBDataSetUpdateObject is especially useful for complicated updates from an OnUpdateRecord event.

19.1.4.4 TIBDataSet Events

The events of the TIBDataSet class are listed here.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 3673)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3673)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3673)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3674)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 3674)	Occurs when the database associated with this dataset is freed.
	TransactionFree (see page 3674)	Occurs when the transaction that includes this dataset's query is freed.

19.1.4.4.1 TIBDataSet.AfterDatabaseDisconnect Event

Occurs immediately after the connection to the associated database is closed.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to respond when the dataset closes its connection to the database specified by the Database property.

19.1.4.4.2 TIBDataSet.AfterTransactionEnd Event

Occurs immediately after a transaction that includes this dataset's query ends.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to respond when the transaction specified by the Transaction property ends.

19.1.4.4.3 TIBDataSet.BeforeDatabaseDisconnect Event

Occurs immediately before the connection to the associated database is closed.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to respond when the dataset is about to close its connection to the database

specified by the Database property.

19.1.4.4.4 TIBDataSet.BeforeTransactionEnd Event

Occurs immediately before a transaction that includes this dataset's query ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to respond when the transaction specified by the Transaction property is about to end.

19.1.4.4.5 TIBDataSet.DatabaseFree Event

Occurs when the database associated with this dataset is freed.

Pascal

```
property DatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent DatabaseFree;
```

Description

Write a DatabaseFree event handler to respond when the database specified by the Database property is freed.

19.1.4.4.6 TIBDataSet.TransactionFree Event

Occurs when the transaction that includes this dataset's query is freed.

Pascal

```
property TransactionFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent TransactionFree;
```

Description

Write a TransactionFree event handler to respond when the transaction specified by the Transaction property is freed.

19.1.5 TIBDataSetUpdateObject Class

TIBDataSetUpdateObject is the abstract base class for update objects used to update otherwise unupdateable queries when cached updates are enabled.

Class Hierarchy

Pascal

```
TIBDataSetUpdateObject = class(TComponent);
```

C++

```
class TIBDataSetUpdateObject : public TComponent;
```

File

IBCustomDataSet

Description

Use TIBDataSetUpdateObject as a base class when creating customized update objects that can be used to update datasets. TIBDataSetUpdateObject declares a single property and some abstract methods, but provides no implementation details: these must be provided by descendant objects.

19.1.5.1 TIBDataSetUpdateObject Members

The following tables list the members exposed by TIBDataSetUpdateObject.

Protected Methods

	Name	Description
	Apply (see page 3676)	This is Apply, a member of class TIBDataSetUpdateObject.
	GetDataSet (see page 3676)	This is GetDataSet, a member of class TIBDataSetUpdateObject.
	GetSQL (see page 3677)	This is GetSQL, a member of class TIBDataSetUpdateObject.
	SetDataSet (see page 3677)	This is SetDataSet, a member of class TIBDataSetUpdateObject.

Protected Properties

	Name	Description
	DataSet (see page 3677)	This is DataSet, a member of class TIBDataSetUpdateObject.

Public Methods

	Name	Description
	Create (see page 3675)	Creates an instance of a TIBDataSetUpdateObject (see page 3674) component.
	Destroy (see page 3676)	Frees all resources associated with this instance.

Published Properties

	Name	Description
	RefreshSQL (see page 3677)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

19.1.5.2 TIBDataSetUpdateObject.Create Constructor

Creates an instance of a TIBDataSetUpdateObject (see page 3674) component.

Pascal

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

C++

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

Description

Call Create to create an instance of a TIBDataSetUpdateObject (see page 3674) component.

19.1.5.3 TIBDataSetUpdateObject.Destroy Destructor

Frees all resources associated with this instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBDataSetUpdateObject();
```

Description

Do not call Destroy directly in an application. Usually destruction of objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not already freed before calling Destroy.

19.1.5.4 TIBDataSetUpdateObject Methods

The methods of the TIBDataSetUpdateObject class are listed here.

Protected Methods

	Name	Description
	Apply (see page 3676)	This is Apply, a member of class TIBDataSetUpdateObject.
	GetDataSet (see page 3676)	This is GetDataSet, a member of class TIBDataSetUpdateObject.
	GetSQL (see page 3677)	This is GetSQL, a member of class TIBDataSetUpdateObject.
	SetDataSet (see page 3677)	This is SetDataSet, a member of class TIBDataSetUpdateObject.

19.1.5.4.1 TIBDataSetUpdateObject.Apply Method

Pascal

```
procedure Apply(UpdateKind: TUpdateKind); virtual; abstract;
```

C++

```
virtual __fastcall Apply(TUpdateKind UpdateKind) = 0;
```

Description

This is Apply, a member of class TIBDataSetUpdateObject.

19.1.5.4.2 TIBDataSetUpdateObject.GetDataSet Method

Pascal

```
function GetDataSet: TIBCustomDataSet; virtual; abstract;
```

C++

```
virtual __fastcall TIBCustomDataSet GetDataSet() = 0;
```

Description

This is GetDataSet, a member of class TIBDataSetUpdateObject.

19.1.5.4.3 TIBDataSetUpdateObject.GetSQL Method

Pascal

```
function GetSQL(UpdateKind: TUpdateKind): TStrings; virtual; abstract;
```

C++

```
virtual __fastcall TStrings GetSQL(TUpdateKind UpdateKind) = 0;
```

Description

This is GetSQL, a member of class TIBDataSetUpdateObject.

19.1.5.4.4 TIBDataSetUpdateObject.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TIBCustomDataSet); virtual; abstract;
```

C++

```
virtual __fastcall SetDataSet(TIBCustomDataSet ADataSet) = 0;
```

Description

This is SetDataSet, a member of class TIBDataSetUpdateObject.

19.1.5.5 TIBDataSetUpdateObject Properties

The properties of the TIBDataSetUpdateObject class are listed here.

Protected Properties

	Name	Description
	DataSet (see page 3677)	This is DataSet, a member of class TIBDataSetUpdateObject.

Published Properties

	Name	Description
	RefreshSQL (see page 3677)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

19.1.5.5.1 TIBDataSetUpdateObject.DataSet Property

Pascal

```
property DataSet: TIBCustomDataSet;
```

C++

```
__property TIBCustomDataSet DataSet;
```

Description

This is DataSet, a member of class TIBDataSetUpdateObject.

19.1.5.5.2 TIBDataSetUpdateObject.RefreshSQL Property

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

Pascal

```
property RefreshSQL: TStrings;
```

C++

```
__property TStrings RefreshSQL;
```

Description

Use RefreshSQL to access the SQL object that encapsulates the RefreshSQL statement.

19.1.6 TIBDSBlobStream Class

TIBDSBlobStream reads from and writes to BLOB fields on behalf of an IB (see page 3541) Dataset.

Class Hierarchy

Pascal

```
TIBDSBlobStream = class(TStream);
```

C++

```
class TIBDSBlobStream : public TStream;
```

File

IBCustomDataSet

Description

TIBDSBlobStream is the type of stream returned when an application calls an IB (see page 3541) dataset's CreateBlobStream method. Use this stream to access or modify the value of a Blob field in the dataset. TIBDSBlobStream allows objects that have no specialized knowledge of how data is stored in a Blob field to read or write such data by employing the uniform stream interface.

To use TIBDSBlobStream, call CreateBlobStream to obtain an instance of the stream and use the methods of the stream to read or write the data. The dataset retains ownership of the stream, so you should not free this Blob stream when you are done.

19.1.6.1 TIBDSBlobStream Members

The following tables list the members exposed by TIBDSBlobStream.

Protected Fields

	Name	Description
	FBlobStream (see page 3679)	This is FBlobStream, a member of class TIBDSBlobStream.
	FField (see page 3680)	This is FField, a member of class TIBDSBlobStream.
	FModified (see page 3680)	This is FModified, a member of class TIBDSBlobStream.

Public Methods

	Name	Description
	Create (see page 3679)	Resets the stream to the beginning of the Blob.
V	Destroy (see page 3679)	Frees the instance of TIBDSBlobStream (see page 3678).
V	Read (see page 3680)	Reads the requested number of bytes from the Blob.
V	Seek (see page 3681)	Resets the current position of the TIBDSBlobStream (see page 3678) object.

	SetSize (see page 3681)	Set the size of the Blob to the requested size.
	Write (see page 3681)	Sets the field to be modified.

19.1.6.2 TIBDSBlobStream.Create Constructor

Resets the stream to the beginning of the Blob.

Pascal

```
constructor Create(AField: TField; ABlobStream: TIBBlobStream; Mode: TBlobStreamMode);
```

C++

```
__fastcall TIBDSBlobStream(TField AField, TIBBlobStream ABlobStream, TBlobStreamMode Mode);
```

Description

Call Create to reset the stream to the beginning of the Blob. Depending on the mode, it also truncates the Blob stream.

19.1.6.3 TIBDSBlobStream.Destroy Destructor

Frees the instance of TIBDSBlobStream (see page 3678).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBDSBlobStream();
```

Description

Destroy frees the blob stream object and writes the BLOB value to the underlying TBlobField object.

Do not explicitly destroy the instance of TIBDSBlobStream (see page 3678) that you obtain by calling a dataset's CreateBlobStream method. The Dataset retains ownership of this stream and is responsible for freeing it.

When working with a Blob stream that you explicitly create and destroy, use the TIBBlobStream (see page 3580) class instead.

19.1.6.4 TIBDSBlobStream Fields

The fields of the TIBDSBlobStream class are listed here.

Protected Fields

	Name	Description
	FBlobStream (see page 3679)	This is FBlobStream, a member of class TIBDSBlobStream.
	FField (see page 3680)	This is FField, a member of class TIBDSBlobStream.
	FModified (see page 3680)	This is FModified, a member of class TIBDSBlobStream.

19.1.6.4.1 TIBDSBlobStream.FBlobStream Field

Pascal

```
FBlobStream: TIBBlobStream;
```

C++

```
TIBBlobStream FBlobStream;
```

Description

This is FBlobStream, a member of class TIBDSBlobStream.

19.1.6.4.2 TIBDSBlobStream.FField Field

Pascal

```
FField: TField;
```

C++

```
TField FField;
```

Description

This is FField, a member of class TIBDSBlobStream.

19.1.6.4.3 TIBDSBlobStream.FModified Field

Pascal

```
FModified: Boolean;
```

C++

```
Boolean FModified;
```

Description

This is FModified, a member of class TIBDSBlobStream.

19.1.6.5 TIBDSBlobStream Methods

The methods of the TIBDSBlobStream class are listed here.

Public Methods

	Name	Description
	Read (see page 3680)	Reads the requested number of bytes from the Blob.
	Seek (see page 3681)	Resets the current position of the TIBDSBlobStream (see page 3678) object.
	SetSize (see page 3681)	Set the size of the Blob to the requested size.
	Write (see page 3681)	Sets the field to be modified.

19.1.6.5.1 TIBDSBlobStream.Read Method

Reads the requested number of bytes from the Blob.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Call Read to read data from the Blob field when the number of bytes in the field's data is not known. Buffer must have at least Count bytes allocated to hold the data that was read from the field.

Read transfers up to Count bytes from the Blob data into Buffer, starting in the current position, and then advances the current

position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read checks the Transliterate property of the field, and converts the data into ANSI from the character set specified by the dataset if Transliterate is true.

All the other data-reading methods of a Blob stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

Do not call Read when the TIBDSBlobStream (see page 3678) was created in bmWrite mode.

19.1.6.5.2 TIBDSBlobStream.Seek Method

Resets the current position of the TIBDSBlobStream (see page 3678) object.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position within the Blob data by the indicated offset. Seek allows an application to read from or write to a particular location within the Blob data.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	Offset is from the beginning of the Blob data Seek moves to the position Offset Offset must be >= 0
soFromCurrent	Offset is from the current position in the Blob data Seek moves to Position + Offset
soFromEnd	Offset is from the end of the Blob data Offset must be <= 0 to indicate a number of bytes before the end of the Blob

Seek returns the new value of the Position property, the new current position in the Blob data.

19.1.6.5.3 TIBDSBlobStream.SetSize Method

Set the size of the Blob to the requested size.

Pascal

```
procedure SetSize(NewSize: Longint); override;
```

C++

```
virtual __fastcall SetSize(Longint NewSize);
```

Description

Call SetSize to set the size of the Blob to the requested size.

19.1.6.5.4 TIBDSBlobStream.Write Method

Sets the field to be modified.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to set the field to be modified, write the requested number of bytes to the Blob stream.

19.1.7 TIBGeneratorField Class

TIBGeneratorField calls the server to generate values for a field in an IB (see page 3541) dataset.

Class Hierarchy

Pascal

```
TIBGeneratorField = class(TPersistent);
```

C++

```
class TIBGeneratorField : public TPersistent;
```

File

IBCustomDataSet

Description

Assign (see page 3684) an instance of TIBGeneratorField as the value of a dataset's GeneratorField property to indicate that the value of a specified field in the dataset can be generated by the server.

Use the Field (see page 3685) property to specify the field whose value is generated by the TIBGeneratorField object. Use the Generator (see page 3685) and IncrementBy (see page 3686) properties to specify how the server should generate field values. Use the ApplyEvent (see page 3685) property to indicate when the TIBGeneratorField should call the server to generate field values.

19.1.7.1 TIBGeneratorField Members

The following tables list the members exposed by TIBGeneratorField.

Public Methods

	Name	Description
	Apply (see page 3683)	Calls the generator on the server to generate a field value and assigns it to the associated field.
	Assign (see page 3684)	Copies the properties of another object.
	Create (see page 3683)	Creates an instance of TIBGeneratorField (see page 3682).
	ValueName (see page 3684)	Returns a string that describes the value that the IB (see page 3541) generate fetches.

Published Properties

	Name	Description
	ApplyEvent (see page 3685)	Indicates when the TIBGeneratorField (see page 3682) instance calls the server to generate a value for the associated field.

	Field (see page 3685)	Specifies the field whose value the TIBGeneratorField (see page 3682) object generates.
	Generator (see page 3685)	Names the generator on the server that TIBGeneratorField (see page 3682) calls to generate field values.
	IncrementBy (see page 3686)	Names the amount by which a generator on the server increments the value of the associated field.

19.1.7.2 TIBGeneratorField.Create Constructor

Creates an instance of TIBGeneratorField (see page 3682).

Pascal

```
constructor Create(ADataSet: TIBCustomDataSet);
```

C++

```
__fastcall TIBGeneratorField(TIBCustomDataSet ADataSet);
```

Description

Do not create instances of TIBGeneratorField (see page 3682). This class is designed to work with an IB (see page 3541) dataset and is automatically created by the dataset to implement its GeneratorField property.

ADataSet is the dataset that creates the TIBGeneratorField (see page 3682) instance and which contains the field whose value the generator field assigns.

19.1.7.3 TIBGeneratorField Methods

The methods of the TIBGeneratorField class are listed here.

Public Methods

	Name	Description
	Apply (see page 3683)	Calls the generator on the server to generate a field value and assigns it to the associated field.
	Assign (see page 3684)	Copies the properties of another object.
	ValueName (see page 3684)	Returns a string that describes the value that the IB (see page 3541) generate fetches.

19.1.7.3.1 TIBGeneratorField.Apply Method

Calls the generator on the server to generate a field value and assigns it to the associated field.

Pascal

```
procedure Apply;
```

C++

```
__fastcall Apply();
```

Description

Call Apply to fetch the server-generated value for the field identified by the Field (see page 3685) property and assign that value to the field.

If the ApplyEvent (see page 3685) property is not gamOnServer, the associated dataset calls Apply automatically at the point indicated by ApplyEvent (see page 3685).

19.1.7.3.2 TIBGeneratorField.Assign Method

Copies the properties of another object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties of another object on to this TIBGeneratorField (see page 3682) instance. TIBGeneratorField (see page 3682) overrides the inherited method to copy the Field (see page 3685), Generator (see page 3685), IncrementBy (see page 3686) and ApplyEvent (see page 3685) properties of the object specified by Source if it is another TIBGeneratorField (see page 3682) instance.

If Source is not another TIBGeneratorField (see page 3682), Assign calls the inherited method, which allows any objects that know how to assign themselves to a TIBGeneratorField (see page 3682) to copy their properties.

19.1.7.3.3 TIBGeneratorField.ValueName Method

Returns a string that describes the value that the IB (see page 3541) generate fetches.

Pascal

```
function ValueName: string;
```

C++

```
__fastcall UnicodeString ValueName();
```

Description

ValueName returns a string of the form

```
Generator -> Field By IncrementBy
```

where Generator (see page 3685) is the value of the Generator (see page 3685) property, Field (see page 3685) is the value of the Field (see page 3685) property, and IncrementBy (see page 3686) is the value of the IncrementBy (see page 3686) property.

19.1.7.4 TIBGeneratorField Properties

The properties of the TIBGeneratorField class are listed here.

Published Properties

	Name	Description
	ApplyEvent (see page 3685)	Indicates when the TIBGeneratorField (see page 3682) instance calls the server to generate a value for the associated field.
	Field (see page 3685)	Specifies the field whose value the TIBGeneratorField (see page 3682) object generates.
	Generator (see page 3685)	Names the generator on the server that TIBGeneratorField (see page 3682) calls to generate field values.
	IncrementBy (see page 3686)	Names the amount by which a generator on the server increments the value of the associated field.

19.1.7.4.1 TIBGeneratorField.ApplyEvent Property

Indicates when the TIBGeneratorField (see page 3682) instance calls the server to generate a value for the associated field.

Pascal

```
property ApplyEvent: TIBGeneratorApplyEvent;
```

C++

```
__property TIBGeneratorApplyEvent ApplyEvent;
```

Description

Set TIBGeneratorApplyEvent (see page 3691) to indicate when the server generates field values using this generator.

Note: Regardless of the value of ApplyEvent, applications can generate the field value at any time by calling the Apply (see page 3683) method.

19.1.7.4.2 TIBGeneratorField.Field Property

Specifies the field whose value the TIBGeneratorField (see page 3682) object generates.

Pascal

```
property Field: string;
```

C++

```
__property UnicodeString Field;
```

Description

Set Field to the name of the field whose value the generator sets. When the event specified by the ApplyEvent (see page 3685) property occurs, or when an application calls the Apply (see page 3683) method, the TIBGeneratorField (see page 3682) object calls the server to generate a field value and assigns that value to the specified field.

Field must be the FieldName property of one of the fields in the associated dataset's Fields property list. The TIBGeneratorField (see page 3682) object is associated with a dataset when it is the value of that dataset's GeneratorField property.

19.1.7.4.3 TIBGeneratorField.Generator Property

Names the generator on the server that TIBGeneratorField (see page 3682) calls to generate field values.

Pascal

```
property Generator: string;
```

C++

```
__property UnicodeString Generator;
```

Description

TIBGeneratorField (see page 3682) uses Generator, along with the IncrementBy (see page 3686) property, to generate the SQL statement that fetches the generated field value. The generated SQL statement has the form

SELECT GEN_ID(Generator, IncrementBy (see page 3686)) FROM RDB$DATABASE

Where 'Generator' is replaced by the value of Generator and 'IncrementBy (see page 3686)' is replaced by the value of IncrementBy (see page 3686).

19.1.7.4.4 TIBGeneratorField.IncrementBy Property

Names the amount by which a generator on the server increments the value of the associated field.

Pascal

```
property IncrementBy: Integer;
```

C++

```
__property int IncrementBy;
```

Description

TIBGeneratorField (see page 3682) uses IncrementBy, along with the Generator (see page 3685) property, to generate the SQL statement that fetches the generated field value. The generated SQL statement has the form

SELECT GEN_ID(Generator (see page 3685), IncrementBy) FROM RDB$DATABASE

Where 'Generator (see page 3685)' is replaced by the value of Generator (see page 3685) and 'IncrementBy' is replaced by the value of IncrementBy.

19.1.8 TIBStringField Class

TIBStringField allows for strings in excess of 8196 bytes of data.

Class Hierarchy

Pascal

```
TIBStringField = class(TStringField);
```

C++

```
class TIBStringField : public TStringField;
```

File

IBCustomDataSet

Description

A value of a string field is physically stored as a sequence of characters. Common uses for string fields are to store text, such as names and addresses.

TIBStringField introduces properties to translate between string values and other data types, and to manage language driver conversions. As a descendent of TStringField, TIBStringField inherits many properties, methods, and events that are useful for managing the value and properties of a field in a database. TIBStringField allows for strings in excess of 8196 bytes of data.

19.1.8.1 TIBStringField Members

The following tables list the members exposed by TIBStringField.

Public Methods

	Name	Description
	CheckTypeSize (see page 3687)	Indicates the type size of the string field.
	Create (see page 3687)	Creates an instance of a TIBStringField (see page 3686) object.

	GetAsString (see page 3688)	Returns the value of a field as a string.
	GetAsVariant (see page 3688)	Returns the value of a field as a variant.
	GetValue (see page 3688)	Returns the value of a field.
	SetAsString (see page 3688)	Sets the value of a field as a string type.

19.1.8.2 TIBStringField.Create Constructor

Creates an instance of a TIBStringField (see page 3686) object.

Pascal

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

C++

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

Description

It is seldom necessary to call Create directly, because a string field component is instantiated automatically for all string fields in a dataset.

After calling the inherited constructor, Create sets

- DataType to ftString.
- Size to 20.
- Transliterate to True.
- BlanksToNull to True

19.1.8.3 TIBStringField Methods

The methods of the TIBStringField class are listed here.

Public Methods

	Name	Description
	CheckTypeSize (see page 3687)	Indicates the type size of the string field.
	GetAsString (see page 3688)	Returns the value of a field as a string.
	GetAsVariant (see page 3688)	Returns the value of a field as a variant.
	GetValue (see page 3688)	Returns the value of a field.
	SetAsString (see page 3688)	Sets the value of a field as a string type.

19.1.8.3.1 TIBStringField.CheckTypeSize Method

Indicates the type size of the string field.

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

Call CheckTypeSize to determine the type size of the string field.

There is both a virtual method and a static method for the CheckTypeSize method.

All sizes are valid for a TIBString field.

19.1.8.3.2 TIBStringField.GetAsString Method

Returns the value of a field as a string.

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall UnicodeString GetAsString();
```

Description

Call GetAsString to return the value of a field as a string.

19.1.8.3.3 TIBStringField.GetAsVariant Method

Returns the value of a field as a variant.

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

Call GetAsVariant return the value of a field as type Variant.

19.1.8.3.4 TIBStringField.GetValue Method

Returns the value of a field.

Pascal

```
function GetValue(var Value: string): Boolean;
```

C++

```
__fastcall Boolean GetValue(UnicodeString Value);
```

Description

Call GetValue to return the value of a field.

19.1.8.3.5 TIBStringField.SetAsString Method

Sets the value of a field as a string type.

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const UnicodeString Value);
```

Description

Call SetAsString to set the value of a field as a string type.

19.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCachedUpdateStatus (see page 3689)	TCachedUpdateStatus indicates the status of a record in a dataset that has cached updates.
	TIBGeneratorApplyEvent (see page 3691)	TIBGeneratorApplyEvent indicates when a generator should create field values.
	TIBUpdateAction (see page 3691)	TIBUpdateAction indicates the action that a dataset should take after an event handler has responded to an update error.
	TLiveMode (see page 3692)	This is record IBCustomDataSet.TLiveMode.

Records

	Name	Description
	TFieldData (see page 3690)	This is record IBCustomDataSet.TFieldData.
	TIBDBKey (see page 3690)	TIBDBKey stores the key for a row in an InterBase table.
	TRecordData (see page 3693)	This is record IBCustomDataSet.TRecordData.

19.2.1 IBCustomDataSet.TCachedUpdateStatus Enumeration

TCachedUpdateStatus indicates the status of a record in a dataset that has cached updates.

Pascal

```
TCachedUpdateStatus = (
  cusUnmodified,
  cusModified,
  cusInserted,
  cusDeleted,
  cusUninserted
);
```

C++

```
enum TCachedUpdateStatus {
  cusUnmodified,
  cusModified,
  cusInserted,
  cusDeleted,
  cusUninserted
};
```

File

IBCustomDataSet

Description

TCachedUpdateStatus values are:

Value Meaning

cusModified Modified records.

cusInserted Inserted records.

cusDeleted Deleted records.

cusUnmodified Unmodified records.

cusUninserted Uninserted records.

19.2.2 IBCustomDataSet.TFieldData Record

Pascal

```
TFieldData = record
  fdDataType: Short;
  fdDataScale: Short;
  fdNullable: Boolean;
  fdIsNull: Boolean;
  fdDataSize: Short;
  fdDataLength: Short;
  fdDataOfs: Integer;
end;
```

C++

```
struct TFieldData {
  Short fdDataType;
  Short fdDataScale;
  Boolean fdNullable;
  Boolean fdIsNull;
  Short fdDataSize;
  Short fdDataLength;
  int fdDataOfs;
};
```

File

IBCustomDataSet

Description

This is record IBCustomDataSet.TFieldData.

19.2.3 IBCustomDataSet.TIBDBKey Record

TIBDBKey stores the key for a row in an InterBase table.

Pascal

```
TIBDBKey = record
  DBKey: array[0..7] of Byte;
end;
```

C++

```
struct TIBDBKey {
  array[0..7] of Byte DBKey;
};
```

File

IBCustomDataSet

Description

TIBDBKey stores the unique row identifier for the current row in the current transaction.

19.2.4 IBCustomDataSet.TIBGeneratorApplyEvent Enumeration

TIBGeneratorApplyEvent indicates when a generator should create field values.

Pascal

```
TIBGeneratorApplyEvent = (
  gamOnNewRecord,
  gamOnPost,
  gamOnServer
);
```

C++

```
enum TIBGeneratorApplyEvent {
  gamOnNewRecord,
  gamOnPost,
  gamOnServer
};
```

File

IBCustomDataSet

Description

TIBGeneratorApplyEvent values are:

Value Meaning

gamOnNewRecord The dataset calls the server to generate the field value immediately after it inserts or appends a new record (before the OnNewRecord event).

gamOnPost The dataset calls the server to generate the field value immediately before it posts a record to the server (before the BeforePost event).

gamOnServer The dataset does not need to call the server to generate field values: the server generates these values automatically. With gamOnServer, TIBGeneratorField (see page 3682) serves primarily to ensure that the user is not required to enter a value in the field.

19.2.5 IBCustomDataSet.TIBUpdateAction Enumeration

TIBUpdateAction indicates the action that a dataset should take after an event handler has responded to an update error.

Pascal

```
TIBUpdateAction = (
  uaFail,
  uaAbort,
  uaSkip,
  uaRetry,
  uaApply,
```

```
    uaApplied
  );
```

C++

```
  enum TIBUpdateAction {
    uaFail,
    uaAbort,
    uaSkip,
    uaRetry,
    uaApply,
    uaApplied
  };
```

File

IBCustomDataSet

Description

TIBUpdateAction values are:

Value Meaning

uaFail Abort the update operation and display an error message

uaAbort Abort the update operation without displaying an error message

uaSkip Skip updating the record that raised the error condition, and leave the unapplied changes in the cache

uaRetry Repeat the update operation that originally raised the error condition

uaApplied The event handler applied the update (not used in error handling event handlers). Free the current record update from the cache.

uaApply For internal use only

19.2.6 IBCustomDataSet.TLiveMode Enumeration

Pascal

```
  TLiveMode = (
    lmInsert,
    lmModify,
    lmDelete,
    lmRefresh
  );
```

C++

```
  enum TLiveMode {
    lmInsert,
    lmModify,
    lmDelete,
    lmRefresh
  };
```

File

IBCustomDataSet

Description

This is record IBCustomDataSet.TLiveMode.

19.2.7 IBCustomDataSet.TRecordData Record

Pascal

```
TRecordData = record
  rdBookmarkFlag: TBookmarkFlag;
  rdFieldCount: Short;
  rdRecordNumber: Long;
  rdCachedUpdateStatus: TCachedUpdateStatus;
  rdUpdateStatus: TUpdateStatus;
  rdSavedOffset: DWORD;
  rdDBKey: TIBDBKey;
  rdFields: array[1..1] of TFieldData;
end;
```

C++

```
struct TRecordData {
  TBookmarkFlag rdBookmarkFlag;
  Short rdFieldCount;
  Long rdRecordNumber;
  TCachedUpdateStatus rdCachedUpdateStatus;
  TUpdateStatus rdUpdateStatus;
  DWORD rdSavedOffset;
  TIBDBKey rdDBKey;
  array[1..1] of TFieldData rdFields;
};
```

File

IBCustomDataSet

Description

This is record IBCustomDataSet.TRecordData.

19.3 Types

The following table lists types in this documentation.

Types

Name	Description
PBlobDataArray (see page 3694)	This is type IBCustomDataSet.PBlobDataArray.
PDateTime (see page 3694)	This is type IBCustomDataSet.PDateTime.
PFieldData (see page 3695)	This is type IBCustomDataSet.PFieldData.
PIBDBKey (see page 3695)	This is type IBCustomDataSet.PIBDBKey.
PRecordData (see page 3695)	This is type IBCustomDataSet.PRecordData.
TBlobDataArray (see page 3696)	This is type IBCustomDataSet.TBlobDataArray.
TIBUpdateErrorEvent (see page 3696)	TIBUpdateErrorEvent is the type of event handlers that respond when an IBX dataset encounters an error applying a cached update.
TIBUpdateRecordEvent (see page 3696)	TIBUpdateRecordEvent is the type of event handlers that respond when an IBX dataset applies a cached update.
TIBUpdateRecordTypes (see page 3697)	TIBUpdateRecordTypes is a subset of all possible record types in a dataset that caches updates.
TLiveModes (see page 3697)	TLiveModes and TLiveModes indicate what operations an application can perform on an IBX dataset.

19.3.1 IBCustomDataSet.PBlobDataArray Type

Pascal

```
PBlobDataArray = ^TBlobDataArray;
```

C++

```
^TBlobDataArray PBlobDataArray;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.PBlobDataArray.

19.3.2 IBCustomDataSet.PDateTime Type

Pascal

```
PDateTime = ^TDateTime;
```

C++

```
^TDateTime PDateTime;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.PDateTime.

19.3.3 IBCustomDataSet.PFieldData Type

Pascal

```
PFieldData = ^TFieldData;
```

C++

```
^TFieldData PFieldData;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.PFieldData.

19.3.4 IBCustomDataSet.PIBDBKey Type

Pascal

```
PIBDBKey = ^TIBDBKey;
```

C++

```
^TIBDBKey PIBDBKey;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.PIBDBKey.

19.3.5 IBCustomDataSet.PRecordData Type

Pascal

```
PRecordData = ^TRecordData;
```

C++

```
^TRecordData PRecordData;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.PRecordData.

19.3.6 IBCustomDataSet.TBlobDataArray Type

Pascal

```
TBlobDataArray = array[0..0] of TIBBlobStream;
```

C++

```
array[0..0] of TIBBlobStream TBlobDataArray;
```

File

IBCustomDataSet

Description

This is type IBCustomDataSet.TBlobDataArray.

19.3.7 IBCustomDataSet.TIBUpdateErrorEvent Type

TIBUpdateErrorEvent is the type of event handlers that respond when an IBX dataset encounters an error applying a cached update.

Pascal

```
TIBUpdateErrorEvent = procedure (DataSet: TDataSet; E: EDatabaseError; UpdateKind:
TUpdateKind; var UpdateAction: TIBUpdateAction) of object;
```

C++

```
(DataSet: TDataSet; E: EDatabaseError; UpdateKind: TUpdateKind; var UpdateAction:
TIBUpdateAction) ( TIBUpdateErrorEvent)();
```

File

IBCustomDataSet

Description

DataSet is the TIBCustomDataSet (see page 3602) descendant that is attempting to apply cached updates.

E is an EDBEngineError object from which an application can extract an error message and the actual cause of the error condition. An OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind indicates whether the error occurred while inserting, deleting, or modifying a record.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

19.3.8 IBCustomDataSet.TIBUpdateRecordEvent Type

TIBUpdateRecordEvent is the type of event handlers that respond when an IBX dataset applies a cached update.

Pascal

```
TIBUpdateRecordEvent = procedure (DataSet: TDataSet; UpdateKind: TUpdateKind; var
```

```
  UpdateAction: TIBUpdateAction) of object;
```

C++

```
  (DataSet: TDataSet; UpdateKind: TUpdateKind; var UpdateAction: TIBUpdateAction) (
  TIBUpdateRecordEvent)();
```

File

IBCustomDataSet

Description

DataSet is the dataset that is applying updates.

UpdateKind whether the current update is the insertion of a record, the deletion of a record, or the modification of a record.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

19.3.9 IBCustomDataSet.TIBUpdateRecordTypes Type

TIBUpdateRecordTypes is a subset of all possible record types in a dataset that caches updates.

Pascal

```
  TIBUpdateRecordTypes = set of TCachedUpdateStatus;
```

C++

```
  TCachedUpdateStatus TIBUpdateRecordTypes;
```

File

IBCustomDataSet

Description

TIBUpdateRecordTypes is a set of TCachedUpdateStatus (see page 3689) values. It describes the different types of status allowed for records in a dataset that caches updates if they are to be included in a specified set.

19.3.10 IBCustomDataSet.TLiveModes Type

TLiveModes and TLiveModes indicate what operations an application can perform on an IBX dataset.

Pascal

```
  TLiveModes = set of TLiveMode;
```

C++

```
  TLiveMode TLiveModes;
```

File

IBCustomDataSet

Description

TLiveModes is a set of TLiveMode (see page 3692) values. The following table lists the possible values:

Value	Meaning
lmInsert	The application can insert records.
lmModify	The application can modify existing records.
lmDelete	The application can delete records.
lmRefresh	The application can refresh the records in the dataset.

19.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CreateProviderProc (see page 3699)	This is variable IBCustomDataSet.CreateProviderProc.

19.4.1 IBCustomDataSet.CreateProviderProc Variable

Pascal

```
CreateProviderProc: function(DataSet: TIBCustomDataSet): IProviderSupport = nil;
```

C++

```
function(DataSet: TIBCustomDataSet): IProviderSupport CreateProviderProc = nil;
```

File

IBCustomDataSet

Description

This is variable IBCustomDataSet.CreateProviderProc.

19.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BufferCacheSize (see page 3700)	This is constant IBCustomDataSet.BufferCacheSize.
DefaultFieldClasses (see page 3700)	This is constant IBCustomDataSet.DefaultFieldClasses.
UniCache (see page 3701)	This is constant IBCustomDataSet.UniCache.

19.5.1 IBCustomDataSet.BufferCacheSize Constant

Pascal

```
BufferCacheSize = 1000;
```

C++

```
BufferCacheSize = 1000;
```

File

IBCustomDataSet

Description

This is constant IBCustomDataSet.BufferCacheSize.

19.5.2 IBCustomDataSet.DefaultFieldClasses Constant

Pascal

```
DefaultFieldClasses: array[TFieldType] of TFieldClass = ( nil, TIBStringField, TSmallintField,
TIntegerField, TWordField, TBooleanField, TFloatField, TCurrencyField, TIBBCDField,
TDateField, TTimeField, TDateTimeField, TBytesField, TVarBytesField, TAutoIncField,
TBlobField, TMemoField, TGraphicField, TBlobField, TBlobField, TBlobField, TBlobField, nil,
TStringField, nil, TLargeIntField, TADTField, TArrayField, TReferenceField, TDataSetField,
TBlobField, TMemoField, TVariantField, TInterfaceField, TIDispatchField, TGuidField,
TSQLTimeStampField, TFMTBcdField, nil, nil, nil, nil);
```

C++

```
array[TFieldType] of TFieldClass DefaultFieldClasses = ( nil, TIBStringField, TSmallintField,
TIntegerField, TWordField, TBooleanField, TFloatField, TCurrencyField, TIBBCDField,
TDateField, TTimeField, TDateTimeField, TBytesField, TVarBytesField, TAutoIncField,
TBlobField, TMemoField, TGraphicField, TBlobField, TBlobField, TBlobField, TBlobField, nil,
TStringField, nil, TLargeIntField, TADTField, TArrayField, TReferenceField, TDataSetField,
TBlobField, TMemoField, TVariantField, TInterfaceField, TIDispatchField, TGuidField,
TSQLTimeStampField, TFMTBcdField, nil, nil, nil, nil);
```

File

IBCustomDataSet

Description

This is constant IBCustomDataSet.DefaultFieldClasses.

19.5.3 IBCustomDataSet.UniCache Constant

Pascal

```
UniCache = 2;
```

C++

```
UniCache = 2;
```

File

IBCustomDataSet

Description

This is constant IBCustomDataSet.UniCache.

20 IBDatabase Namespace

This is namespace IBDatabase.

Topics

Name	Description
Classes (see page 3705)	The following table lists classes in this documentation.
Interfaces (see page 3761)	The following table lists interfaces in this documentation.
Functions (see page 3764)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3765)	The following table lists structs, records, enums in this documentation.
Types (see page 3767)	The following table lists types in this documentation.
Constants (see page 3768)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIBBase (see page 3705)	TIBBase is the ancestor object from which TIBTransaction (see page 3746) descends.
	TIBDataBase (see page 3716)	This is class IBDatabase.TIBDataBase.
	TIBSchema (see page 3741)	This is class IBDatabase.TIBSchema.
	TIBTimer (see page 3742)	This is class IBDatabase.TIBTimer.
	TIBTransaction (see page 3746)	TIBTransaction provides discrete transaction control over a one or more database connections in a database application.

Constants

Name	Description
DPBConstantNames (see page 3768)	This is constant IBDatabase.DPBConstantNames.
DPBPrefix (see page 3769)	This is constant IBDatabase.DPBPrefix.
TPBConstantNames (see page 3769)	This is constant IBDatabase.TPBConstantNames.
TPBPrefix (see page 3769)	This is constant IBDatabase.TPBPrefix.

Functions

	Name	Description
	GenerateDPB (see page 3764)	This is function IBDatabase.GenerateDPB.
	GenerateTPB (see page 3764)	This is function IBDatabase.GenerateTPB.

Interfaces

	Name	Description
	IIBEventNotifier (see page 3761)	IIBEventNotifier allows a TIBDatabase component to automatically register interest in events on the InterBase server.

Structs, Records, Enums

	Name	Description
	TAutoStopAction (see page 3765)	TAutoStopAction indicates the action to take on automatically ended transactions.
	TTransactionAction (see page 3766)	TTransactionAction indicates the action a transaction can take when it times out.

Types

Name	Description
TIBDatabaseLoginEvent (see page 3767)	TIBDatabaseLoginEvent is the type of event handlers that respond when an IB (see page 3541) database component forms a connection to an InterBase server.
TIBFileName (see page 3767)	This is type IBDatabase.TIBFileName.

20.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBBase (see page 3705)	TIBBase is the ancestor object from which TIBTransaction (see page 3746) descends.
	TIBDataBase (see page 3716)	This is class IBDatabase.TIBDataBase.
	TIBSchema (see page 3741)	This is class IBDatabase.TIBSchema.
	TIBTimer (see page 3742)	This is class IBDatabase.TIBTimer.
	TIBTransaction (see page 3746)	TIBTransaction provides discrete transaction control over a one or more database connections in a database application.

20.1.1 TIBBase Class

TIBBase is the ancestor object from which TIBTransaction (see page 3746) descends.

Class Hierarchy

Pascal

```
TIBBase = class(TObject);
```

C++

```
class TIBBase : public TObject;
```

File

IBDatabase

Description

TIBBase is the ancestor object from which TIBTransaction (see page 3746) descends.

20.1.1.1 TIBBase Members

The following tables list the members exposed by TIBBase.

Protected Fields

	Name	Description
	FAfterDatabaseDisconnect (see page 3707)	This is FAfterDatabaseDisconnect, a member of class TIBBase.
	FAfterTransactionEnd (see page 3707)	This is FAfterTransactionEnd, a member of class TIBBase.
	FBeforeDatabaseDisconnect (see page 3708)	This is FBeforeDatabaseDisconnect, a member of class TIBBase.
	FBeforeTransactionEnd (see page 3708)	This is FBeforeTransactionEnd, a member of class TIBBase.
	FDatabase (see page 3708)	This is FDatabase, a member of class TIBBase.
	FIndexInDatabase (see page 3708)	This is FIndexInDatabase, a member of class TIBBase.
	FIndexInTransaction (see page 3709)	This is FIndexInTransaction, a member of class TIBBase.
	FOnDatabaseFree (see page 3709)	This is FOnDatabaseFree, a member of class TIBBase.

	FOnTransactionFree (see page 3709)	This is FOnTransactionFree, a member of class TIBBase.
	FOwner (see page 3709)	This is FOwner, a member of class TIBBase.
	FTransaction (see page 3709)	This is FTransaction, a member of class TIBBase.

Protected Methods

	Name	Description
	DoAfterDatabaseDisconnect (see page 3711)	This is DoAfterDatabaseDisconnect, a member of class TIBBase.
	DoAfterTransactionEnd (see page 3711)	This is DoAfterTransactionEnd, a member of class TIBBase.
	DoBeforeDatabaseDisconnect (see page 3711)	This is DoBeforeDatabaseDisconnect, a member of class TIBBase.
	DoBeforeTransactionEnd (see page 3711)	This is DoBeforeTransactionEnd, a member of class TIBBase.
	DoDatabaseFree (see page 3711)	This is DoDatabaseFree, a member of class TIBBase.
	DoTransactionFree (see page 3712)	This is DoTransactionFree, a member of class TIBBase.
	GetDBHandle (see page 3712)	This is GetDBHandle, a member of class TIBBase.
	GetTRHandle (see page 3712)	This is GetTRHandle, a member of class TIBBase.
	SetDatabase (see page 3712)	This is SetDatabase, a member of class TIBBase.
	SetTransaction (see page 3713)	This is SetTransaction, a member of class TIBBase.

Public Events

	Name	Description
	AfterDatabaseDisconnect (see page 3715)	Occurs after a database is disconnected.
	AfterTransactionEnd (see page 3715)	Occurs after a transaction has ended.
	BeforeDatabaseDisconnect (see page 3715)	Occurs before a database disconnects.
	BeforeTransactionEnd (see page 3715)	Occurs before a transaction ends.
	OnDatabaseFree (see page 3716)	Occurs after a database is freed from memory.
	OnTransactionFree (see page 3716)	Occurs after a transaction has been freed from memory.

Public Methods

	Name	Description
	CheckDatabase (see page 3710)	Checks if the database is active.
	CheckTransaction (see page 3710)	Checks if the transaction is active.
	Create (see page 3706)	Creates an instance of an IBBase object.
	Destroy (see page 3707)	Destroys an IBBase object.

Public Properties

	Name	Description
	Database (see page 3713)	Sets or returns the associated database.
	DBHandle (see page 3713)	Indicates the database handle.
	Owner (see page 3714)	Indicates which component owns the component.
	Transaction (see page 3714)	Sets or returns the associated transaction.
	TRHandle (see page 3714)	Returns the transaction handle.

20.1.1.2 TIBBase.Create Constructor

Creates an instance of an IBBase object.

Pascal

```
constructor Create(AOwner: TObject);
```

C++

```
__fastcall TIBBase(TObject * AOwner);
```

Description

Call Create to create an instance of an IBBase object.

20.1.1.3 TIBBase.Destroy Destructor

Destroys an IBBase object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBBase();
```

Description

Call Destroy to destroy an IBBase object and to free up any resources associated with it.

20.1.1.4 TIBBase Fields

The fields of the TIBBase class are listed here.

Protected Fields

	Name	Description
	FAfterDatabaseDisconnect (see page 3707)	This is FAfterDatabaseDisconnect, a member of class TIBBase.
	FAfterTransactionEnd (see page 3707)	This is FAfterTransactionEnd, a member of class TIBBase.
	FBeforeDatabaseDisconnect (see page 3708)	This is FBeforeDatabaseDisconnect, a member of class TIBBase.
	FBeforeTransactionEnd (see page 3708)	This is FBeforeTransactionEnd, a member of class TIBBase.
	FDatabase (see page 3708)	This is FDatabase, a member of class TIBBase.
	FIndexInDatabase (see page 3708)	This is FIndexInDatabase, a member of class TIBBase.
	FIndexInTransaction (see page 3709)	This is FIndexInTransaction, a member of class TIBBase.
	FOnDatabaseFree (see page 3709)	This is FOnDatabaseFree, a member of class TIBBase.
	FOnTransactionFree (see page 3709)	This is FOnTransactionFree, a member of class TIBBase.
	FOwner (see page 3709)	This is FOwner, a member of class TIBBase.
	FTransaction (see page 3709)	This is FTransaction, a member of class TIBBase.

20.1.1.4.1 TIBBase.FAfterDatabaseDisconnect Field

Pascal

```
FAfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
TNotifyEvent FAfterDatabaseDisconnect;
```

Description

This is FAfterDatabaseDisconnect, a member of class TIBBase.

20.1.1.4.2 TIBBase.FAfterTransactionEnd Field

Pascal

```
FAfterTransactionEnd: TNotifyEvent;
```

C++

```
TNotifyEvent FAfterTransactionEnd;
```

Description

This is FAfterTransactionEnd, a member of class TIBBase.

20.1.1.4.3 TIBBase.FBeforeDatabaseDisconnect Field

Pascal

```
FBeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
TNotifyEvent FBeforeDatabaseDisconnect;
```

Description

This is FBeforeDatabaseDisconnect, a member of class TIBBase.

20.1.1.4.4 TIBBase.FBeforeTransactionEnd Field

Pascal

```
FBeforeTransactionEnd: TNotifyEvent;
```

C++

```
TNotifyEvent FBeforeTransactionEnd;
```

Description

This is FBeforeTransactionEnd, a member of class TIBBase.

20.1.1.4.5 TIBBase.FDatabase Field

Pascal

```
FDatabase: TIBDatabase;
```

C++

```
TIBDatabase FDatabase;
```

Description

This is FDatabase, a member of class TIBBase.

20.1.1.4.6 TIBBase.FIndexInDatabase Field

Pascal

```
FIndexInDatabase: Integer;
```

C++

```
int FIndexInDatabase;
```

Description

This is FIndexInDatabase, a member of class TIBBase.

20.1.1.4.7 TIBBase.FIndexInTransaction Field

Pascal

```
FIndexInTransaction: Integer;
```

C++

```
int FIndexInTransaction;
```

Description

This is FIndexInTransaction, a member of class TIBBase.

20.1.1.4.8 TIBBase.FOnDatabaseFree Field

Pascal

```
FOnDatabaseFree: TNotifyEvent;
```

C++

```
TNotifyEvent FOnDatabaseFree;
```

Description

This is FOnDatabaseFree, a member of class TIBBase.

20.1.1.4.9 TIBBase.FOnTransactionFree Field

Pascal

```
FOnTransactionFree: TNotifyEvent;
```

C++

```
TNotifyEvent FOnTransactionFree;
```

Description

This is FOnTransactionFree, a member of class TIBBase.

20.1.1.4.10 TIBBase.FOwner Field

Pascal

```
FOwner: TObject;
```

C++

```
TObject * FOwner;
```

Description

This is FOwner, a member of class TIBBase.

20.1.1.4.11 TIBBase.FTransaction Field

Pascal

```
FTransaction: TIBTransaction;
```

C++

```
TIBTransaction FTransaction;
```

Description

This is FTransaction, a member of class TIBBase.

20.1.1.5 TIBBase Methods

The methods of the TIBBase class are listed here.

Protected Methods

	Name	Description
	DoAfterDatabaseDisconnect (see page 3711)	This is DoAfterDatabaseDisconnect, a member of class TIBBase.
	DoAfterTransactionEnd (see page 3711)	This is DoAfterTransactionEnd, a member of class TIBBase.
	DoBeforeDatabaseDisconnect (see page 3711)	This is DoBeforeDatabaseDisconnect, a member of class TIBBase.
	DoBeforeTransactionEnd (see page 3711)	This is DoBeforeTransactionEnd, a member of class TIBBase.
	DoDatabaseFree (see page 3711)	This is DoDatabaseFree, a member of class TIBBase.
	DoTransactionFree (see page 3712)	This is DoTransactionFree, a member of class TIBBase.
	GetDBHandle (see page 3712)	This is GetDBHandle, a member of class TIBBase.
	GetTRHandle (see page 3712)	This is GetTRHandle, a member of class TIBBase.
	SetDatabase (see page 3712)	This is SetDatabase, a member of class TIBBase.
	SetTransaction (see page 3713)	This is SetTransaction, a member of class TIBBase.

Public Methods

	Name	Description
	CheckDatabase (see page 3710)	Checks if the database is active.
	CheckTransaction (see page 3710)	Checks if the transaction is active.

20.1.1.5.1 TIBBase.CheckDatabase Method

Checks if the database is active.

Pascal

```
procedure CheckDatabase; virtual;
```

C++

```
virtual __fastcall CheckDatabase();
```

Description

Call CheckDatabase to check if the database is assigned and active.

20.1.1.5.2 TIBBase.CheckTransaction Method

Checks if the transaction is active.

Pascal

```
procedure CheckTransaction; virtual;
```

C++

```
virtual __fastcall CheckTransaction();
```

Description

Call CheckTransaction to check if the transaction is assigned and active.

20.1.1.5.3 TIBBase.DoAfterDatabaseDisconnect Method

Pascal

```
procedure DoAfterDatabaseDisconnect; virtual;
```

C++

```
virtual __fastcall DoAfterDatabaseDisconnect();
```

Description

This is DoAfterDatabaseDisconnect, a member of class TIBBase.

20.1.1.5.4 TIBBase.DoAfterTransactionEnd Method

Pascal

```
procedure DoAfterTransactionEnd; virtual;
```

C++

```
virtual __fastcall DoAfterTransactionEnd();
```

Description

This is DoAfterTransactionEnd, a member of class TIBBase.

20.1.1.5.5 TIBBase.DoBeforeDatabaseDisconnect Method

Pascal

```
procedure DoBeforeDatabaseDisconnect; virtual;
```

C++

```
virtual __fastcall DoBeforeDatabaseDisconnect();
```

Description

This is DoBeforeDatabaseDisconnect, a member of class TIBBase.

20.1.1.5.6 TIBBase.DoBeforeTransactionEnd Method

Pascal

```
procedure DoBeforeTransactionEnd; virtual;
```

C++

```
virtual __fastcall DoBeforeTransactionEnd();
```

Description

This is DoBeforeTransactionEnd, a member of class TIBBase.

20.1.1.5.7 TIBBase.DoDatabaseFree Method

Pascal

```
procedure DoDatabaseFree; virtual;
```

C++

```
virtual __fastcall DoDatabaseFree();
```

Description

This is DoDatabaseFree, a member of class TIBBase.

20.1.1.5.8 TIBBase.DoTransactionFree Method

Pascal

```
procedure DoTransactionFree; virtual;
```

C++

```
virtual __fastcall DoTransactionFree();
```

Description

This is DoTransactionFree, a member of class TIBBase.

20.1.1.5.9 TIBBase.GetDBHandle Method

Pascal

```
function GetDBHandle: PISC_DB_HANDLE; virtual;
```

C++

```
virtual __fastcall PISC_DB_HANDLE GetDBHandle();
```

Description

This is GetDBHandle, a member of class TIBBase.

20.1.1.5.10 TIBBase.GetTRHandle Method

Pascal

```
function GetTRHandle: PISC_TR_HANDLE; virtual;
```

C++

```
virtual __fastcall PISC_TR_HANDLE GetTRHandle();
```

Description

This is GetTRHandle, a member of class TIBBase.

20.1.1.5.11 TIBBase.SetDatabase Method

Pascal

```
procedure SetDatabase(Value: TIBDatabase); virtual;
```

C++

```
virtual __fastcall SetDatabase(TIBDatabase Value);
```

Description

This is SetDatabase, a member of class TIBBase.

20.1.1.5.12 TIBBase.SetTransaction Method

Pascal

```
procedure SetTransaction(Value: TIBTransaction); virtual;
```

C++

```
virtual __fastcall SetTransaction(TIBTransaction Value);
```

Description

This is SetTransaction, a member of class TIBBase.

20.1.1.6 TIBBase Properties

The properties of the TIBBase class are listed here.

Public Properties

	Name	Description
	Database (see page 3713)	Sets or returns the associated database.
R	DBHandle (see page 3713)	Indicates the database handle.
R	Owner (see page 3714)	Indicates which component owns the component.
	Transaction (see page 3714)	Sets or returns the associated transaction.
R	TRHandle (see page 3714)	Returns the transaction handle.

20.1.1.6.1 TIBBase.Database Property

Sets or returns the associated database.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to set or return the associated database.

20.1.1.6.2 TIBBase.DBHandle Property

Indicates the database handle.

Pascal

```
property DBHandle: PISC_DB_HANDLE;
```

C++

```
__property PISC_DB_HANDLE DBHandle;
```

Description

Use DBHandle to return the database handle.

20.1.1.6.3 TIBBase.Owner Property

Indicates which component owns the component.

Pascal

```
property Owner: TObject;
```

C++

```
__property TObject * Owner;
```

Description

Owner refers to the SQL object (DataSet, TIBSQL, or Blob) that created the TIBBase (see page 3705) component.

20.1.1.6.4 TIBBase.Transaction Property

Sets or returns the associated transaction.

Pascal

```
property Transaction: TIBTransaction;
```

C++

```
__property TIBTransaction Transaction;
```

Description

Use Transaction to set or return the associated transaction.

20.1.1.6.5 TIBBase.TRHandle Property

Returns the transaction handle.

Pascal

```
property TRHandle: PISC_TR_HANDLE;
```

C++

```
__property PISC_TR_HANDLE TRHandle;
```

Description

Use TRHandle to return the transaction handle.

20.1.1.7 TIBBase Events

The events of the TIBBase class are listed here.

Public Events

	Name	Description
	AfterDatabaseDisconnect (see page 3715)	Occurs after a database is disconnected.
	AfterTransactionEnd (see page 3715)	Occurs after a transaction has ended.
	BeforeDatabaseDisconnect (see page 3715)	Occurs before a database disconnects.
	BeforeTransactionEnd (see page 3715)	Occurs before a transaction ends.
	OnDatabaseFree (see page 3716)	Occurs after a database is freed from memory.
	OnTransactionFree (see page 3716)	Occurs after a transaction has been freed from memory.

20.1.1.7.1 TIBBase.AfterDatabaseDisconnect Event

Occurs after a database is disconnected.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to take specific actions after a database is disconnected.

20.1.1.7.2 TIBBase.AfterTransactionEnd Event

Occurs after a transaction has ended.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to take specific actions after a transaction has ended.

20.1.1.7.3 TIBBase.BeforeDatabaseDisconnect Event

Occurs before a database disconnects.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to take specific actions before a database disconnects.

20.1.1.7.4 TIBBase.BeforeTransactionEnd Event

Occurs before a transaction ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to take specific actions before a transaction ends.

20.1.1.7.5 TIBBase.OnDatabaseFree Event

Occurs after a database is freed from memory.

Pascal

```
property OnDatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDatabaseFree;
```

Description

Write an OnDatabaseFree event handler to take specific actions after a database is freed from memory.

20.1.1.7.6 TIBBase.OnTransactionFree Event

Occurs after a transaction has been freed from memory.

Pascal

```
property OnTransactionFree: TNotifyEvent;
```

C++

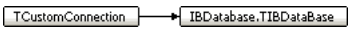
```
__property TNotifyEvent OnTransactionFree;
```

Description

Write an OnTransactionFree event handler to take specific actions after a transaction has been freed from memory.

20.1.2 TIBDataBase Class

Class Hierarchy



Pascal

```
TIBDataBase = class(TCustomConnection);
```

C++

```
class TIBDataBase : public TCustomConnection;
```

File

IBDatabase

Description

This is class IBDatabase.TIBDataBase.

20.1.2.1 TIBDataBase Members

The following tables list the members exposed by TIBDataBase.

Protected Methods

	Name	Description
	DoConnect (see page 3722)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 3723)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 3724)	Returns the value of the Connected property.
	Loaded (see page 3726)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 3726)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	AddEventNotifier (see page 3720)	Adds an event notifier to the database's internal list so that the database can handle registering and unregistering interest in events.
	AddTransaction (see page 3720)	Adds an association between the transaction component and the database component.
	ApplyUpdates (see page 3720)	Posts pending cached updates for specified datasets to the database server.
	Call (see page 3721)	Returns an error message based on the error code.
	CheckActive (see page 3721)	Checks to see if the database connection is active.
	CheckDatabaseName (see page 3721)	Checks to see if the DatabaseName (see page 3730) property is not empty.
	CheckInactive (see page 3721)	Checks to see if the database connection is inactive.
	CloseDataSets (see page 3722)	Closes all datasets associated with the database component without disconnecting from the database server.
	Create (see page 3718)	Creates an instance of a TIBDatabase component.
	CreateDatabase (see page 3722)	Creates a database using Params (see page 3736).
	Destroy (see page 3718)	Destroys the instance of the database component.
	DropDatabase (see page 3723)	Drops a database.
	FindDefaultTransaction (see page 3723)	Returns the default transaction.
	FindTransaction (see page 3724)	Finds the index of a transaction.
	FlushSchema (see page 3724)	Clears the schema cache for the database.
	ForceClose (see page 3724)	Forces the database connection to close.
	GetFieldNames (see page 3725)	Populates a list with the names of the fields in the table.
	GetTableNames (see page 3725)	Populates a string list with the names of tables in the database.
	Has_COMPUTED_BLR (see page 3725)	Indicates whether a specified field in a specified relation includes a COMPUTED_BLR attribute.
	Has_DEFAULT_VALUE (see page 3726)	Indicates whether a specified field in a specified relation includes a DEFAULT_VALUE attribute.
	IndexOfDBConst (see page 3726)	Searches for the named parameter in the database parameters list.
	RemoveEventNotifier (see page 3727)	Removes an event notifier from the database's internal list so that the database no longer handles registering and unregistering interest in events.
	RemoveTransaction (see page 3727)	Disassociates a transaction from the database.
	RemoveTransactions (see page 3728)	Disassociates all transactions from the database.
	SetHandle (see page 3728)	Sets the handle for the database.
	TestConnected (see page 3728)	Tests whether a database is connected.

Public Properties

	Name	Description
	DBParamByDPB (see page 3730)	Specifies the name of the database to associate with this database component.
	Handle (see page 3731)	Specifies the InterBase API database handle.
	HandleIsShared (see page 3731)	Indicates whether or not a the handle is shared.
	InternalTransaction (see page 3732)	Used internally by InterBase transactions.
	IsReadOnly (see page 3732)	Indicates whether or not the database is set to read-only.
	SQLObjectCount (see page 3737)	Returns the number of SQL objects.
	SQLObjects (see page 3737)	Returns an SQL object.
	TransactionCount (see page 3738)	Returns the number of transactions associated with the TIBDatabase component.
	Transactions (see page 3738)	Specifies a transaction for the given index.

Published Events

	Name	Description
	AfterConnect (see page 3738)	Occurs after a connection is established.
	AfterDisconnect (see page 3739)	Occurs after the connection closes.
	BeforeConnect (see page 3739)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 3739)	Occurs immediately before the connection closes.
	OnDialectDowngradeWarning (see page 3740)	Occurs after the SQL dialect of the client connection is downgraded.
	OnIdleTimer (see page 3740)	Occurs after a database connection times out.
	OnLogin (see page 3741)	Occurs when an application connects to a database.

Published Properties

	Name	Description
	AllowStreamedConnected (see page 3729)	Indicates whether the Connected (see page 3729) property can be set at design time.
	Connected (see page 3729)	Indicates whether or not a database connection is active.
	DatabaseName (see page 3730)	Specifies the name of the database to associate with this database component.
R	DBSQLDialect (see page 3730)	Returns the database SQL dialect.
	DefaultTransaction (see page 3731)	Sets or returns the default database transaction.
	IdleTimer (see page 3731)	Specifies how long the database should wait before disconnecting an idle connection.
	LoginPrompt (see page 3732)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 3736)	Specifies the database parameters to pass to the InterBase server.
	SQLDialect (see page 3736)	Sets or returns the SQL dialect used by the client.
	TraceFlags (see page 3737)	Specifies the database operations to track with the SQL Monitor at runtime.

20.1.2.2 TIBDataBase.Create Constructor

Creates an instance of a TIBDatabase component.

Pascal

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

C++

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

Description

Call (see page 3721) Create to instantiate a database component at runtime. An application can create a database component in order to control the component's existence and set its properties and events.

Create instantiates a database component and creates an empty list of dataset components for the DataSets property and an empty string list for the Params (see page 3736) property.

20.1.2.3 TIBDataBase.Destroy Destructor

Destroys the instance of the database component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBDataBase();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the database component is not already freed before calling Destroy.

Destroy disconnects from the database server, if necessary. It then frees the string resources allocated for the Params (see page 3736) and DataSets properties before calling its inherited destructor.

20.1.2.4 TIBDataBase Methods

The methods of the TIBDataBase class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 3722)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 3723)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 3724)	Returns the value of the Connected property.
	Loaded (see page 3726)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 3726)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	AddEventNotifier (see page 3720)	Adds an event notifier to the database's internal list so that the database can handle registering and unregistering interest in events.
	AddTransaction (see page 3720)	Adds an association between the transaction component and the database component.
	ApplyUpdates (see page 3720)	Posts pending cached updates for specified datasets to the database server.
	Call (see page 3721)	Returns an error message based on the error code.
	CheckActive (see page 3721)	Checks to see if the database connection is active.
	CheckDatabaseName (see page 3721)	Checks to see if the DatabaseName (see page 3730) property is not empty.
	CheckInactive (see page 3721)	Checks to see if the database connection is inactive.
	CloseDataSets (see page 3722)	Closes all datasets associated with the database component without disconnecting from the database server.
	CreateDatabase (see page 3722)	Creates a database using Params (see page 3736).
	DropDatabase (see page 3723)	Drops a database.
	FindDefaultTransaction (see page 3723)	Returns the default transaction.
	FindTransaction (see page 3724)	Finds the index of a transaction.
	FlushSchema (see page 3724)	Clears the schema cache for the database.
	ForceClose (see page 3724)	Forces the database connection to close.
	GetFieldNames (see page 3725)	Populates a list with the names of the fields in the table.
	GetTableNames (see page 3725)	Populates a string list with the names of tables in the database.
	Has_COMPUTED_BLR (see page 3725)	Indicates whether a specified field in a specified relation includes a COMPUTED_BLR attribute.
	Has_DEFAULT_VALUE (see page 3726)	Indicates whether a specified field in a specified relation includes a DEFAULT_VALUE attribute.
	IndexOfDBConst (see page 3726)	Searches for the named parameter in the database parameters list.
	RemoveEventNotifier (see page 3727)	Removes an event notifier from the database's internal list so that the database no longer handles registering and unregistering interest in events.
	RemoveTransaction (see page 3727)	Disassociates a transaction from the database.
	RemoveTransactions (see page 3728)	Disassociates all transactions from the database.
	SetHandle (see page 3728)	Sets the handle for the database.
	TestConnected (see page 3728)	Tests whether a database is connected.

20.1.2.4.1 TIBDataBase.AddEventNotifier Method

Adds an event notifier to the database's internal list so that the database can handle registering and unregistering interest in events.

Pascal

```
procedure AddEventNotifier(Notifier: IIBEventNotifier);
```

C++

```
__fastcall AddEventNotifier(IIBEventNotifier Notifier);
```

Description

Call (see page 3721) AddEventNotifier when you want the database to automatically register and unregister an interest in server events when it opens and closes a connection to the server.

Notifier is the interface to an object that has in interest in a set of events on the InterBase server. For example, Notifier can be the interface to a TIBEvents object.

AddEventNotifier adds Notifier to its internal list of event notifiers. When TIBDatabase opens a connection, it checks every notifier in its list, and if a notifier indicates that it wants to be auto-registered, AddEventNotifier calls its RegisterEvents method so that the notifier registers its interest with the InterBase Event manager. Before TIBDatabase closes a connection, it unregisters interest from every notifier in its list.

Note: If Notifier is a TIBEvents object, AddEventNotifier is automatically called when you set the Database property.

20.1.2.4.2 TIBDataBase.AddTransaction Method

Adds an association between the transaction component and the database component.

Pascal

```
function AddTransaction(TR: TIBTransaction): Integer;
```

C++

```
__fastcall int AddTransaction(TIBTransaction TR);
```

Description

Call (see page 3721) AddTransaction to add an association between the transaction component and the database component.

20.1.2.4.3 TIBDataBase.ApplyUpdates Method

Posts pending cached updates for specified datasets to the database server.

Pascal

```
procedure ApplyUpdates(const DataSets: array of TDataSet);
```

C++

```
__fastcall ApplyUpdates(const array of TDataSet DataSets);
```

Description

Call (see page 3721) ApplyUpdates to post pending cached updates for a specific set of open datasets to the database server. ApplyUpdates is only meaningful if the CachedUpdates property of a specified dataset is true.

DataSets is a list of dataset names specifying the datasets for which to post pending updates. DataSets need not list every currently open dataset. For each listed dataset ApplyUpdates calls the dataset's ApplyUpdates and CommitUpdates methods to

post that dataset's pending cached updates.

20.1.2.4.4 TIBDataBase.Call Method

Returns an error message based on the error code.

Pascal

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

C++

```
__fastcall ISC_STATUS Call(ISC_STATUS ErrCode, Boolean RaiseError);
```

Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

20.1.2.4.5 TIBDataBase.CheckActive Method

Checks to see if the database connection is active.

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

Call (see page 3721) CheckActive to determine if the connection to a database server is active. If the database connection is inactive, an ibxeDatabaseClosed exception is raised.

20.1.2.4.6 TIBDataBase.CheckDatabaseName Method

Checks to see if the DatabaseName (see page 3730) property is not empty.

Pascal

```
procedure CheckDatabaseName;
```

C++

```
__fastcall CheckDatabaseName();
```

Description

Call (see page 3721) CheckDatabaseName to determine whether the DatabaseName (see page 3730) property is empty. If the DatabaseName (see page 3730) is empty, an ibxeDatabaseNameMissing exception is raised.

20.1.2.4.7 TIBDataBase.CheckInactive Method

Checks to see if the database connection is inactive.

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

Call (see page 3721) CheckInactive to determine whether the connection to a database server is inactive. If the database connection is active, an ibxeDatabaseOpen exception is raised.

20.1.2.4.8 TIBDataBase.CloseDataSets Method

Closes all datasets associated with the database component without disconnecting from the database server.

Pascal

```
procedure CloseDataSets;
```

C++

```
__fastcall CloseDataSets();
```

Description

Call (see page 3721) CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close, all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

20.1.2.4.9 TIBDataBase.CreateDatabase Method

Creates a database using Params (see page 3736).

Pascal

```
procedure CreateDatabase;
```

C++

```
__fastcall CreateDatabase();
```

Description

Call (see page 3721) CreateDatabase to create a database using Params (see page 3736) as the rest of the CREATE DATABASE command.

For example, if you wanted to create a local InterBase database, you could do the following:

1. Set the database name to the drive, path, and filename of the database file.
2. Set Params (see page 3736) to the parameter for the CREATE DATABASE statement: `USER "SYSDBA" PASSWORD "masterkey" PAGE_SIZE 4096`
3. Set the SQLDialect (see page 3736) value.
4. Call (see page 3721) the CreateDatabase method.

20.1.2.4.10 TIBDataBase.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

20.1.2.4.11 TIBDataBase.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

20.1.2.4.12 TIBDataBase.DropDatabase Method

Drops a database.

Pascal

```
procedure DropDatabase;
```

C++

```
__fastcall DropDatabase();
```

Description

Call (see page 3721) DropDatabase to drop a database, which removes the database file from the server.

20.1.2.4.13 TIBDataBase.FindDefaultTransaction Method

Returns the default transaction.

Pascal

```
function FindDefaultTransaction: TIBTransaction;
```

C++

```
__fastcall TIBTransaction FindDefaultTransaction();
```

Description

Call (see page 3721) FindDefaultTransaction to return the default transaction for the database.

20.1.2.4.14 TIBDataBase.FindTransaction Method

Finds the index of a transaction.

Pascal

```
function FindTransaction(TR: TIBTransaction): Integer;
```

C++

```
__fastcall int FindTransaction(TIBTransaction TR);
```

Description

Call (see page 3721) FindTransaction to find the index of a specified transaction.

20.1.2.4.15 TIBDataBase.FlushSchema Method

Clears the schema cache for the database.

Pascal

```
procedure FlushSchema;
```

C++

```
__fastcall FlushSchema();
```

Description

Call (see page 3721) FlushSchema to clear the schema cache.

20.1.2.4.16 TIBDataBase.ForceClose Method

Forces the database connection to close.

Pascal

```
procedure ForceClose;
```

C++

```
__fastcall ForceClose();
```

Description

Use ForceClose to force the database connection to close.

Note: Forcing a database to close attempts to close the connection to the server. Even if the call fails, the database handle is reset to nil (Delphi) or NULL (C++).

20.1.2.4.17 TIBDataBase.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to

return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

20.1.2.4.18 TIBDataBase.GetFieldNames Method

Populates a list with the names of the fields in the table.

Pascal

```
procedure GetFieldNames(const TableName: Widestring; List: TWideStrings);
```

C++

```
__fastcall GetFieldNames(const BSTR TableName, TWideStrings List);
```

Description

Call (see page 3721) GetFieldNames to retrieve a list of fields in the associated table.

20.1.2.4.19 TIBDataBase.GetTableNames Method

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TWideStrings; SystemTables: Boolean = False);
```

C++

```
__fastcall GetTableNames(TWideStrings List, Boolean SystemTables = False);
```

Description

Call (see page 3721) GetTableNames to retrieve a list of tables in the associated database.

List is the already-existing string list object into which the tables' names are put.

Set SystemTables to indicate whether the list of table names should include the database's system tables.

```
IBDatabase1.GetTableNames(ListBox2.Items, False);
    IBDatabase1->GetTableNames(ListBox2->Items, false);
```

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

20.1.2.4.20 TIBDataBase.Has_COMPUTED_BLR Method

Indicates whether a specified field in a specified relation includes a COMPUTED_BLR attribute.

Pascal

```
function Has_COMPUTED_BLR(Relation: String; Field: String): Boolean;
```

C++

```
__fastcall Boolean Has_COMPUTED_BLR(UnicodeString Relation, UnicodeString Field);
```

Description

Call (see page 3721) Has_COMPUTED_BLR to determine whether the Field in Relation includes a COMPUTED_BLR attribute.

20.1.2.4.21 TIBDataBase.Has_DEFAULT_VALUE Method

Indicates whether a specified field in a specified relation includes a DEFAULT_VALUE attribute.

Pascal

```
function Has_DEFAULT_VALUE(Relation: String; Field: String): Boolean;
```

C++

```
__fastcall Boolean Has_DEFAULT_VALUE(UnicodeString Relation, UnicodeString Field);
```

Description

Call (see page 3721) Has_DEFAULT_VALUE to determine whether Field has a default value when used in Relation.

20.1.2.4.22 TIBDataBase.IndexOfDBConst Method

Searches for the named parameter in the database parameters list.

Pascal

```
function IndexOfDBConst(st: String): Integer;
```

C++

```
__fastcall int IndexOfDBConst(UnicodeString st);
```

Description

Use IndexOfDBConst to locate a parameter in the database parameters list. If the parameter is not found, IndexOfDBConst returns -1.

20.1.2.4.23 TIBDataBase.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected

Loaded

20.1.2.4.24 TIBDataBase.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

20.1.2.4.25 TIBDataBase.RemoveEventNotifier Method

Removes an event notifier from the database's internal list so that the database no longer handles registering and unregistering interest in events.

Pascal

```
procedure RemoveEventNotifier(Notifier: IIBEventNotifier);
```

C++

```
__fastcall RemoveEventNotifier(IIBEventNotifier Notifier);
```

Description

Call (see page 3721) RemoveEventNotifier when you no longer want the database to automatically register and unregister an interest in server events when it opens and closes a connection to the server. For example, TIBEvents automatically calls RemoveEventNotifier when you change its Database property.

Notifier is the event notifier previously added to the database's internal list by a call to AddEventNotifier (see page 3720).

RemoveEventNotifier removes Notifier from its internal list of event notifiers.

Note: Removing an event notifier from the list does not first unregister an interest in any registered events.

20.1.2.4.26 TIBDataBase.RemoveTransaction Method

Disassociates a transaction from the database.

Pascal

```
procedure RemoveTransaction(Idx: Integer);
```

C++

```
__fastcall RemoveTransaction(int Idx);
```

Description

Call (see page 3721) RemoveTransaction to disassociate a specified transaction from the database.

20.1.2.4.27 TIBDataBase.RemoveTransactions Method

Disassociates all transactions from the database.

Pascal

```
procedure RemoveTransactions;
```

C++

```
__fastcall RemoveTransactions();
```

Description

Call (see page 3721) RemoveTransactions to disassociate all transactions from the database.

20.1.2.4.28 TIBDataBase.SetHandle Method

Sets the handle for the database.

Pascal

```
procedure SetHandle(Value: TISC_DB_HANDLE);
```

C++

```
__fastcall SetHandle(TISC_DB_HANDLE Value);
```

Description

Call (see page 3721) SetHandle to set the handle for the database.

20.1.2.4.29 TIBDataBase.TestConnected Method

Tests whether a database is connected.

Pascal

```
function TestConnected: Boolean;
```

C++

```
__fastcall Boolean TestConnected();
```

Description

Use TestConnected to determine whether a database is connected to the server. TestConnected returns true if the connection is good, and false if it is not.

20.1.2.5 TIBDataBase Properties

The properties of the TIBDataBase class are listed here.

Public Properties

	Name	Description
	DBParamByDPB (see page 3730)	Specifies the name of the database to associate with this database component.
R	Handle (see page 3731)	Specifies the InterBase API database handle.

	HandleIsShared (see page 3731)	Indicates whether or not a the handle is shared.
	InternalTransaction (see page 3732)	Used internally by InterBase transactions.
	IsReadOnly (see page 3732)	Indicates whether or not the database is set to read-only.
	SQLObjectCount (see page 3737)	Returns the number of SQL objects.
	SQLObjects (see page 3737)	Returns an SQL object.
	TransactionCount (see page 3738)	Returns the number of transactions associated with the TIBDatabase component.
	Transactions (see page 3738)	Specifies a transaction for the given index.

Published Properties

	Name	Description
	AllowStreamedConnected (see page 3729)	Indicates whether the Connected (see page 3729) property can be set at design time.
	Connected (see page 3729)	Indicates whether or not a database connection is active.
	DatabaseName (see page 3730)	Specifies the name of the database to associate with this database component.
	DBSQLDialect (see page 3730)	Returns the database SQL dialect.
	DefaultTransaction (see page 3731)	Sets or returns the default database transaction.
	IdleTimer (see page 3731)	Specifies how long the database should wait before disconnecting an idle connection.
	LoginPrompt (see page 3732)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 3736)	Specifies the database parameters to pass to the InterBase server.
	SQLDialect (see page 3736)	Sets or returns the SQL dialect used by the client.
	TraceFlags (see page 3737)	Specifies the database operations to track with the SQL Monitor at runtime.

20.1.2.5.1 TIBDataBase.AllowStreamedConnected Property

Indicates whether the Connected (see page 3729) property can be set at design time.

Pascal

```
property AllowStreamedConnected: Boolean;
```

C++

```
__property Boolean AllowStreamedConnected;
```

Description

When AllowStreamedConnected is true (the default), then setting the Connected (see page 3729) property to true at design time causes this component to automatically establish a database connection when the application starts up. When AllowStreamedConnected is false, the connection must be explicitly opened at runtime, either by setting the Connected (see page 3729) property to true, or by calling the Open method.

Set AllowStreamedConnected to false when using a local test database at design time and a different database at runtime. This allows the application to set the appropriate runtime connection information before opening a runtime connection, without having to explicitly shut down the database connection that is used at design time.

20.1.2.5.2 TIBDataBase.Connected Property

Indicates whether or not a database connection is active.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to true to establish a database connection without opening a dataset. Set Connected to false to close a database connection. An application can check Connected to determine the current status of a database connection. If Connected is true, the database connection is active; if false, then the connection is inactive.

20.1.2.5.3 TIBDataBase.DatabaseName Property

Specifies the name of the database to associate with this database component.

Pascal

```
property DatabaseName: TIBFileName;
```

C++

```
__property TIBFileName DatabaseName;
```

Description

Use DatabaseName to specify the name of the database to use with a database component. For local InterBase databases, this can be a filename.

To connect to an InterBase database on a remote server using TCP/IP the syntax is <server_name>:<filename>.

To connect to an InterBase database on a remote server using NetBEUI, the syntax is: \\<server_name>\<filename>.

To connect to an InterBase database on a remote server using SPX, the syntax is: <server_name>@<filename>.

20.1.2.5.4 TIBDataBase.DBParamByDPB Property

Specifies the name of the database to associate with this database component.

Pascal

```
property DBParamByDPB [const Idx: Integer]: String;
```

C++

```
__property UnicodeString DBParamByDPB[int const Idx];
```

Description

Use DBParamByDPB to inspect and set DPB parameters without looking at the Params (see page 3736) string list.

For example,

```
DBParamByDPB[isc_dpb_user_name]
```

can be used to set and inspect the user name.

20.1.2.5.5 TIBDataBase.DBSQLDialect Property

Returns the database SQL dialect.

Pascal

```
property DBSQLDialect: Integer;
```

C++

```
__property int DBSQLDialect;
```

Description

Use DBSQLDialect to get the database SQL dialect.

20.1.2.5.6 TIBDataBase.DefaultTransaction Property

Sets or returns the default database transaction.

Pascal

```
property DefaultTransaction: TIBTransaction;
```

C++

```
__property TIBTransaction DefaultTransaction;
```

Description

Use DefaultTransaction to set or return the default database transaction.

A single database connection can manage one or more transactions. DefaultTransaction is a convenient way to specify a default transaction to a database connection.

20.1.2.5.7 TIBDataBase.Handle Property

Specifies the InterBase API database handle.

Pascal

```
property Handle: TISC_DB_HANDLE;
```

C++

```
__property TISC_DB_HANDLE Handle;
```

Description

Use Handle to make calls directly to the InterBase API. Many of the InterBase API functions require a database handle as one of their arguments. Handle is assigned an initial value when a database is opened.

20.1.2.5.8 TIBDataBase.HandleIsShared Property

Indicates whether or not a the handle is shared.

Pascal

```
property HandleIsShared: Boolean;
```

C++

```
__property Boolean HandleIsShared;
```

Description

Read HandleIsShared to determine if the handle to the database is shared.

20.1.2.5.9 TIBDataBase.IdleTimer Property

Specifies how long the database should wait before disconnecting an idle connection.

Pascal

```
property IdleTimer: Integer;
```

C++

```
__property int IdleTimer;
```

Description

Use IdleTimer to indicate how long the database should wait before automatically terminating the connection.

20.1.2.5.10 TIBDataBase.InternalTransaction Property

Used internally by InterBase transactions.

Pascal

```
property InternalTransaction: TIBTransaction;
```

C++

```
__property TIBTransaction InternalTransaction;
```

Description

Do not use InternalTransaction directly. It is for internal use only.

20.1.2.5.11 TIBDataBase.IsReadOnly Property

Indicates whether or not the database is set to read-only.

Pascal

```
property IsReadOnly: Boolean;
```

C++

```
__property Boolean IsReadOnly;
```

Description

Read IsReadOnly to determine if the database is read-only.

Read-only databases are an InterBase 6 feature.

20.1.2.5.12 TIBDataBase.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt;
```

C++

```
__property LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If

there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

C++ Examples:

```
//
//This example demostrates the use of ADO for database conectivity.
//Example assumes that a TDBGrid is placed on the form.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Login details */
    String UserName = "db_user_name";
    String PassWord = "db_pass_word";
    String Server = "my.db.server";

    /* Connection String */
    String ConnString =
        "Provider=SQLOLEDB.1;Persist Security Info=False;";
    ConnString +=
        "User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;";
    ConnString +=
        "Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;";
    ConnString +=
        "Tag with column collation when possible=False";

    /* SQL Query */
    String SQLStr = "SELECT * FROM customer WHERE customer_id = :AnId;";

    /* All ADO variables */
    TADOConnection* ADOConn;
    TADOQuery* ADOQuery;
    TDataSource* DataSrc;
    TParameter* Param;

    /* Create an ADO connection */
    ADOConn = new TADOConnection(this);

    /* Setup the connection string */
    ADOConn->ConnectionString = Format(ConnString,
```

```
        ARRAYOFCONST((UserName, PassWord, Server)));

    /* Disable login prompt */
    ADOConn->LoginPrompt = False;

    try
    {
        ADOConn->Connected = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the query */
    ADOQuery = new TADOQuery(this);
    ADOQuery->Connection = ADOConn;
    ADOQuery->SQL->Add(SQLStr);

    /* Update the parameter that was parsed from the SQL query: AnId */
    Param = ADOQuery->Parameters->ParamByName("AnId");
    Param->DataType = ftInteger;
    Param->Value = 1;

    /* Set the query to Prepared - will improve performance */
    ADOQuery->Prepared = true;

    try
    {
        ADOQuery->Active = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the data source */
    DataSrc = new TDataSource(this);
    DataSrc->DataSet = ADOQuery;
    DataSrc->Enabled = true;

    /* Finally initilalize the grid */
    DBGrid1->DataSource = DataSrc;
}
```

Delphi Examples:

```
{
This example demostrates the use of ADO for database conectivity.
Example assumes that a TDBGrid is placed on the form.
}
procedure TForm2.FormCreate(Sender: TObject);
const
  { Connection string }
  ConnString =
  'Provider=SQLOLEDB.1;Persist Security Info=False;' +
  'User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;' +
  'Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;'+
  'Tag with column collation when possible=False';

  { SQL Query }
  SQLStr = 'SELECT * FROM customer WHERE customer_id = :AnId;';
```

```
    { User access }
    UserName = 'db_user_name';
    PassWord = 'db_pass_word';
    Server = 'my.db.server';

  var
    ADOConn  : TADOConnection;
    ADOQuery : TADOQuery;
    DataSrc  : TDataSource;
    Param    : TParameter;

  begin
    { Create an ADO connection }
    ADOConn := TADOConnection.Create(Self);
    { Setup the provider engine }

    { Setup the connection string }
    ADOConn.ConnectionString := Format(ConnString,
      [UserName, PassWord, Server]);

    { Disable login prompt }
    ADOConn.LoginPrompt := False;

    try
      ADOConn.Connected := True;
    except
      on e: EADOError do
      begin
        MessageDlg('Error while connecting', mtError,
                   [mbOK], 0);

        Exit;
      end;
    end;

    { Create the query }
    ADOQuery := TADOQuery.Create(Self);
    ADOQuery.Connection := ADOConn;
    ADOQuery.SQL.Add(SQLStr);

    { Update the parameter that was parsed from the SQL query: AnId }
    Param := ADOQuery.Parameters.ParamByName('AnId');
    Param.DataType := ftInteger;
    Param.Value := 1;

    { Set the query to Prepared - will improve performance }
    ADOQuery.Prepared := true;

    try
      ADOQuery.Active := True;
    except
      on e: EADOError do
      begin
        MessageDlg('Error while doing query', mtError,
                   [mbOK], 0);

        Exit;
      end;
    end;

    { Create the data source }
    DataSrc := TDataSource.Create(Self);
    DataSrc.DataSet := ADOQuery;
    DataSrc.Enabled := true;

    { Finally initilalize the grid }
```

```
  DBGrid1.DataSource := DataSrc;
end;
```

See Also

OnLogin

BeforeConnect

AfterConnect

OnGetUserName

OnLogin

OnLogin

Controlling Server Login

20.1.2.5.13 TIBDataBase.Params Property

Specifies the database parameters to pass to the InterBase server.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Use Params to specify the database parameters to pass to the InterBase server.

Database parameters are passed to the server as text in order to establish the connection. For example:

```
user_name=sysdba
    password=masterkey
    sql_role_name=finance
    lc_ctype=WIN1252
```

For more information on character sets, refer to 'Character Sets and Collation Orders' in the InterBase Language Reference.

For other information, refer to the InterBase API Guide.

20.1.2.5.14 TIBDataBase.SQLDialect Property

Sets or returns the SQL dialect used by the client.

Pascal

```
property SQLDialect: Integer;
```

C++

```
__property int SQLDialect;
```

Description

Use the SQLDialect property to set or return the SQL dialect used by the client. If the connection is active, the SQLDialect property cannot be set to a value greater than the database SQL dialect. If the connection is inactive, then on connect an OnDialectDowngradeWarning (see page 3740) event may be fired if the SQLDialect is greater than the database SQL dialect. In such a case, the SQLDialect property will be downgraded to match the database SQL dialect.

20.1.2.5.15 TIBDataBase.SQLObjectCount Property

Returns the number of SQL objects.

Pascal

```
property SQLObjectCount: Integer;
```

C++

```
__property int SQLObjectCount;
```

Description

Use the SQLObjectCount property to return the number of SQL objects in the database.

SQL objects are usually defined as InterBase datasets, IBSQL (see page 4035), and Blobs.

20.1.2.5.16 TIBDataBase.SQLObjects Property

Returns an SQL object.

Pascal

```
property SQLObjects [Index: Integer]: TIBBase;
```

C++

```
__property TIBBase SQLObjects[int Index];
```

Description

Use the SQLObjects property to return an SQL object based on its numeric index.

SQL objects are usually defined as InterBase datasets, IBSQL (see page 4035), and Blobs.

20.1.2.5.17 TIBDataBase.TraceFlags Property

Specifies the database operations to track with the SQL Monitor at runtime.

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

Note: Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

The value of a session component's TraceFlags property determines the initial settings of the TraceFlags property for database components associated with the session.

Because TraceFlags is a set property, an application can specify different combinations of flags to monitor different combinations of statements. For example, the following statement limits monitoring to database connections and SQL statement preparation:

```
TraceFlags := [tfConnect, tfQPrepare];
    TraceFlags = [tfConnect, tfQPrepare];
```

20.1.2.5.18 TIBDataBase.TransactionCount Property

Returns the number of transactions associated with the TIBDatabase component.

Pascal

```
property TransactionCount: Integer;
```

C++

```
__property int TransactionCount;
```

Description

Use TransactionCount to return how many transactions are currently associated with the InterBase database component.

20.1.2.5.19 TIBDataBase.Transactions Property

Specifies a transaction for the given index.

Pascal

```
property Transactions [Index: Integer]: TIBTransaction;
```

C++

```
__property TIBTransaction Transactions[int Index];
```

Description

Given an integer index, Transactions returns the transaction at the given index. This is used internally for broadcasting important messages to attached components.

20.1.2.6 TIBDataBase Events

The events of the TIBDataBase class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 3738)	Occurs after a connection is established.
	AfterDisconnect (see page 3739)	Occurs after the connection closes.
	BeforeConnect (see page 3739)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 3739)	Occurs immediately before the connection closes.
	OnDialectDowngradeWarning (see page 3740)	Occurs after the SQL dialect of the client connection is downgraded.
	OnIdleTimer (see page 3740)	Occurs after a database connection times out.
	OnLogin (see page 3741)	Occurs when an application connects to a database.

20.1.2.6.1 TIBDataBase.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

20.1.2.6.2 TIBDataBase.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

20.1.2.6.3 TIBDataBase.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

20.1.2.6.4 TIBDataBase.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

20.1.2.6.5 TIBDataBase.OnDialectDowngradeWarning Event

Occurs after the SQL dialect of the client connection is downgraded.

Pascal

```
property OnDialectDowngradeWarning: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialectDowngradeWarning;
```

Description

Write an OnDialectDowngradeWarning event handler to take specific actions when the SQL dialect is downgraded.

For example, if the SQL dialect for your application is set to 3 and then a connection is made to a dialect 1 database, then the SQL dialect is downgraded to 1 and a OnDialectDowngradeWarning event is fired.

20.1.2.6.6 TIBDataBase.OnIdleTimer Event

Occurs after a database connection times out.

Pascal

```
property OnIdleTimer: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnIdleTimer;
```

Description

Write an OnIdleTimer event handler to take specific actions when the connection times out in the time specified by IdleTimer (see page 3731).

20.1.2.6.7 TIBDataBase.OnLogin Event

Occurs when an application connects to a database.

Pascal

```
property OnLogin: TIBDatabaseLoginEvent;
```

C++

```
__property TIBDatabaseLoginEvent OnLogin;
```

Description

Write an OnLogin event handler to take specific actions when an application attempts to connect to a database. By default, a database login is required. The current USER_NAME is read from the Params (see page 3736) property, and a standard Login dialog box opens. The dialog prompts for a user name and password combination, and then uses the values entered by the user to set the USER_NAME and PASSWORD values in the Params (see page 3736) property. These values are then passed to the remote server.

Applications that provide alternative OnLogin event handlers must set the USER_NAME and PASSWORD values in LoginParams parameter of the event handler.

Note: OnLogin does not occur unless the LoginPrompt (see page 3732) property is set to true.

20.1.3 TIBSchema Class

Class Hierarchy

Pascal

```
TIBSchema = class(TObject);
```

C++

```
class TIBSchema : public TObject;
```

File

IBDatabase

Description

This is class IBDatabase.TIBSchema.

20.1.3.1 TIBSchema Members

The following tables list the members exposed by TIBSchema.

Public Methods

	Name	Description
	FreeNodes (see page 3742)	This is FreeNodes, a member of class TIBSchema.
	Has_COMPUTED_BLR (see page 3742)	This is Has_COMPUTED_BLR, a member of class TIBSchema.
	Has_DEFAULT_VALUE (see page 3742)	This is Has_DEFAULT_VALUE, a member of class TIBSchema.

20.1.3.2 TIBSchema Methods

The methods of the TIBSchema class are listed here.

Public Methods

	Name	Description
	FreeNodes (see page 3742)	This is FreeNodes, a member of class TIBSchema.
	Has_COMPUTED_BLR (see page 3742)	This is Has_COMPUTED_BLR, a member of class TIBSchema.
	Has_DEFAULT_VALUE (see page 3742)	This is Has_DEFAULT_VALUE, a member of class TIBSchema.

20.1.3.2.1 TIBSchema.FreeNodes Method

Pascal

```
procedure FreeNodes; virtual; abstract;
```

C++

```
virtual __fastcall FreeNodes() = 0;
```

Description

This is FreeNodes, a member of class TIBSchema.

20.1.3.2.2 TIBSchema.Has_COMPUTED_BLR Method

Pascal

```
function Has_COMPUTED_BLR(Relation: String; Field: String): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean Has_COMPUTED_BLR(UnicodeString Relation, UnicodeString Field) = 0;
```

Description

This is Has_COMPUTED_BLR, a member of class TIBSchema.

20.1.3.2.3 TIBSchema.Has_DEFAULT_VALUE Method

Pascal

```
function Has_DEFAULT_VALUE(Relation: String; Field: String): Boolean; virtual; abstract;
```

C++

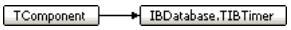
```
virtual __fastcall Boolean Has_DEFAULT_VALUE(UnicodeString Relation, UnicodeString Field) = 0;
```

Description

This is Has_DEFAULT_VALUE, a member of class TIBSchema.

20.1.4 TIBTimer Class

Class Hierarchy



Pascal

```
TIBTimer = class(TComponent);
```

C++

```
class TIBTimer : public TComponent;
```

File

IBDatabase

Description

This is class IBDatabase.TIBTimer.

20.1.4.1 TIBTimer Members

The following tables list the members exposed by TIBTimer.

Protected Methods

	Name	Description
	Timer (see page 3745)	This is Timer, a member of class TIBTimer.

Public Methods

	Name	Description
	Create (see page 3743)	Allocates memory and constructs a safely initialized instance of a component.
	Destroy (see page 3744)	Disposes of the component and its owned components.

Published Events

	Name	Description
	OnTimer (see page 3746)	This is OnTimer, a member of class TIBTimer.

Published Properties

	Name	Description
	Enabled (see page 3745)	This is Enabled, a member of class TIBTimer.
	Interval (see page 3746)	This is Interval, a member of class TIBTimer.

20.1.4.2 TIBTimer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create

Destroy

System::TObject::Free

Owner

20.1.4.3 TIBTimer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBTimer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy

DestroyComponents

Destroying

System::TObject::Free

Notification

Owner

Release (see page 2133)

Components

20.1.4.4 TIBTimer Methods

The methods of the TIBTimer class are listed here.

Protected Methods

	Name	Description
	Timer (see page 3745)	This is Timer, a member of class TIBTimer.

20.1.4.4.1 TIBTimer.Timer Method

Pascal

```
procedure Timer; dynamic;
```

C++

```
__fastcall Timer();
```

Description

This is Timer, a member of class TIBTimer.

20.1.4.5 TIBTimer Properties

The properties of the TIBTimer class are listed here.

Published Properties

	Name	Description
	Enabled (see page 3745)	This is Enabled, a member of class TIBTimer.
	Interval (see page 3746)	This is Interval, a member of class TIBTimer.

20.1.4.5.1 TIBTimer.Enabled Property

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

This is Enabled, a member of class TIBTimer.

20.1.4.5.2 TIBTimer.Interval Property

Pascal

```
property Interval: Cardinal;
```

C++

```
__property unsigned Interval;
```

Description

This is Interval, a member of class TIBTimer.

20.1.4.6 TIBTimer Events

The events of the TIBTimer class are listed here.

Published Events

	Name	Description
	OnTimer (see page 3746)	This is OnTimer, a member of class TIBTimer.

20.1.4.6.1 TIBTimer.OnTimer Event

Pascal

```
property OnTimer: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTimer;
```

Description

This is OnTimer, a member of class TIBTimer.

20.1.5 TIBTransaction Class

TIBTransaction provides discrete transaction control over a one or more database connections in a database application.

Class Hierarchy

Pascal

```
TIBTransaction = class(TComponent);
```

C++

```
class TIBTransaction : public TComponent;
```

File

IBDatabase

Description

All TIBCustomDataSet descendants and TIBSQL need to use a transaction along with a database component to gain access to data in a database.

Note: In applications that connect an InterBaseExpress dataset to a client dataset, every query must be in its own transaction. You must use one transaction component for each query component.

See Also

Getting Started with InterBase Express

20.1.5.1 TIBTransaction Members

The following tables list the members exposed by TIBTransaction.

Protected Methods

	Name	Description
	Loaded (see page 3752)	Initializes the component after the form file has been read into memory.
	Notification (see page 3752)	Forwards notification messages to all owned components.
	SetHandle (see page 3754)	This is SetHandle, a member of class TIBTransaction.

Public Methods

	Name	Description
	AddDatabase (see page 3749)	Associates a database to the transaction.
	Call (see page 3749)	Returns an error message based on the error code.
	CheckAutoStop (see page 3750)	Performs the action that the AutoStopAction (see page 3755) property specifies.
	CheckDatabasesInList (see page 3750)	Checks for databases in the list.
	CheckInTransaction (see page 3750)	Checks whether the transaction is active and whether there are any databases in the transaction's database list.
	CheckNotInTransaction (see page 3750)	Checks that the transaction is not active and that there are no databases in the transaction's database list.
	Commit (see page 3751)	Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.
	CommitRetaining (see page 3751)	Commits the active transaction and retains the transaction context after a commit.
	Create (see page 3748)	Creates an instance of a transaction component.
	Destroy (see page 3748)	Destroys the instance of the transaction component.
	FindDatabase (see page 3751)	Finds the index of the associated database.
	FindDefaultDatabase (see page 3752)	Finds the default database for the transaction.
	RemoveDatabase (see page 3753)	Disassociates a database from the transaction.
	RemoveDatabases (see page 3753)	Disassociates all databases from the transaction.
	Rollback (see page 3753)	Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.
	RollbackRetaining (see page 3754)	Cancels all updates, insertions, and deletions for the current transaction and retains the transaction context.
	StartTransaction (see page 3754)	Begins a new transaction against the database server.

Public Properties

	Name	Description
	DatabaseCount (see page 3756)	Indicates the number of databases that are part of the transaction.
	Databases (see page 3756)	Returns the database at the given integer index.
	Handle (see page 3757)	Returns the transaction handle.
	HandleIsShared (see page 3757)	Indicates whether or not the handle is shared.
	InTransaction (see page 3758)	Indicates whether a database transaction is in progress or not.
	SQLObjectCount (see page 3758)	Returns the number of active datasets associated with the database component.
	SQLObjects (see page 3758)	Provides an indexed array of all active datasets for a database component.
	TPB (see page 3759)	Provides a read-only view of the transaction parameter buffer.
	TPBLength (see page 3759)	Returns the length of the transaction parameter buffer.

Published Events

	Name	Description
	OnIdleTimer (see page 3759)	Occurs after a transaction has timed out.

Published Properties

	Name	Description
	Active (see page 3755)	Specifies whether or not a transaction is active.
	AutoStopAction (see page 3755)	Specifies what action to take when automatically ending transactions.
	DefaultAction (see page 3756)	Specifies what action a transaction should take upon timing out.
	DefaultDatabase (see page 3757)	Sets or returns the default database for the transaction.
	IdleTimer (see page 3757)	Specifies how long the transaction should wait before automatically committing or rolling back.
	Params (see page 3758)	Returns the transaction parameter buffer associated with the transaction component.

20.1.5.2 TIBTransaction.Create Constructor

Creates an instance of a transaction component.

Pascal

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

C++

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

Description

Call (see page 3749) Create to instantiate a transaction component at runtime. An application creates a transaction component in order to control the component's existence and set its properties and events.

Create instantiates a transaction component and creates an empty string list for the Params (see page 3758) property.

20.1.5.3 TIBTransaction.Destroy Destructor

Destroys the instance of the transaction component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBTransaction();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the transaction component is not already freed before calling Destroy.

Destroy disconnects from the database server, if necessary. It then frees the string resources allocated for the Params (see page 3758) and SQLObjects (see page 3758) properties before calling its inherited destructor.

20.1.5.4 TIBTransaction Methods

The methods of the TIBTransaction class are listed here.

Protected Methods

	Name	Description
	Loaded (see page 3752)	Initializes the component after the form file has been read into memory.
	Notification (see page 3752)	Forwards notification messages to all owned components.
	SetHandle (see page 3754)	This is SetHandle, a member of class TIBTransaction.

Public Methods

	Name	Description
	AddDatabase (see page 3749)	Associates a database to the transaction.
	Call (see page 3749)	Returns an error message based on the error code.
	CheckAutoStop (see page 3750)	Performs the action that the AutoStopAction (see page 3755) property specifies.
	CheckDatabasesInList (see page 3750)	Checks for databases in the list.
	CheckInTransaction (see page 3750)	Checks whether the transaction is active and whether there are any databases in the transaction's database list.
	CheckNotInTransaction (see page 3750)	Checks that the transaction is not active and that there are no databases in the transaction's database list.
	Commit (see page 3751)	Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.
	CommitRetaining (see page 3751)	Commits the active transaction and retains the transaction context after a commit.
	FindDatabase (see page 3751)	Finds the index of the associated database.
	FindDefaultDatabase (see page 3752)	Finds the default database for the transaction.
	RemoveDatabase (see page 3753)	Disassociates a database from the transaction.
	RemoveDatabases (see page 3753)	Disassociates all databases from the transaction.
	Rollback (see page 3753)	Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.
	RollbackRetaining (see page 3754)	Cancels all updates, insertions, and deletions for the current transaction and retains the transaction context.
	StartTransaction (see page 3754)	Begins a new transaction against the database server.

20.1.5.4.1 TIBTransaction.AddDatabase Method

Associates a database to the transaction.

Pascal

```
function AddDatabase(db: TIBDatabase): Integer;
```

C++

```
__fastcall int AddDatabase(TIBDatabase db);
```

Description

Call (see page 3749) AddDatabase to associate a database to the transaction.

20.1.5.4.2 TIBTransaction.Call Method

Returns an error message based on the error code.

Pascal

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

C++

```
__fastcall ISC_STATUS Call(ISC_STATUS ErrCode, Boolean RaiseError);
```

Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning

an error based on the value of RaiseError.

20.1.5.4.3 TIBTransaction.CheckAutoStop Method

Performs the action that the AutoStopAction (see page 3755) property specifies.

Pascal

```
procedure CheckAutoStop;
```

C++

```
__fastcall CheckAutoStop();
```

Description

Datasets call CheckAutoStop at times when it is appropriate to implicitly end a transaction. CheckAutoStop then performs the action that the AutoStopAction (see page 3755) property specifies.

20.1.5.4.4 TIBTransaction.CheckDatabasesInList Method

Checks for databases in the list.

Pascal

```
procedure CheckDatabasesInList;
```

C++

```
__fastcall CheckDatabasesInList();
```

Description

Call (see page 3749) CheckDatabasesInList to check if there are any databases in the list. If there are no databases in the list, an exception is raised.

20.1.5.4.5 TIBTransaction.CheckInTransaction Method

Checks whether the transaction is active and whether there are any databases in the transaction's database list.

Pascal

```
procedure CheckInTransaction;
```

C++

```
__fastcall CheckInTransaction();
```

Description

Call (see page 3749) CheckInTransaction to determine whether the transaction is active and whether there are any databases in the transaction's database list. If either condition is false, an exception is raised.

20.1.5.4.6 TIBTransaction.CheckNotInTransaction Method

Checks that the transaction is not active and that there are no databases in the transaction's database list.

Pascal

```
procedure CheckNotInTransaction;
```

C++

```
__fastcall CheckNotInTransaction();
```

Description

Call (see page 3749) CheckInTransaction (see page 3750) to determine that the transaction is not active and that there are no databases in the transaction's database list. If either condition is false, an exception is raised.

20.1.5.4.7 TIBTransaction.Commit Method

Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.

Pascal

```
procedure Commit;
```

C++

```
__fastcall Commit();
```

Description

Call (see page 3749) Commit to permanently store to the database server all updates, insertions, and deletions of data associated with the current transaction and then end the transaction. The current transaction is the last transaction started by calling StartTransaction (see page 3754).

Note: Before calling Commit, an application may check the status of the InTransaction (see page 3758) property. If an application calls Commit and there is no current transaction, an exception is raised.

20.1.5.4.8 TIBTransaction.CommitRetaining Method

Commits the active transaction and retains the transaction context after a commit.

Pascal

```
procedure CommitRetaining;
```

C++

```
__fastcall CommitRetaining();
```

Description

Call (see page 3749) CommitRetaining to permanently store to the database server all updates, insertions, and deletions of data associated with the current transaction and then retain the transaction context. The current transaction is the last transaction started by calling StartTransaction (see page 3754).

Note: Before calling CommitRetaining, an application may check the status of the InTransaction (see page 3758) property. If an application calls CommitRetaining and there is no current transaction, an exception is raised.

20.1.5.4.9 TIBTransaction.FindDatabase Method

Finds the index of the associated database.

Pascal

```
function FindDatabase(db: TIBDatabase): Integer;
```

C++

```
__fastcall int FindDatabase(TIBDatabase db);
```

Description

Call (see page 3749) FindDatabase to find the index of the associated database.

20.1.5.4.10 TIBTransaction.FindDefaultDatabase Method

Finds the default database for the transaction.

Pascal

```
function FindDefaultDatabase: TIBDatabase;
```

C++

```
__fastcall TIBDatabase FindDefaultDatabase();
```

Description

Call (see page 3749) FindDefaultDatabase to find the default database associated with the transaction.

20.1.5.4.11 TIBTransaction.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

20.1.5.4.12 TIBTransaction.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

20.1.5.4.13 TIBTransaction.RemoveDatabase Method

Disassociates a database from the transaction.

Pascal

```
procedure RemoveDatabase(Idx: Integer);
```

C++

```
__fastcall RemoveDatabase(int Idx);
```

Description

Call (see page 3749) RemoveDatabase to disassociate a specified database from the transaction.

20.1.5.4.14 TIBTransaction.RemoveDatabases Method

Disassociates all databases from the transaction.

Pascal

```
procedure RemoveDatabases;
```

C++

```
__fastcall RemoveDatabases();
```

Description

Call (see page 3749) RemoveDatabases to disassociate all databases from the transaction.

20.1.5.4.15 TIBTransaction.Rollback Method

Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.

Pascal

```
procedure Rollback;
```

C++

```
__fastcall Rollback();
```

Description

Call (see page 3749) Rollback to cancel all updates, insertions, and deletions for the current transaction and to end the transaction. The current transaction is the last transaction started by calling StartTransaction (see page 3754).

Note: Before calling Rollback, an application may check the status of the InTransaction (see page 3758) property. If an application calls Rollback and there is no current transaction, an exception is raised.

20.1.5.4.16 TIBTransaction.RollbackRetaining Method

Cancels all updates, insertions, and deletions for the current transaction and retains the transaction context.

Pascal

```
procedure RollbackRetaining;
```

C++

```
__fastcall RollbackRetaining();
```

Description

Call (see page 3749) RollbackRetaining to roll back to the database server all updates, insertions, and deletions of data associated with the current transaction and then retain the transaction context. The current transaction is the last transaction started by calling StartTransaction (see page 3754).

Note: Before calling RollbackRetaining, an application may check the status of the InTransaction (see page 3758) property. If an application calls RollbackRetaining and there is no current transaction, an exception is raised.

Note: You must install InterBase 6 to use this feature.

20.1.5.4.17 TIBTransaction.SetHandle Method

Pascal

```
procedure SetHandle(Value: TISC_TR_HANDLE);
```

C++

```
__fastcall SetHandle(TISC_TR_HANDLE Value);
```

Description

This is SetHandle, a member of class TIBTransaction.

20.1.5.4.18 TIBTransaction.StartTransaction Method

Begins a new transaction against the database server.

Pascal

```
procedure StartTransaction;
```

C++

```
__fastcall StartTransaction();
```

Description

Call (see page 3749) StartTransaction to begin a new transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction (see page 3758) property. If InTransaction (see page 3758) is true, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit (see page 3751) or Rollback (see page 3753) to end the current transaction raises an exception.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit (see page 3751) to save the changes or Rollback (see page 3753) is to cancel them.

20.1.5.5 TIBTransaction Properties

The properties of the TIBTransaction class are listed here.

Public Properties

	Name	Description
	DatabaseCount (see page 3756)	Indicates the number of databases that are part of the transaction.
	Databases (see page 3756)	Returns the database at the given integer index.
	Handle (see page 3757)	Returns the transaction handle.
	HandleIsShared (see page 3757)	Indicates whether or not the handle is shared.
	InTransaction (see page 3758)	Indicates whether a database transaction is in progress or not.
	SQLObjectCount (see page 3758)	Returns the number of active datasets associated with the database component.
	SQLObjects (see page 3758)	Provides an indexed array of all active datasets for a database component.
	TPB (see page 3759)	Provides a read-only view of the transaction parameter buffer.
	TPBLength (see page 3759)	Returns the length of the transaction parameter buffer.

Published Properties

	Name	Description
	Active (see page 3755)	Specifies whether or not a transaction is active.
	AutoStopAction (see page 3755)	Specifies what action to take when automatically ending transactions.
	DefaultAction (see page 3756)	Specifies what action a transaction should take upon timing out.
	DefaultDatabase (see page 3757)	Sets or returns the default database for the transaction.
	IdleTimer (see page 3757)	Specifies how long the transaction should wait before automatically committing or rolling back.
	Params (see page 3758)	Returns the transaction parameter buffer associated with the transaction component.

20.1.5.5.1 TIBTransaction.Active Property

Specifies whether or not a transaction is active.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set a transaction's active state.

20.1.5.5.2 TIBTransaction.AutoStopAction Property

Specifies what action to take when automatically ending transactions.

Pascal

```
property AutoStopAction: TAutoStopAction;
```

C++

```
__property TAutoStopAction AutoStopAction;
```

Description

Use AutoStopAction to specify how the transaction should end when it is implicitly stopped. Transactions are implicitly stopped when AutoStopAction is not saNone and the last dataset associated with the transaction closes.

Note: This property only works with the closing of TDataSet descendants. If the transaction also includes TIBSQL components, AutoStopAction should be saNone, because the transaction can't tell whether there are any active TIBSQL components.

20.1.5.5.3 TIBTransaction.DatabaseCount Property

Indicates the number of databases that are part of the transaction.

Pascal

```
property DatabaseCount: Integer;
```

C++

```
__property int DatabaseCount;
```

Description

Use DatabaseCount to determine the number of databases involved in a transaction.

20.1.5.5.4 TIBTransaction.Databases Property

Returns the database at the given integer index.

Pascal

```
property Databases [Index: Integer]: TIBDatabase;
```

C++

```
__property TIBDatabase Databases[int Index];
```

Description

Use Databases to return the database at the given integer index.

20.1.5.5.5 TIBTransaction.DefaultAction Property

Specifies what action a transaction should take upon timing out.

Pascal

```
property DefaultAction: TTransactionAction;
```

C++

```
__property TTransactionAction DefaultAction;
```

Description

Use DefaultAction to what action the transaction should take when the IdleTimer (see page 3757) limit is met.

20.1.5.5.6 TIBTransaction.DefaultDatabase Property

Sets or returns the default database for the transaction.

Pascal

```
property DefaultDatabase: TIBDatabase;
```

C++

```
__property TIBDatabase DefaultDatabase;
```

Description

Use DefaultDatabase to set or return the default database for the transaction.

20.1.5.5.7 TIBTransaction.Handle Property

Returns the transaction handle.

Pascal

```
property Handle: TISC_TR_HANDLE;
```

C++

```
__property TISC_TR_HANDLE Handle;
```

Description

Use Handle to retrieve a handle to the transaction. Handle is assigned an initial value when a transaction is started.

20.1.5.5.8 TIBTransaction.HandleIsShared Property

Indicates whether or not the handle is shared.

Pascal

```
property HandleIsShared: Boolean;
```

C++

```
__property Boolean HandleIsShared;
```

Description

HandleIsShared returns true when the transaction handle is shared by more than one transaction component.

20.1.5.5.9 TIBTransaction.IdleTimer Property

Specifies how long the transaction should wait before automatically committing or rolling back.

Pascal

```
property IdleTimer: Integer;
```

C++

```
__property int IdleTimer;
```

Description

Use IdleTimer to indicate how long a transaction should be allowed to remain idle before automatically committing or rolling back the data. Use DefaultAction (see page 3756) to determine which action the transaction should take.

20.1.5.5.10 TIBTransaction.InTransaction Property

Indicates whether a database transaction is in progress or not.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Examine InTransaction at run-time to determine if a database transaction is currently in progress. InTransaction is true if a transaction is in progress, false otherwise.

The value of InTransaction cannot be changed directly. Calling StartTransaction (see page 3754) sets InTransaction to true. Calling Commit (see page 3751) or Rollback (see page 3753) sets InTransaction to false.

20.1.5.5.11 TIBTransaction.Params Property

Returns the transaction parameter buffer associated with the transaction component.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Use Params to examine and set parameters in the transaction parameter buffer. Refer to the Interbase API Guide for the names of the parameters to provide.

20.1.5.5.12 TIBTransaction.SQLObjectCount Property

Returns the number of active datasets associated with the database component.

Pascal

```
property SQLObjectCount: Integer;
```

C++

```
__property int SQLObjectCount;
```

Description

Use the SQLObjectCount property to return the number currently active InterBase datasets, TIBSQL objects, and Blobs associated with the database component. As SQL objects are opened and closed, this value changes appropriately.

20.1.5.5.13 TIBTransaction.SQLObjects Property

Provides an indexed array of all active datasets for a database component.

Pascal

```
property SQLObjects [Index: Integer]: TIBBase;
```

C++

```
__property TIBBase SQLObjects[int Index];
```

Description

Use the SQLObjects to access active InterBase datasets, TIBSQL objects, and Blobs associated with the database component.

20.1.5.5.14 TIBTransaction.TPB Property

Provides a read-only view of the transaction parameter buffer.

Pascal

```
property TPB: PChar;
```

C++

```
__property const char * TPB;
```

Description

Use TPB view the transaction parameter buffer. To write to the transaction parameter buffer, use the Params (see page 3758) property.

20.1.5.5.15 TIBTransaction.TPBLength Property

Returns the length of the transaction parameter buffer.

Pascal

```
property TPBLength: Short;
```

C++

```
__property Short TPBLength;
```

Description

Use TPBLength to retrieve the length of the transaction parameter buffer

20.1.5.6 TIBTransaction Events

The events of the TIBTransaction class are listed here.

Published Events

	Name	Description
	OnIdleTimer (see page 3759)	Occurs after a transaction has timed out.

20.1.5.6.1 TIBTransaction.OnIdleTimer Event

Occurs after a transaction has timed out.

Pascal

```
property OnIdleTimer: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnIdleTimer;
```

Description

Write an OnIdleTimer event handler to take specific actions after a transaction is allowed to remain idle for the number of seconds specified by IdleTimer (see page 3757).

20.2 Interfaces

The following table lists interfaces in this documentation.

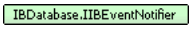
Interfaces

	Name	Description
	IIBEventNotifier (see page 3761)	IIBEventNotifier allows a TIBDatabase component to automatically register interest in events on the InterBase server.

20.2.1 IIBEventNotifier Interface

IIBEventNotifier allows a TIBDatabase component to automatically register interest in events on the InterBase server.

Class Hierarchy



Pascal

```
IIBEventNotifier = interface;
```

C++

```
__interface IIBEventNotifier;
```

File

IBDatabase

Description

TIBDatabase uses the IIBEventNotifier interface to communicate with objects such as TIBEvents that register an interest in events on the InterBase server.

Note: VCL Method declarations that use IIBEventNotifier use the _di_IIBEventNotifier type instead. This is a DelphiInterface wrapper around the IIBEventNotifier interface:

```
typedef System::DelphiInterface< IIBEventNotifier > _di_IIBEventNotifier;
```

20.2.1.1 IIBEventNotifier Members

The following tables list the members exposed by IIBEventNotifier.

Public Methods

	Name	Description
	GetAutoRegister (see page 3762)	Indicates whether the database should automatically register events after it connects to the server.
	RegisterEvents (see page 3762)	Registers interest in a set of events with the InterBase Event manager.
	UnRegisterEvents (see page 3762)	Unregisters interest in the events previously registered by a call to RegisterEvents (see page 3762).

20.2.1.2 IIBEventNotifier Methods

The methods of the IIBEventNotifier class are listed here.

Public Methods

	Name	Description
	GetAutoRegister (see page 3762)	Indicates whether the database should automatically register events after it connects to the server.
	RegisterEvents (see page 3762)	Registers interest in a set of events with the InterBase Event manager.
	UnRegisterEvents (see page 3762)	Unregisters interest in the events previously registered by a call to RegisterEvents (see page 3762).

20.2.1.2.1 IIBEventNotifier.GetAutoRegister Method

Indicates whether the database should automatically register events after it connects to the server.

Pascal

```
function GetAutoRegister: Boolean;
```

C++

```
__fastcall Boolean GetAutoRegister();
```

Description

When TIBDatabase connects to an InterBase server, it calls the GetAutoRegister method for every object in its list of event notifiers. For any event notifier that returns true, it calls the RegisterEvents (see page 3762) method, so that the event notifier can register interest in its events with the InterBase Event Manager.

20.2.1.2.2 IIBEventNotifier.RegisterEvents Method

Registers interest in a set of events with the InterBase Event manager.

Pascal

```
procedure RegisterEvents;
```

C++

```
__fastcall RegisterEvents();
```

Description

RegisterEvents tells the event notifier to register its interest in a set of events with the InterBase Event manager.

20.2.1.2.3 IIBEventNotifier.UnRegisterEvents Method

Unregisters interest in the events previously registered by a call to RegisterEvents (see page 3762).

Pascal

```
procedure UnRegisterEvents;
```

C++

```
__fastcall UnRegisterEvents();
```

Description

UnRegisterEvents cancels the interest in the events that was previously registered by a call to RegisterEvents (see page 3762). TIBDataBase (see page 3716) uses this interface to unregister all of its event notifiers when it disconnects from the

InterBase server.

20.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GenerateDPB (see page 3764)	This is function IBDatabase.GenerateDPB.
	GenerateTPB (see page 3764)	This is function IBDatabase.GenerateTPB.

20.3.1 IBDatabase.GenerateDPB Function

Pascal

```
procedure GenerateDPB(sl: TStrings; var DPB: string; var DPBLength: Short);
```

C++

```
GenerateDPB(TStrings sl, UnicodeString DPB, Short DPBLength);
```

File

IBDatabase

Description

This is function IBDatabase.GenerateDPB.

20.3.2 IBDatabase.GenerateTPB Function

Pascal

```
procedure GenerateTPB(sl: TStrings; var TPB: string; var TPBLength: Short);
```

C++

```
GenerateTPB(TStrings sl, UnicodeString TPB, Short TPBLength);
```

File

IBDatabase

Description

This is function IBDatabase.GenerateTPB.

20.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoStopAction (see page 3765)	TAutoStopAction indicates the action to take on automatically ended transactions.
	TTransactionAction (see page 3766)	TTransactionAction indicates the action a transaction can take when it times out.

20.4.1 IBDatabase.TAutoStopAction Enumeration

TAutoStopAction indicates the action to take on automatically ended transactions.

Pascal

```
TAutoStopAction = (
  saNone,
  saRollback,
  saCommit,
  saRollbackRetaining,
  saCommitRetaining
);
```

C++

```
enum TAutoStopAction {
  saNone,
  saRollback,
  saCommit,
  saRollbackRetaining,
  saCommitRetaining
};
```

File

IBDatabase

Description

TAutoStopAction values are:

Value Meaning

saNone Transactions are not implicitly stopped.

saRollback Transactions are rolled back and closed when stopped implicitly.

saCommit Transactions are committed and closed when stopped implicitly.

saRollbackRetaining Transactions are not stopped when the last dataset closes, but all updates, insertions, and deletions of data associated with the current transaction are backed out. This option is only supported in InterBase 6.0 and later.

saCommitRetaining Transactions are not stopped when the last dataset closes, but all updates, insertions, and deletions of data associated with the current transaction are committed.

20.4.2 IBDatabase.TTransactionAction Enumeration

TTransactionAction indicates the action a transaction can take when it times out.

Pascal

```
TTransactionAction = (
  TARollback,
  TACommit,
  TARollbackRetaining,
  TACommitRetaining
);
```

C++

```
enum TTransactionAction {
  TARollback,
  TACommit,
  TARollbackRetaining,
  TACommitRetaining
};
```

File

IBDatabase

Description

TTransactionAction values are:

Value Meaning

TARollback Roll back the transaction.

TACommit Commit the transaction.

TARollbackRetaining Roll back the transaction, but retain the current transaction context. (Requires InterBase 6.0 or later)

TACommitRetaining Commit the transaction, but retain the current transaction context.

20.5 Types

The following table lists types in this documentation.

Types

Name	Description
TIBDatabaseLoginEvent (see page 3767)	TIBDatabaseLoginEvent is the type of event handlers that respond when an IB (see page 3541) database component forms a connection to an InterBase server.
TIBFileName (see page 3767)	This is type IBDatabase.TIBFileName.

20.5.1 IBDatabase.TIBDatabaseLoginEvent Type

TIBDatabaseLoginEvent is the type of event handlers that respond when an IB (see page 3541) database component forms a connection to an InterBase server.

Pascal

```
TIBDatabaseLoginEvent = procedure (Database: TIBDatabase; LoginParams: TStrings) of object;
```

C++

```
(Database: TIBDatabase; LoginParams: TStrings) ( TIBDatabaseLoginEvent)();
```

File

IBDatabase

Description

Database is the connection component that establishes the connection to the InterBase server.

LoginParams is a temporary string list that contains the login parameters for forming the connection. The event handler can examine or change any of these value. Each parameter is of the form Name=Value. Of particular importance are the parameters with the names USER_NAME and PASSWORD. This string list does not persist: the IB (see page 3541) database component frees it when it is no longer needed.

20.5.2 IBDatabase.TIBFileName Type

Pascal

```
TIBFileName = type string;
```

C++

```
type string TIBFileName;
```

File

IBDatabase

Description

This is type IBDatabase.TIBFileName.

20.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DPBConstantNames (see page 3768)	This is constant IBDatabase.DPBConstantNames.
DPBPrefix (see page 3769)	This is constant IBDatabase.DPBPrefix.
TPBConstantNames (see page 3769)	This is constant IBDatabase.TPBConstantNames.
TPBPrefix (see page 3769)	This is constant IBDatabase.TPBPrefix.

20.6.1 IBDatabase.DPBConstantNames Constant

Pascal

```
DPBConstantNames: array[1..isc_dpb_last_dpb_constant] of string = ( 'cdd_pathname',
'allocation', 'journal', 'page_size', 'num_buffers', 'buffer_length', 'debug',
'garbage_collect', 'verify', 'sweep', 'enable_journal', 'disable_journal', 'dbkey_scope',
'number_of_users', 'trace', 'no_garbage_collect', 'damaged', 'license', 'sys_user_name',
'encrypt_key', 'activate_shadow', 'sweep_interval', 'delete_shadow', 'force_write',
'begin_log', 'quit_log', 'no_reserve', 'user_name', 'password', 'password_enc',
'sys_user_name_enc', 'interp', 'online_dump', 'old_file_size', 'old_num_files', 'old_file',
'old_start_page', 'old_start_seqno', 'old_start_file', 'drop_walfile', 'old_dump_id',
'wal_backup_dir', 'wal_chkptlen', 'wal_numbufs', 'wal_bufsize', 'wal_grp_cmt_wait',
'lc_messages', 'lc_ctype', 'cache_manager', 'shutdown', 'online', 'shutdown_delay',
'reserved', 'overwrite', 'sec_attach', 'disable_wal', 'connect_timeout',
'dummy_packet_interval', 'gbak_attach', 'sql_role_name', 'set_page_buffers',
'working_directory', 'sql_dialect', 'set_db_readonly', 'set_db_sql_dialect', 'gfix_attach',
'gstat_attach' );
```

C++

```
array[1..isc_dpb_last_dpb_constant] of string DPBConstantNames = ( 'cdd_pathname',
'allocation', 'journal', 'page_size', 'num_buffers', 'buffer_length', 'debug',
'garbage_collect', 'verify', 'sweep', 'enable_journal', 'disable_journal', 'dbkey_scope',
'number_of_users', 'trace', 'no_garbage_collect', 'damaged', 'license', 'sys_user_name',
'encrypt_key', 'activate_shadow', 'sweep_interval', 'delete_shadow', 'force_write',
'begin_log', 'quit_log', 'no_reserve', 'user_name', 'password', 'password_enc',
'sys_user_name_enc', 'interp', 'online_dump', 'old_file_size', 'old_num_files', 'old_file',
'old_start_page', 'old_start_seqno', 'old_start_file', 'drop_walfile', 'old_dump_id',
'wal_backup_dir', 'wal_chkptlen', 'wal_numbufs', 'wal_bufsize', 'wal_grp_cmt_wait',
'lc_messages', 'lc_ctype', 'cache_manager', 'shutdown', 'online', 'shutdown_delay',
'reserved', 'overwrite', 'sec_attach', 'disable_wal', 'connect_timeout',
'dummy_packet_interval', 'gbak_attach', 'sql_role_name', 'set_page_buffers',
'working_directory', 'sql_dialect', 'set_db_readonly', 'set_db_sql_dialect', 'gfix_attach',
'gstat_attach' );
```

File

IBDatabase

Description

This is constant IBDatabase.DPBConstantNames.

20.6.2 IBDatabase.DPBPrefix Constant

Pascal

```
DPBPrefix = 'isc_dpb_';
```

C++

```
DPBPrefix = 'isc_dpb_';
```

File

IBDatabase

Description

This is constant IBDatabase.DPBPrefix.

20.6.3 IBDatabase.TPBConstantNames Constant

Pascal

```
TPBConstantNames: array[1..isc_tpb_last_tpb_constant] of string = ( 'consistency',
'concurrency', 'shared', 'protected', 'exclusive', 'wait', 'nowait', 'read', 'write',
'lock_read', 'lock_write', 'verb_time', 'commit_time', 'ignore_limbo', 'read_committed',
'autocommit', 'rec_version', 'no_rec_version', 'restart_requests', 'no_auto_undo' );
```

C++

```
array[1..isc_tpb_last_tpb_constant] of string TPBConstantNames = ( 'consistency',
'concurrency', 'shared', 'protected', 'exclusive', 'wait', 'nowait', 'read', 'write',
'lock_read', 'lock_write', 'verb_time', 'commit_time', 'ignore_limbo', 'read_committed',
'autocommit', 'rec_version', 'no_rec_version', 'restart_requests', 'no_auto_undo' );
```

File

IBDatabase

Description

This is constant IBDatabase.TPBConstantNames.

20.6.4 IBDatabase.TPBPrefix Constant

Pascal

```
TPBPrefix = 'isc_tpb_';
```

C++

```
TPBPrefix = 'isc_tpb_';
```

File

IBDatabase

Description

This is constant IBDatabase.TPBPrefix.

21 IBDatabaseInfo Namespace

This is namespace IBDatabaseInfo.

Topics

Name	Description
Classes (see page 3772)	The following table lists classes in this documentation.

Classes

	Name	Description
	TIBDatabaseInfo (see page 3772)	TIBDatabaseInfo returns information about the attached database.

21.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBDatabaseInfo (see page 3772)	TIBDatabaseInfo returns information about the attached database.

21.1.1 TIBDatabaseInfo Class

TIBDatabaseInfo returns information about the attached database.

Class Hierarchy

Pascal

```
TIBDatabaseInfo = class(TComponent);
```

C++

```
class TIBDatabaseInfo : public TComponent;
```

File

IBDatabaseInfo

Description

Use TIBDatabaseInfo to fetch information about the attached database, such as the version of the online disk structure (ODS) used by the attachment, the number of database cache buffers allocated, the number of database pages read from or written to, or write-ahead log information.

See Also

Getting Started with InterBase Express

21.1.1.1 TIBDatabaseInfo Members

The following tables list the members exposed by TIBDatabaseInfo.

Protected Fields

	Name	Description
	FBackoutCount (see page 3775)	This is FBackoutCount, a member of class TIBDatabaseInfo.
	FDatabase (see page 3775)	This is FDatabase, a member of class TIBDatabaseInfo.
	FDeleteCount (see page 3776)	This is FDeleteCount, a member of class TIBDatabaseInfo.
	FExpungeCount (see page 3776)	This is FExpungeCount, a member of class TIBDatabaseInfo.
	FIBLoaded (see page 3776)	This is FIBLoaded, a member of class TIBDatabaseInfo.
	FInsertCount (see page 3776)	This is FInsertCount, a member of class TIBDatabaseInfo.
	FPurgeCount (see page 3776)	This is FPurgeCount, a member of class TIBDatabaseInfo.
	FReadIdxCount (see page 3777)	This is FReadIdxCount, a member of class TIBDatabaseInfo.

	FReadSeqCount (see page 3777)	This is FReadSeqCount, a member of class TIBDatabaseInfo.
	FUpdateCount (see page 3777)	This is FUpdateCount, a member of class TIBDatabaseInfo.
	FUserNames (see page 3777)	This is FUserNames, a member of class TIBDatabaseInfo.

Protected Methods

	Name	Description
	GetAllocation (see page 3779)	This is GetAllocation, a member of class TIBDatabaseInfo.
	GetBackoutCount (see page 3779)	This is GetBackoutCount, a member of class TIBDatabaseInfo.
	GetBaseLevel (see page 3779)	This is GetBaseLevel, a member of class TIBDatabaseInfo.
	GetCurrentMemory (see page 3779)	This is GetCurrentMemory, a member of class TIBDatabaseInfo.
	GetDBFileName (see page 3779)	This is GetDBFileName, a member of class TIBDatabaseInfo.
	GetDBImplementationClass (see page 3780)	This is GetDBImplementationClass, a member of class TIBDatabaseInfo.
	GetDBImplementationNo (see page 3780)	This is GetDBImplementationNo, a member of class TIBDatabaseInfo.
	GetDBSiteName (see page 3780)	This is GetDBSiteName, a member of class TIBDatabaseInfo.
	GetDBSQLDialect (see page 3780)	This is GetDBSQLDialect, a member of class TIBDatabaseInfo.
	GetDeleteCount (see page 3781)	This is GetDeleteCount, a member of class TIBDatabaseInfo.
	GetExpungeCount (see page 3781)	This is GetExpungeCount, a member of class TIBDatabaseInfo.
	GetFetches (see page 3781)	This is GetFetches, a member of class TIBDatabaseInfo.
	GetForcedWrites (see page 3781)	This is GetForcedWrites, a member of class TIBDatabaseInfo.
	GetInsertCount (see page 3781)	This is GetInsertCount, a member of class TIBDatabaseInfo.
	GetMarks (see page 3782)	This is GetMarks, a member of class TIBDatabaseInfo.
	GetMaxMemory (see page 3782)	This is GetMaxMemory, a member of class TIBDatabaseInfo.
	GetNoReserve (see page 3782)	This is GetNoReserve, a member of class TIBDatabaseInfo.
	GetNumBuffers (see page 3783)	This is GetNumBuffers, a member of class TIBDatabaseInfo.
	GetODSMajorVersion (see page 3783)	This is GetODSMajorVersion, a member of class TIBDatabaseInfo.
	GetODSMinorVersion (see page 3783)	This is GetODSMinorVersion, a member of class TIBDatabaseInfo.
	GetOperationCounts (see page 3783)	This is GetOperationCounts, a member of class TIBDatabaseInfo.
	GetPageSize (see page 3784)	This is GetPageSize, a member of class TIBDatabaseInfo.
	GetPurgeCount (see page 3784)	This is GetPurgeCount, a member of class TIBDatabaseInfo.
	GetReadIdxCount (see page 3784)	This is GetReadIdxCount, a member of class TIBDatabaseInfo.
	GetReadOnly (see page 3784)	This is GetReadOnly, a member of class TIBDatabaseInfo.
	GetReads (see page 3784)	This is GetReads, a member of class TIBDatabaseInfo.
	GetReadSeqCount (see page 3785)	This is GetReadSeqCount, a member of class TIBDatabaseInfo.
	GetStringDatabaseInfo (see page 3785)	This is GetStringDatabaseInfo, a member of class TIBDatabaseInfo.
	GetSweepInterval (see page 3785)	This is GetSweepInterval, a member of class TIBDatabaseInfo.
	GetUpdateCount (see page 3785)	This is GetUpdateCount, a member of class TIBDatabaseInfo.
	GetUserNames (see page 3786)	This is GetUserNames, a member of class TIBDatabaseInfo.
	GetVersion (see page 3786)	This is GetVersion, a member of class TIBDatabaseInfo.
	GetWrites (see page 3786)	This is GetWrites, a member of class TIBDatabaseInfo.

Public Methods

	Name	Description
	Call (see page 3778)	Returns an error message based on the error code.
V	Create (see page 3774)	Creates an instance of a DatabaseInfo component.
V	Destroy (see page 3774)	Destroys an instance of a DatabaseInfo component.
	GetLongDatabaseInfo (see page 3782)	Returns a specified long integer that describes some aspect of the associated database.

Public Properties

	Name	Description
R	Allocation (see page 3787)	Returns the number of database pages allocated.
R	BackoutCount (see page 3787)	Returns the number of removals of a version of a record.
R	BaseLevel (see page 3788)	Returns the database version number.
R	CurrentMemory (see page 3788)	Returns the amount of server memory currently in use.
R	DBFileName (see page 3788)	Returns the database filename.

	DBImplementationClass (see page 3789)	Returns the database implementation class number.
	DBImplementationNo (see page 3789)	Returns the database implementation number.
	DBSiteName (see page 3789)	Returns the database site name.
	DBSQLDialect (see page 3789)	Returns the SQL dialect.
	DeleteCount (see page 3790)	Returns the number of database deletes since the database was last attached.
	ExpungeCount (see page 3790)	Returns the number of removals of a record and all of its ancestors.
	Fetches (see page 3790)	Returns the number of reads from the memory buffer cache.
	ForcedWrites (see page 3790)	Returns the mode in which database writes are performed.
	InsertCount (see page 3791)	Returns number of inserts into the database since the database was last attached.
	Marks (see page 3791)	Returns the number of writes to the memory buffer cache.
	MaxMemory (see page 3791)	Returns the maximum amount of memory used at one time since the first process attached to the database.
	NoReserve (see page 3791)	Returns whether or not space is reserved on each database page for holding backup versions of modified records.
	NumBuffers (see page 3792)	Returns the number of memory buffers currently allocated.
	ODSMajorVersion (see page 3792)	Returns the on disk structure (ODS) major version number.
	ODSMinorVersion (see page 3792)	Returns the on disk structure (ODS) minor version number.
	PageSize (see page 3792)	Returns the number of bytes per page of the attached database.
	PurgeCount (see page 3793)	Returns the number of removals of fully mature records from the database.
	ReadIdxCount (see page 3793)	Returns the number of reads done via an index since the database was last attached.
	ReadOnly (see page 3793)	Indicates whether or not the database is read only.
	Reads (see page 3793)	Returns the number of page reads from the database.
	ReadSeqCount (see page 3794)	Returns the number of sequential database reads done on each table since the database was last attached.
	SweepInterval (see page 3794)	Returns the number of transactions that are committed between 'sweeps.'
	UpdateCount (see page 3794)	Returns the number of database updates since the database was last attached.
	UserNames (see page 3795)	Returns the names of all users currently attached to the database.
	Version (see page 3795)	Returns the version of the database implementation.
	Writes (see page 3795)	Returns the number of page writes to the database.

Published Properties

	Name	Description
	Database (see page 3788)	Sets or returns the database.

21.1.1.2 TIBDatabaseInfo.Create Constructor

Creates an instance of a DatabaseInfo component.

Pascal

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

C++

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

Description

Call (see page 3778) Create to instantiate a DatabaseInfo component declared in an application.

21.1.1.3 TIBDatabaseInfo.Destroy Destructor

Destroys an instance of a DatabaseInfo component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBDatabaseInfo();
```

Description

Do not call Destroy directly. Instead call Free to verify that the DatabaseInfo component is not already freed before calling Destroy. Destroy disconnects from the server, frees the parameter list, and calls its inherited Destroy destructor.

21.1.1.4 TIBDatabaseInfo Fields

The fields of the TIBDatabaseInfo class are listed here.

Protected Fields

	Name	Description
	FBackoutCount (see page 3775)	This is FBackoutCount, a member of class TIBDatabaseInfo.
	FDatabase (see page 3775)	This is FDatabase, a member of class TIBDatabaseInfo.
	FDeleteCount (see page 3776)	This is FDeleteCount, a member of class TIBDatabaseInfo.
	FExpungeCount (see page 3776)	This is FExpungeCount, a member of class TIBDatabaseInfo.
	FIBLoaded (see page 3776)	This is FIBLoaded, a member of class TIBDatabaseInfo.
	FInsertCount (see page 3776)	This is FInsertCount, a member of class TIBDatabaseInfo.
	FPurgeCount (see page 3776)	This is FPurgeCount, a member of class TIBDatabaseInfo.
	FReadIdxCount (see page 3777)	This is FReadIdxCount, a member of class TIBDatabaseInfo.
	FReadSeqCount (see page 3777)	This is FReadSeqCount, a member of class TIBDatabaseInfo.
	FUpdateCount (see page 3777)	This is FUpdateCount, a member of class TIBDatabaseInfo.
	FUserNames (see page 3777)	This is FUserNames, a member of class TIBDatabaseInfo.

21.1.1.4.1 TIBDatabaseInfo.FBackoutCount Field

Pascal

```
FBackoutCount: TStringList;
```

C++

```
TStringList FBackoutCount;
```

Description

This is FBackoutCount, a member of class TIBDatabaseInfo.

21.1.1.4.2 TIBDatabaseInfo.FDatabase Field

Pascal

```
FDatabase: TIBDatabase;
```

C++

```
TIBDatabase FDatabase;
```

Description

This is FDatabase, a member of class TIBDatabaseInfo.

21.1.1.4.3 TIBDatabaseInfo.FDeleteCount Field

Pascal

```
FDeleteCount: TStringList;
```

C++

```
TStringList FDeleteCount;
```

Description

This is FDeleteCount, a member of class TIBDatabaseInfo.

21.1.1.4.4 TIBDatabaseInfo.FExpungeCount Field

Pascal

```
FExpungeCount: TStringList;
```

C++

```
TStringList FExpungeCount;
```

Description

This is FExpungeCount, a member of class TIBDatabaseInfo.

21.1.1.4.5 TIBDatabaseInfo.FIBLoaded Field

Pascal

```
FIBLoaded: Boolean;
```

C++

```
Boolean FIBLoaded;
```

Description

This is FIBLoaded, a member of class TIBDatabaseInfo.

21.1.1.4.6 TIBDatabaseInfo.FInsertCount Field

Pascal

```
FInsertCount: TStringList;
```

C++

```
TStringList FInsertCount;
```

Description

This is FInsertCount, a member of class TIBDatabaseInfo.

21.1.1.4.7 TIBDatabaseInfo.FPurgeCount Field

Pascal

```
FPurgeCount: TStringList;
```

C++

```
TStringList FPurgeCount;
```

Description

This is FPurgeCount, a member of class TIBDatabaseInfo.

21.1.1.4.8 TIBDatabaseInfo.FReadIdxCount Field

Pascal

```
FReadIdxCount: TStringList;
```

C++

```
TStringList FReadIdxCount;
```

Description

This is FReadIdxCount, a member of class TIBDatabaseInfo.

21.1.1.4.9 TIBDatabaseInfo.FReadSeqCount Field

Pascal

```
FReadSeqCount: TStringList;
```

C++

```
TStringList FReadSeqCount;
```

Description

This is FReadSeqCount, a member of class TIBDatabaseInfo.

21.1.1.4.10 TIBDatabaseInfo.FUpdateCount Field

Pascal

```
FUpdateCount: TStringList;
```

C++

```
TStringList FUpdateCount;
```

Description

This is FUpdateCount, a member of class TIBDatabaseInfo.

21.1.1.4.11 TIBDatabaseInfo.FUserNames Field

Pascal

```
FUserNames: TStringList;
```

C++

```
TStringList FUserNames;
```

Description

This is FUserNames, a member of class TIBDatabaseInfo.

21.1.1.5 TIBDatabaseInfo Methods

The methods of the TIBDatabaseInfo class are listed here.

Protected Methods

	Name	Description
	GetAllocation (see page 3779)	This is GetAllocation, a member of class TIBDatabaseInfo.
	GetBackoutCount (see page 3779)	This is GetBackoutCount, a member of class TIBDatabaseInfo.
	GetBaseLevel (see page 3779)	This is GetBaseLevel, a member of class TIBDatabaseInfo.
	GetCurrentMemory (see page 3779)	This is GetCurrentMemory, a member of class TIBDatabaseInfo.
	GetDBFileName (see page 3779)	This is GetDBFileName, a member of class TIBDatabaseInfo.
	GetDBImplementationClass (see page 3780)	This is GetDBImplementationClass, a member of class TIBDatabaseInfo.
	GetDBImplementationNo (see page 3780)	This is GetDBImplementationNo, a member of class TIBDatabaseInfo.
	GetDBSiteName (see page 3780)	This is GetDBSiteName, a member of class TIBDatabaseInfo.
	GetDBSQLDialect (see page 3780)	This is GetDBSQLDialect, a member of class TIBDatabaseInfo.
	GetDeleteCount (see page 3781)	This is GetDeleteCount, a member of class TIBDatabaseInfo.
	GetExpungeCount (see page 3781)	This is GetExpungeCount, a member of class TIBDatabaseInfo.
	GetFetches (see page 3781)	This is GetFetches, a member of class TIBDatabaseInfo.
	GetForcedWrites (see page 3781)	This is GetForcedWrites, a member of class TIBDatabaseInfo.
	GetInsertCount (see page 3781)	This is GetInsertCount, a member of class TIBDatabaseInfo.
	GetMarks (see page 3782)	This is GetMarks, a member of class TIBDatabaseInfo.
	GetMaxMemory (see page 3782)	This is GetMaxMemory, a member of class TIBDatabaseInfo.
	GetNoReserve (see page 3782)	This is GetNoReserve, a member of class TIBDatabaseInfo.
	GetNumBuffers (see page 3783)	This is GetNumBuffers, a member of class TIBDatabaseInfo.
	GetODSMajorVersion (see page 3783)	This is GetODSMajorVersion, a member of class TIBDatabaseInfo.
	GetODSMinorVersion (see page 3783)	This is GetODSMinorVersion, a member of class TIBDatabaseInfo.
	GetOperationCounts (see page 3783)	This is GetOperationCounts, a member of class TIBDatabaseInfo.
	GetPageSize (see page 3784)	This is GetPageSize, a member of class TIBDatabaseInfo.
	GetPurgeCount (see page 3784)	This is GetPurgeCount, a member of class TIBDatabaseInfo.
	GetReadIdxCount (see page 3784)	This is GetReadIdxCount, a member of class TIBDatabaseInfo.
	GetReadOnly (see page 3784)	This is GetReadOnly, a member of class TIBDatabaseInfo.
	GetReads (see page 3784)	This is GetReads, a member of class TIBDatabaseInfo.
	GetReadSeqCount (see page 3785)	This is GetReadSeqCount, a member of class TIBDatabaseInfo.
	GetStringDatabaseInfo (see page 3785)	This is GetStringDatabaseInfo, a member of class TIBDatabaseInfo.
	GetSweepInterval (see page 3785)	This is GetSweepInterval, a member of class TIBDatabaseInfo.
	GetUpdateCount (see page 3785)	This is GetUpdateCount, a member of class TIBDatabaseInfo.
	GetUserNames (see page 3786)	This is GetUserNames, a member of class TIBDatabaseInfo.
	GetVersion (see page 3786)	This is GetVersion, a member of class TIBDatabaseInfo.
	GetWrites (see page 3786)	This is GetWrites, a member of class TIBDatabaseInfo.

Public Methods

	Name	Description
	Call (see page 3778)	Returns an error message based on the error code.
	GetLongDatabaseInfo (see page 3782)	Returns a specified long integer that describes some aspect of the associated database.

21.1.1.5.1 TIBDatabaseInfo.Call Method

Returns an error message based on the error code.

Pascal

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

C++

```
__fastcall ISC_STATUS Call(ISC_STATUS ErrCode, Boolean RaiseError);
```

Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning

an error based on the value of RaiseError.

21.1.1.5.2 TIBDatabaseInfo.GetAllocation Method

Pascal

```
function GetAllocation: Long;
```

C++

```
__fastcall Long GetAllocation();
```

Description

This is GetAllocation, a member of class TIBDatabaseInfo.

21.1.1.5.3 TIBDatabaseInfo.GetBackoutCount Method

Pascal

```
function GetBackoutCount: TStringList;
```

C++

```
__fastcall TStringList GetBackoutCount();
```

Description

This is GetBackoutCount, a member of class TIBDatabaseInfo.

21.1.1.5.4 TIBDatabaseInfo.GetBaseLevel Method

Pascal

```
function GetBaseLevel: Long;
```

C++

```
__fastcall Long GetBaseLevel();
```

Description

This is GetBaseLevel, a member of class TIBDatabaseInfo.

21.1.1.5.5 TIBDatabaseInfo.GetCurrentMemory Method

Pascal

```
function GetCurrentMemory: Long;
```

C++

```
__fastcall Long GetCurrentMemory();
```

Description

This is GetCurrentMemory, a member of class TIBDatabaseInfo.

21.1.1.5.6 TIBDatabaseInfo.GetDBFileName Method

Pascal

```
function GetDBFileName: String;
```

C++

```
__fastcall UnicodeString GetDBFileName();
```

Description

This is GetDBFileName, a member of class TIBDatabaseInfo.

21.1.1.5.7 TIBDatabaseInfo.GetDBImplementationClass Method

Pascal

```
function GetDBImplementationClass: Long;
```

C++

```
__fastcall Long GetDBImplementationClass();
```

Description

This is GetDBImplementationClass, a member of class TIBDatabaseInfo.

21.1.1.5.8 TIBDatabaseInfo.GetDBImplementationNo Method

Pascal

```
function GetDBImplementationNo: Long;
```

C++

```
__fastcall Long GetDBImplementationNo();
```

Description

This is GetDBImplementationNo, a member of class TIBDatabaseInfo.

21.1.1.5.9 TIBDatabaseInfo.GetDBSiteName Method

Pascal

```
function GetDBSiteName: String;
```

C++

```
__fastcall UnicodeString GetDBSiteName();
```

Description

This is GetDBSiteName, a member of class TIBDatabaseInfo.

21.1.1.5.10 TIBDatabaseInfo.GetDBSQLDialect Method

Pascal

```
function GetDBSQLDialect: Long;
```

C++

```
__fastcall Long GetDBSQLDialect();
```

Description

This is GetDBSQLDialect, a member of class TIBDatabaseInfo.

21.1.1.5.11 TIBDatabaseInfo.GetDeleteCount Method

Pascal

```
function GetDeleteCount: TStringList;
```

C++

```
__fastcall TStringList GetDeleteCount();
```

Description

This is GetDeleteCount, a member of class TIBDatabaseInfo.

21.1.1.5.12 TIBDatabaseInfo.GetExpungeCount Method

Pascal

```
function GetExpungeCount: TStringList;
```

C++

```
__fastcall TStringList GetExpungeCount();
```

Description

This is GetExpungeCount, a member of class TIBDatabaseInfo.

21.1.1.5.13 TIBDatabaseInfo.GetFetches Method

Pascal

```
function GetFetches: Long;
```

C++

```
__fastcall Long GetFetches();
```

Description

This is GetFetches, a member of class TIBDatabaseInfo.

21.1.1.5.14 TIBDatabaseInfo.GetForcedWrites Method

Pascal

```
function GetForcedWrites: Long;
```

C++

```
__fastcall Long GetForcedWrites();
```

Description

This is GetForcedWrites, a member of class TIBDatabaseInfo.

21.1.1.5.15 TIBDatabaseInfo.GetInsertCount Method

Pascal

```
function GetInsertCount: TStringList;
```

C++

```
__fastcall TStringList GetInsertCount();
```

Description

This is GetInsertCount, a member of class TIBDatabaseInfo.

21.1.1.5.16 TIBDatabaseInfo.GetLongDatabaseInfo Method

Returns a specified long integer that describes some aspect of the associated database.

Pascal

```
function GetLongDatabaseInfo(DatabaseInfoCommand: Integer): Long;
```

C++

```
__fastcall Long GetLongDatabaseInfo(int DatabaseInfoCommand);
```

Description

TIBDatabaseInfo (see page 3772) uses GetLongDatabaseInfo internally to fetch information that can be expressed as a long integer. For example, this method provides the underlying implementation for fetching the version number or page size. It is exposed as a public member so that as the underlying _DatabaseInfoCommand interface expands its capabilities, TIBDatabaseInfo (see page 3772) can still be used to fetch information for which there is no specified property.

DatabaseInfoCommand is an index that identifies the information to fetch. It is one of the constants declared in the IBHeader unit.

Long returns desired information.

21.1.1.5.17 TIBDatabaseInfo.GetMarks Method

Pascal

```
function GetMarks: Long;
```

C++

```
__fastcall Long GetMarks();
```

Description

This is GetMarks, a member of class TIBDatabaseInfo.

21.1.1.5.18 TIBDatabaseInfo.GetMaxMemory Method

Pascal

```
function GetMaxMemory: Long;
```

C++

```
__fastcall Long GetMaxMemory();
```

Description

This is GetMaxMemory, a member of class TIBDatabaseInfo.

21.1.1.5.19 TIBDatabaseInfo.GetNoReserve Method

Pascal

```
function GetNoReserve: Long;
```

C++

```
__fastcall Long GetNoReserve();
```

Description

This is GetNoReserve, a member of class TIBDatabaseInfo.

21.1.1.5.20 TIBDatabaseInfo.GetNumBuffers Method

Pascal

```
function GetNumBuffers: Long;
```

C++

```
__fastcall Long GetNumBuffers();
```

Description

This is GetNumBuffers, a member of class TIBDatabaseInfo.

21.1.1.5.21 TIBDatabaseInfo.GetODSMajorVersion Method

Pascal

```
function GetODSMajorVersion: Long;
```

C++

```
__fastcall Long GetODSMajorVersion();
```

Description

This is GetODSMajorVersion, a member of class TIBDatabaseInfo.

21.1.1.5.22 TIBDatabaseInfo.GetODSMinorVersion Method

Pascal

```
function GetODSMinorVersion: Long;
```

C++

```
__fastcall Long GetODSMinorVersion();
```

Description

This is GetODSMinorVersion, a member of class TIBDatabaseInfo.

21.1.1.5.23 TIBDatabaseInfo.GetOperationCounts Method

Pascal

```
function GetOperationCounts(DBInfoCommand: Integer; FOperation: TStringList): TStringList;
```

C++

```
__fastcall TStringList GetOperationCounts(int DBInfoCommand, TStringList FOperation);
```

Description

This is GetOperationCounts, a member of class TIBDatabaseInfo.

21.1.1.5.24 TIBDatabaseInfo.GetPageSize Method

Pascal

```
function GetPageSize: Long;
```

C++

```
__fastcall Long GetPageSize();
```

Description

This is GetPageSize, a member of class TIBDatabaseInfo.

21.1.1.5.25 TIBDatabaseInfo.GetPurgeCount Method

Pascal

```
function GetPurgeCount: TStringList;
```

C++

```
__fastcall TStringList GetPurgeCount();
```

Description

This is GetPurgeCount, a member of class TIBDatabaseInfo.

21.1.1.5.26 TIBDatabaseInfo.GetReadIdxCount Method

Pascal

```
function GetReadIdxCount: TStringList;
```

C++

```
__fastcall TStringList GetReadIdxCount();
```

Description

This is GetReadIdxCount, a member of class TIBDatabaseInfo.

21.1.1.5.27 TIBDatabaseInfo.GetReadOnly Method

Pascal

```
function GetReadOnly: Long;
```

C++

```
__fastcall Long GetReadOnly();
```

Description

This is GetReadOnly, a member of class TIBDatabaseInfo.

21.1.1.5.28 TIBDatabaseInfo.GetReads Method

Pascal

```
function GetReads: Long;
```

C++

```
__fastcall Long GetReads();
```

Description

This is GetReads, a member of class TIBDatabaseInfo.

21.1.1.5.29 TIBDatabaseInfo.GetReadSeqCount Method

Pascal

```
function GetReadSeqCount: TStringList;
```

C++

```
__fastcall TStringList GetReadSeqCount();
```

Description

This is GetReadSeqCount, a member of class TIBDatabaseInfo.

21.1.1.5.30 TIBDatabaseInfo.GetStringDatabaseInfo Method

Pascal

```
function GetStringDatabaseInfo(DatabaseInfoCommand: Integer): String;
```

C++

```
__fastcall UnicodeString GetStringDatabaseInfo(int DatabaseInfoCommand);
```

Description

This is GetStringDatabaseInfo, a member of class TIBDatabaseInfo.

21.1.1.5.31 TIBDatabaseInfo.GetSweepInterval Method

Pascal

```
function GetSweepInterval: Long;
```

C++

```
__fastcall Long GetSweepInterval();
```

Description

This is GetSweepInterval, a member of class TIBDatabaseInfo.

21.1.1.5.32 TIBDatabaseInfo.GetUpdateCount Method

Pascal

```
function GetUpdateCount: TStringList;
```

C++

```
__fastcall TStringList GetUpdateCount();
```

Description

This is GetUpdateCount, a member of class TIBDatabaseInfo.

21.1.1.5.33 TIBDatabaseInfo.GetUserNames Method

Pascal

```
function GetUserNames: TStringList;
```

C++

```
__fastcall TStringList GetUserNames();
```

Description

This is GetUserNames, a member of class TIBDatabaseInfo.

21.1.1.5.34 TIBDatabaseInfo.GetVersion Method

Pascal

```
function GetVersion: String;
```

C++

```
__fastcall UnicodeString GetVersion();
```

Description

This is GetVersion, a member of class TIBDatabaseInfo.

21.1.1.5.35 TIBDatabaseInfo.GetWrites Method

Pascal

```
function GetWrites: Long;
```

C++

```
__fastcall Long GetWrites();
```

Description

This is GetWrites, a member of class TIBDatabaseInfo.

21.1.1.6 TIBDatabaseInfo Properties

The properties of the TIBDatabaseInfo class are listed here.

Public Properties

	Name	Description
	Allocation (see page 3787)	Returns the number of database pages allocated.
	BackoutCount (see page 3787)	Returns the number of removals of a version of a record.
	BaseLevel (see page 3788)	Returns the database version number.
	CurrentMemory (see page 3788)	Returns the amount of server memory currently in use.
	DBFileName (see page 3788)	Returns the database filename.
	DBImplementationClass (see page 3789)	Returns the database implementation class number.
	DBImplementationNo (see page 3789)	Returns the database implementation number.
	DBSiteName (see page 3789)	Returns the database site name.
	DBSQLDialect (see page 3789)	Returns the SQL dialect.
	DeleteCount (see page 3790)	Returns the number of database deletes since the database was last attached.
	ExpungeCount (see page 3790)	Returns the number of removals of a record and all of its ancestors.
	Fetches (see page 3790)	Returns the number of reads from the memory buffer cache.

	ForcedWrites (see page 3790)	Returns the mode in which database writes are performed.
	InsertCount (see page 3791)	Returns number of inserts into the database since the database was last attached.
	Marks (see page 3791)	Returns the number of writes to the memory buffer cache.
	MaxMemory (see page 3791)	Returns the maximum amount of memory used at one time since the first process attached to the database.
	NoReserve (see page 3791)	Returns whether or not space is reserved on each database page for holding backup versions of modified records.
	NumBuffers (see page 3792)	Returns the number of memory buffers currently allocated.
	ODSMajorVersion (see page 3792)	Returns the on disk structure (ODS) major version number.
	ODSMinorVersion (see page 3792)	Returns the on disk structure (ODS) minor version number.
	PageSize (see page 3792)	Returns the number of bytes per page of the attached database.
	PurgeCount (see page 3793)	Returns the number of removals of fully mature records from the database.
	ReadIdxCount (see page 3793)	Returns the number of reads done via an index since the database was last attached.
	ReadOnly (see page 3793)	Indicates whether or not the database is read only.
	Reads (see page 3793)	Returns the number of page reads from the database.
	ReadSeqCount (see page 3794)	Returns the number of sequential database reads done on each table since the database was last attached.
	SweepInterval (see page 3794)	Returns the number of transactions that are committed between 'sweeps.'
	UpdateCount (see page 3794)	Returns the number of database updates since the database was last attached.
	UserNames (see page 3795)	Returns the names of all users currently attached to the database.
	Version (see page 3795)	Returns the version of the database implementation.
	Writes (see page 3795)	Returns the number of page writes to the database.

Published Properties

	Name	Description
	Database (see page 3788)	Sets or returns the database.

21.1.1.6.1 TIBDatabaseInfo.Allocation Property

Returns the number of database pages allocated.

Pascal

```
property Allocation: Long;
```

C++

```
__property Long Allocation;
```

Description

Use Allocation to return the number of database pages allocated.

21.1.1.6.2 TIBDatabaseInfo.BackoutCount Property

Returns the number of removals of a version of a record.

Pascal

```
property BackoutCount: TStringList;
```

C++

```
__property TStringList BackoutCount;
```

Description

Use BackoutCount to determine the number of times a version of a database record has been removed.

21.1.1.6.3 TIBDatabaseInfo.BaseLevel Property

Returns the database version number.

Pascal

```
property BaseLevel: Long;
```

C++

```
__property Long BaseLevel;
```

Description

Use BaseLevel to return the database version number, which consists of 1 byte containing the number 1, and 1 byte containing the version number.

21.1.1.6.4 TIBDatabaseInfo.CurrentMemory Property

Returns the amount of server memory currently in use.

Pascal

```
property CurrentMemory: Long;
```

C++

```
__property Long CurrentMemory;
```

Description

Use CurrentMemory to return the amount of server memory (in bytes) currently in use.

21.1.1.6.5 TIBDatabaseInfo.Database Property

Sets or returns the database.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to set or return the database on which information is being returned.

21.1.1.6.6 TIBDatabaseInfo.DBFileName Property

Returns the database filename.

Pascal

```
property DBFileName: String;
```

C++

```
__property UnicodeString DBFileName;
```

Description

Use DBFileName to return the database filename.

21.1.1.6.7 TIBDatabaseInfo.DBImplementationClass Property

Returns the database implementation class number.

Pascal

```
property DBImplementationClass: Long;
```

C++

```
__property Long DBImplementationClass;
```

Description

Use DBImplementationClass to return the database implementation class number, either 1 or 12.

21.1.1.6.8 TIBDatabaseInfo.DBImplementationNo Property

Returns the database implementation number.

Pascal

```
property DBImplementationNo: Long;
```

C++

```
__property Long DBImplementationNo;
```

Description

Use DBImplementationNo to return the database implementation number.

21.1.1.6.9 TIBDatabaseInfo.DBSiteName Property

Returns the database site name.

Pascal

```
property DBSiteName: String;
```

C++

```
__property UnicodeString DBSiteName;
```

Description

Use DBSiteName to return the database site name.

21.1.1.6.10 TIBDatabaseInfo.DBSQLDialect Property

Returns the SQL dialect.

Pascal

```
property DBSQLDialect: Long;
```

C++

```
__property Long DBSQLDialect;
```

Description

Use DBSQLDialect to return the SQL dialect.

21.1.1.6.11 TIBDatabaseInfo.DeleteCount Property

Returns the number of database deletes since the database was last attached.

Pascal

```
property DeleteCount: TStringList;
```

C++

```
__property TStringList DeleteCount;
```

Description

Use DeleteCount to return the number of database deletes since the database was last attached.

21.1.1.6.12 TIBDatabaseInfo.ExpungeCount Property

Returns the number of removals of a record and all of its ancestors.

Pascal

```
property ExpungeCount: TStringList;
```

C++

```
__property TStringList ExpungeCount;
```

Description

Use ExpungeCount to return the number of removals of a record and all of its ancestors for records whose deletions have been committed.

21.1.1.6.13 TIBDatabaseInfo.Fetches Property

Returns the number of reads from the memory buffer cache.

Pascal

```
property Fetches: Long;
```

C++

```
__property Long Fetches;
```

Description

Use Fetches to return the number of reads from the memory buffer cache.

21.1.1.6.14 TIBDatabaseInfo.ForcedWrites Property

Returns the mode in which database writes are performed.

Pascal

```
property ForcedWrites: Long;
```

C++

```
__property Long ForcedWrites;
```

Description

Use ForcedWrites to return the number specifying the mode in which database writes are performed. ForcedWrites returns 0 for asynchronous mode, or returns 1 for synchronous mode

21.1.1.6.15 TIBDatabaseInfo.InsertCount Property

Returns number of inserts into the database since the database was last attached.

Pascal

```
property InsertCount: TStringList;
```

C++

```
__property TStringList InsertCount;
```

Description

Use InsertCount to return the number of inserts into the database since the database was last attached.

21.1.1.6.16 TIBDatabaseInfo.Marks Property

Returns the number of writes to the memory buffer cache.

Pascal

```
property Marks: Long;
```

C++

```
__property Long Marks;
```

Description

Use Marks to return the number of writes to the memory buffer cache.

21.1.1.6.17 TIBDatabaseInfo.MaxMemory Property

Returns the maximum amount of memory used at one time since the first process attached to the database.

Pascal

```
property MaxMemory: Long;
```

C++

```
__property Long MaxMemory;
```

Description

Use MaxMemory to return in bytes the maximum amount of memory used at one time since the first process attached to the database.

21.1.1.6.18 TIBDatabaseInfo.NoReserve Property

Returns whether or not space is reserved on each database page for holding backup versions of modified records.

Pascal

```
property NoReserve: Long;
```

C++

```
__property Long NoReserve;
```

Description

Use NoReserve to return whether or not space is reserved on each database page for holding backup versions of modified records. NoReserve will return 0 to indicate that space is reserved (the default) or 1 to indicate that no space is reserved.

21.1.1.6.19 TIBDatabaseInfo.NumBuffers Property

Returns the number of memory buffers currently allocated.

Pascal

```
property NumBuffers: Long;
```

C++

```
__property Long NumBuffers;
```

Description

Use NumBuffers to return the number of memory buffers currently allocated.

21.1.1.6.20 TIBDatabaseInfo.ODSMajorVersion Property

Returns the on disk structure (ODS) major version number.

Pascal

```
property ODSMajorVersion: Long;
```

C++

```
__property Long ODSMajorVersion;
```

Description

Use ODSMajorVersion to return the ODS major version number for the database. Databases with different major version numbers have different physical layouts.

A database engine can access only databases with a particular ODS major version number. Attaching to a database with a different ODS number results in an error.

21.1.1.6.21 TIBDatabaseInfo.ODSMinorVersion Property

Returns the on disk structure (ODS) minor version number.

Pascal

```
property ODSMinorVersion: Long;
```

C++

```
__property Long ODSMinorVersion;
```

Description

Use ODSMinorVersion to return the (ODS) minor version number. An increase in a minor version number indicates a non-structural change, one that still allows the database to be accessed by databases with the same major version number but possibly different minor version numbers.

21.1.1.6.22 TIBDatabaseInfo.PageSize Property

Returns the number of bytes per page of the attached database.

Pascal

```
property PageSize: Long;
```

C++

```
__property Long PageSize;
```

Description

Use PageSize to return the number of bytes per page of the attached database. Use with Allocation (see page 3787) to determine the size of the database.

21.1.1.6.23 TIBDatabaseInfo.PurgeCount Property

Returns the number of removals of fully mature records from the database.

Pascal

```
property PurgeCount: TStringList;
```

C++

```
__property TStringList PurgeCount;
```

Description

Use PurgeCount to return the number of removals of fully mature records (that is, records committed, resulting in older versions no longer being needed) from the database.

21.1.1.6.24 TIBDatabaseInfo.ReadIdxCount Property

Returns the number of reads done via an index since the database was last attached.

Pascal

```
property ReadIdxCount: TStringList;
```

C++

```
__property TStringList ReadIdxCount;
```

Description

Use ReadIdxCount to return the number of reads done via an index since the database was last attached.

21.1.1.6.25 TIBDatabaseInfo.ReadOnly Property

Indicates whether or not the database is read only.

Pascal

```
property ReadOnly: Long;
```

C++

```
__property Long ReadOnly;
```

Description

Use ReadOnly to determine whether the database is read only or not. ReadOnly returns 1 if the database is read-write and 0 if it is read only.

Read-only databases are an InterBase 6 feature.

21.1.1.6.26 TIBDatabaseInfo.Reads Property

Returns the number of page reads from the database.

Pascal

```
property Reads: Long;
```

C++

```
__property Long Reads;
```

Description

Use Reads to return the number of page reads from the database since the current database was first attached, that is, an aggregate of all reads done by all attached processes, rather than the number of reads done for the calling program since it attached to the database.

21.1.1.6.27 TIBDatabaseInfo.ReadSeqCount Property

Returns the number of sequential database reads done on each table since the database was last attached.

Pascal

```
property ReadSeqCount: TStringList;
```

C++

```
__property TStringList ReadSeqCount;
```

Description

Use ReadSeqCount to return the number of sequential database reads (that is, the number of sequential table scans) done on each table since the database was last attached

21.1.1.6.28 TIBDatabaseInfo.SweepInterval Property

Returns the number of transactions that are committed between 'sweeps.'

Pascal

```
property SweepInterval: Long;
```

C++

```
__property Long SweepInterval;
```

Description

Use SweepInterval to return the number of transactions that are committed between 'sweeps' to remove database record versions that are no longer needed.

21.1.1.6.29 TIBDatabaseInfo.UpdateCount Property

Returns the number of database updates since the database was last attached.

Pascal

```
property UpdateCount: TStringList;
```

C++

```
__property TStringList UpdateCount;
```

Description

Use UpdateCount to return the number of database updates since the database was last attached.

21.1.1.6.30 TIBDatabaseInfo.UserNames Property

Returns the names of all users currently attached to the database.

Pascal

```
property UserNames: TStringList;
```

C++

```
__property TStringList UserNames;
```

Description

Use UserNames to return the names of all users currently attached to the database.

21.1.1.6.31 TIBDatabaseInfo.Version Property

Returns the version of the database implementation.

Pascal

```
property Version: String;
```

C++

```
__property UnicodeString Version;
```

Description

Use Version to return the version identification string of the database implementation.

21.1.1.6.32 TIBDatabaseInfo.Writes Property

Returns the number of page writes to the database.

Pascal

```
property Writes: Long;
```

C++

```
__property Long Writes;
```

Description

Use Writes to return the number of page writes to the current database since it was first attached by any process; that is, an aggregate of all write done by all attached processes, rather than the number of writes done for the calling program since it attached to the database.

22 IBEvents Namespace

This is namespace IBEvents.

Topics

Name	Description
Classes (see page 3798)	The following table lists classes in this documentation.
Types (see page 3806)	The following table lists types in this documentation.

Classes

	Name	Description
	TIBEvents (see page 3798)	TIBEvents enables applications to respond to posted events.

Types

Name	Description
TErrorEvent (see page 3806)	TErrorEvent is the type of event handlers that respond to exceptions that occur when monitoring InterBase events.
TEventAlert (see page 3806)	TEventAlert is the type of event handlers that respond to InterBase events.

22.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBEvents (see page 3798)	TIBEvents enables applications to respond to posted events.

22.1.1 TIBEvents Class

TIBEvents enables applications to respond to posted events.

Class Hierarchy

Pascal

```
TIBEvents = class(TComponent, IIBEventNotifier);
```

C++

```
class TIBEvents : public TComponent, public IIBEventNotifier;
```

File

IBEvents

Description

Use a TIBEvents component to allow your application to register interest in, and asynchronously handle, events posted by an InterBase server. The InterBase event allows applications to respond to actions and database changes made by other, concurrently running applications, without having to resort to polling the database on a regular basis, or communicating directly with the other applications.

In essence, the TIBEvents component allows an application to say 'I want to be informed when events X, Y and Z occur.' When any of the requested events does occur, the InterBase server notifies the application and OnEventAlert (see page 3805) is called.

To use TIBEvents, set the Database (see page 3803) property to the component that connects to the InterBase server. Then, use the Events (see page 3804) property to indicate the events to which you want to respond. You can set the AutoRegister (see page 3803) property to true if you want TIBEvents to automatically register interest in its events when the database component opens a connection, or set the Registered (see page 3804) property to true after the connection is open to explicitly register your interest. Finally, write an OnEventAlert (see page 3805) event handler to respond when the InterBase event handler informs TIBEvents that an event occurred.

22.1.1.1 TIBEvents Members

The following tables list the members exposed by TIBEvents.

Protected Methods

	Name	Description
	EventChange (see page 3800)	This is EventChange, a member of class TIBEvents.
	GetNativeHandle (see page 3801)	This is GetNativeHandle, a member of class TIBEvents.
	Notification (see page 3801)	Forwards notification messages to all owned components.
	ThreadEnded (see page 3802)	This is ThreadEnded, a member of class TIBEvents.
	ValidateDatabase (see page 3803)	This is ValidateDatabase, a member of class TIBEvents.

Public Methods

	Name	Description
	Create (see page 3799)	Creates an instance of TIBEvents (see page 3798).
	Destroy (see page 3800)	Destroys an instance of TIBEvents (see page 3798).
	GetAutoRegister (see page 3800)	Returns the value of the AutoRegister (see page 3803) property.
	RegisterEvents (see page 3801)	Registers interest in the events listed by the Events (see page 3804) property.
	SetAutoRegister (see page 3802)	Sets the AutoRegister (see page 3803) property.
	UnRegisterEvents (see page 3802)	Unregisters interest in the events listed by the Events (see page 3804) property.

Public Properties

	Name	Description
	NativeHandle (see page 3804)	Supplies the InterBase handle for the database on which events occur.

Published Events

	Name	Description
	OnError (see page 3805)	Occurs when an exception is caught while monitoring events.
	OnEventAlert (see page 3805)	Occurs when an InterBase event is received.

Published Properties

	Name	Description
	AutoRegister (see page 3803)	Specifies the AutoRegister on which to perform event alerter tasks.
	Database (see page 3803)	Specifies the database on which to perform event alerter tasks.
	Events (see page 3804)	Specifies the events to which TIBEvents (see page 3798) responds.
	Registered (see page 3804)	Indicates whether the events in the Events (see page 3804) property are registered.

22.1.1.2 TIBEvents.Create Constructor

Creates an instance of TIBEvents (see page 3798).

Pascal

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

C++

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

Description

Call Create to create an instance of TIBEvents (see page 3798). TIBEvents (see page 3798) objects placed on a form or data module at design time are created automatically.

AOwner specifies the object (typically a form or data module) that is responsible for freeing the TIBEvents (see page 3798) instance.

22.1.1.3 TIBEvents.Destroy Destructor

Destroys an instance of TIBEvents (see page 3798).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBEvents();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBEvents (see page 3798) object is not nil and only then calls Destroy. Destroy unregisters the events and then frees them.

22.1.1.4 TIBEvents Methods

The methods of the TIBEvents class are listed here.

Protected Methods

	Name	Description
	EventChange (see page 3800)	This is EventChange, a member of class TIBEvents.
	GetNativeHandle (see page 3801)	This is GetNativeHandle, a member of class TIBEvents.
	Notification (see page 3801)	Forwards notification messages to all owned components.
	ThreadEnded (see page 3802)	This is ThreadEnded, a member of class TIBEvents.
	ValidateDatabase (see page 3803)	This is ValidateDatabase, a member of class TIBEvents.

Public Methods

	Name	Description
	GetAutoRegister (see page 3800)	Returns the value of the AutoRegister (see page 3803) property.
	RegisterEvents (see page 3801)	Registers interest in the events listed by the Events (see page 3804) property.
	SetAutoRegister (see page 3802)	Sets the AutoRegister (see page 3803) property.
	UnRegisterEvents (see page 3802)	Unregisters interest in the events listed by the Events (see page 3804) property.

22.1.1.4.1 TIBEvents.EventChange Method

Pascal

```
procedure EventChange(Sender: TObject); virtual;
```

C++

```
virtual __fastcall EventChange(TObject * Sender);
```

Description

This is EventChange, a member of class TIBEvents.

22.1.1.4.2 TIBEvents.GetAutoRegister Method

Returns the value of the AutoRegister (see page 3803) property.

Pascal

```
function GetAutoRegister: Boolean;
```

C++

```
__fastcall Boolean GetAutoRegister();
```

Description

GetAutoRegister is the read implementation of the AutoRegister (see page 3803) property.

22.1.1.4.3 TIBEvents.GetNativeHandle Method

Pascal

```
function GetNativeHandle: TISC_DB_HANDLE; virtual;
```

C++

```
virtual __fastcall TISC_DB_HANDLE GetNativeHandle();
```

Description

This is GetNativeHandle, a member of class TIBEvents.

22.1.1.4.4 TIBEvents.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

22.1.1.4.5 TIBEvents.RegisterEvents Method

Registers interest in the events listed by the Events (see page 3804) property.

Pascal

```
procedure RegisterEvents; virtual;
```

C++

```
virtual __fastcall RegisterEvents();
```

Description

Call RegisterEvents to register interest in the events listed in the Events (see page 3804) property with the InterBase Event Manager. Once this interest is registered, the InterBase Event Manager generates an OnEventAlert (see page 3805) event when any of the registered events occurs.

Note: RegisterEvents is called automatically when you set the Registered (see page 3804) property to true or when the AutoRegister (see page 3803) property is true and the database connection is open.

22.1.1.4.6 TIBEvents.SetAutoRegister Method

Sets the AutoRegister (see page 3803) property.

Pascal

```
procedure SetAutoRegister(const Value: Boolean);
```

C++

```
__fastcall SetAutoRegister(const Boolean Value);
```

Description

SetAutoRegister is the write implementation method for the AutoRegister (see page 3803) property.

Value is the new value for the AutoRegister (see page 3803) property.

When SetAutoRegister changes the AutoRegister (see page 3803) property to true, it checks whether the database connection is already open. If so, it calls RegisterEvents (see page 3801) to register the events.

22.1.1.4.7 TIBEvents.ThreadEnded Method

Pascal

```
procedure ThreadEnded(Sender: TObject); virtual;
```

C++

```
virtual __fastcall ThreadEnded(TObject * Sender);
```

Description

This is ThreadEnded, a member of class TIBEvents.

22.1.1.4.8 TIBEvents.UnRegisterEvents Method

Unregisters interest in the events listed by the Events (see page 3804) property.

Pascal

```
procedure UnRegisterEvents; virtual;
```

C++

```
virtual __fastcall UnRegisterEvents();
```

Description

Call UnRegisterEvents to unregister interest in the events in the Events (see page 3804) list. When the database closes a connection, or when the IBEvents (see page 3797) component is destroyed, UnRegisterEvents is called automatically.

22.1.1.4.9 TIBEvents.ValidateDatabase Method

Pascal

```
procedure ValidateDatabase(DataBase: TIBDatabase); virtual;
```

C++

```
virtual __fastcall ValidateDatabase(TIBDatabase DataBase);
```

Description

This is ValidateDatabase, a member of class TIBEvents.

22.1.1.5 TIBEvents Properties

The properties of the TIBEvents class are listed here.

Public Properties

	Name	Description
	NativeHandle (see page 3804)	Supplies the InterBase handle for the database on which events occur.

Published Properties

	Name	Description
	AutoRegister (see page 3803)	Specifies the AutoRegister on which to perform event alerter tasks.
	Database (see page 3803)	Specifies the database on which to perform event alerter tasks.
	Events (see page 3804)	Specifies the events to which TIBEvents (see page 3798) responds.
	Registered (see page 3804)	Indicates whether the events in the Events (see page 3804) property are registered.

22.1.1.5.1 TIBEvents.AutoRegister Property

Specifies the AutoRegister on which to perform event alerter tasks.

Pascal

```
property AutoRegister: Boolean;
```

C++

```
__property Boolean AutoRegister;
```

Description

When AutoRegister is true, the database automatically registers this component's events whenever it opens a connection to the InterBase server. When AutoRegister is false, you must explicitly register the events by setting the Registered (see page 3804) property to true (or calling RegisterEvents (see page 3801)) after the database connection is opened.

Note: The database always unregisters the events when it closes a connection. By setting AutoRegister to true, you do not need to explicitly re-register interest in the events when the database reopens a connection.

22.1.1.5.2 TIBEvents.Database Property

Specifies the database on which to perform event alerter tasks.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to specify the component that manages a connection to the InterBase server. This connection must be open for TIBEvents (see page 3798) to register an interest in its events with the InterBase Event manager on that server.

When you set the Database property, TIBEvents (see page 3798) automatically calls the database component's AddEventNotifier method so that it can register this component's events when it opens a connection (if AutoRegister (see page 3803) is true), and so that it can unregister this component's events before closing the connection.

22.1.1.5.3 TIBEvents.Events Property

Specifies the events to which TIBEvents (see page 3798) responds.

Pascal

```
property Events: TStrings;
```

C++

```
__property TStrings Events;
```

Description

Use the Events property to list events to which the IBEvents (see page 3797) component responds.

To add an event use code similar to the following:

```
Delphi: IBEvents.Events.Add('EVENT_NAME')
    C++: IBEvents->Events->Add("EVENT_NAME")
```

Note: Event names are case-sensitive.

22.1.1.5.4 TIBEvents.NativeHandle Property

Supplies the InterBase handle for the database on which events occur.

Pascal

```
property NativeHandle: TISC_DB_HANDLE;
```

C++

```
__property TISC_DB_HANDLE NativeHandle;
```

Description

Read NativeHandle to get the InterBase handle for the server on which the events occur. This property can be used when working with the InterBase API.

22.1.1.5.5 TIBEvents.Registered Property

Indicates whether the events in the Events (see page 3804) property are registered.

Pascal

```
property Registered: Boolean;
```

C++

```
__property Boolean Registered;
```

Description

Set Registered to true to call RegisterEvents (see page 3801), which registers the events listed by the Events (see page 3804) property. Read Registered to determine whether TIBEvents (see page 3798) is currently registered (that is, if the InterBase Event manager is notifying this component when any of its events occur.)

22.1.1.6 TIBEvents Events

The events of the TIBEvents class are listed here.

Published Events

	Name	Description
	OnError (see page 3805)	Occurs when an exception is caught while monitoring events.
	OnEventAlert (see page 3805)	Occurs when an InterBase event is received.

22.1.1.6.1 TIBEvents.OnError Event

Occurs when an exception is caught while monitoring events.

Pascal

```
property OnError: TErrorEvent;
```

C++

```
__property TErrorEvent OnError;
```

Description

Write an OnError event handler to respond when TIBEvents (see page 3798) encounters an exception in the process of monitoring events.

22.1.1.6.2 TIBEvents.OnEventAlert Event

Occurs when an InterBase event is received.

Pascal

```
property OnEventAlert: TEventAlert;
```

C++

```
__property TEventAlert OnEventAlert;
```

Description

Write an OnEventAlert event handler to take specific actions when an InterBase event is received.

In the event handler, set CancelAlerts to true to cancel interest in any further events. To start receiving events again, call the GetAutoRegister (see page 3800) method. You cannot call RegisterEvents (see page 3801), UnRegisterEvents (see page 3802), GetAutoRegister (see page 3800) or SetAutoRegister (see page 3802) from within an OnEventAlert event handler.

OnEventAlert runs as a separate thread to allow for true asynchronous event processing, however, the IBEvents (see page 3797) component provides synchronization code to ensure that only one OnEventAlert event handler executes at any one time.

22.2 Types

The following table lists types in this documentation.

Types

Name	Description
TErrorEvent (see page 3806)	TErrorEvent is the type of event handlers that respond to exceptions that occur when monitoring InterBase events.
TEventAlert (see page 3806)	TEventAlert is the type of event handlers that respond to InterBase events.

22.2.1 IBEvents.TErrorEvent Type

TErrorEvent is the type of event handlers that respond to exceptions that occur when monitoring InterBase events.

Pascal

```
TErrorEvent = procedure ( Sender: TObject; ErrorCode:integer) of object;
```

C++

```
( Sender: TObject; ErrorCode:integer) ( TErrorEvent)();
```

File

IBEvents

Description

Sender is the TIBEvents (see page 3798) component that encounters the exception.

If the exception is an EIBError (see page 3543) object, ErrorCode is the InterBase error code it represents. Otherwise, ErrorCode is 0.

22.2.2 IBEvents.TEventAlert Type

TEventAlert is the type of event handlers that respond to InterBase events.

Pascal

```
TEventAlert = procedure ( Sender: TObject; EventName: string; EventCount: longint; var
CancelAlerts: Boolean) of object;
```

C++

```
( Sender: TObject; EventName: string; EventCount: longint; var CancelAlerts: Boolean) (
TEventAlert)();
```

File

IBEvents

Description

Sender is the TIBEvents (see page 3798) object that is monitoring InterBase events.

EventName is the name of the most recently received event.

EventCount is the number of EventName events received since the last time the event handler was called.

CancelAlerts returns true if the TIBEvents (see page 3798) object should stop monitoring events, false if the event monitoring should continue.

23 IBExtract Namespace

This is namespace IBExtract.

Topics

Name	Description
Classes (see page 3811)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3823)	The following table lists structs, records, enums in this documentation.
Types (see page 3827)	The following table lists types in this documentation.
Constants (see page 3829)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIBExtract (see page 3811)	TIBExtract fetches metadata from an InterBase server.

Constants

Name	Description
ColumnTypes (see page 3830)	This is constant IBExtract.ColumnTypes.
FILE_cache (see page 3830)	This is constant IBExtract.FILE_cache.
FILE_conditional (see page 3830)	This is constant IBExtract.FILE_conditional.
FILE_inactive (see page 3831)	This is constant IBExtract.FILE_inactive.
FILE_manual (see page 3831)	This is constant IBExtract.FILE_manual.
FILE_shadow (see page 3831)	This is constant IBExtract.FILE_shadow.
IntegralSubtypes (see page 3832)	This is constant IBExtract.IntegralSubtypes.
LOG_default (see page 3832)	This is constant IBExtract.LOG_default.
LOG_overflow (see page 3832)	This is constant IBExtract.LOG_overflow.
LOG_raw (see page 3833)	This is constant IBExtract.LOG_raw.
LOG_serial (see page 3833)	This is constant IBExtract.LOG_serial.
MAX_INTSUBTYPES (see page 3833)	This is constant IBExtract.MAX_INTSUBTYPES.
MAXSUBTYPES (see page 3834)	This is constant IBExtract.MAXSUBTYPES.
obj_computed (see page 3834)	This is constant IBExtract.obj_computed.
obj_count (see page 3834)	This is constant IBExtract.obj_count.
obj_exception (see page 3835)	This is constant IBExtract.obj_exception.
obj_expression_index (see page 3835)	This is constant IBExtract.obj_expression_index.
obj_field (see page 3835)	This is constant IBExtract.obj_field.
obj_index (see page 3836)	This is constant IBExtract.obj_index.
obj_procedure (see page 3836)	This is constant IBExtract.obj_procedure.

obj_relation (see page 3836)	This is constant IBExtract.obj_relation.
obj_sql_role (see page 3837)	This is constant IBExtract.obj_sql_role.
obj_trigger (see page 3837)	This is constant IBExtract.obj_trigger.
obj_user (see page 3837)	This is constant IBExtract.obj_user.
obj_user_group (see page 3838)	This is constant IBExtract.obj_user_group.
obj_validation (see page 3838)	This is constant IBExtract.obj_validation.
obj_view (see page 3838)	This is constant IBExtract.obj_view.
ODS_VERSION10 (see page 3839)	This is constant IBExtract.ODS_VERSION10.
ODS_VERSION6 (see page 3839)	This is constant IBExtract.ODS_VERSION6.
ODS_VERSION7 (see page 3839)	This is constant IBExtract.ODS_VERSION7.
ODS_VERSION8 (see page 3840)	This is constant IBExtract.ODS_VERSION8.
ODS_VERSION9 (see page 3840)	This is constant IBExtract.ODS_VERSION9.
priv_DELETE (see page 3840)	This is constant IBExtract.priv_DELETE.
priv_EXECUTE (see page 3841)	This is constant IBExtract.priv_EXECUTE.
priv_INSERT (see page 3841)	This is constant IBExtract.priv_INSERT.
priv_REFERENCES (see page 3841)	This is constant IBExtract.priv_REFERENCES.
priv_SELECT (see page 3842)	This is constant IBExtract.priv_SELECT.
priv_UNKNOWN (see page 3842)	This is constant IBExtract.priv_UNKNOWN.
priv_UPDATE (see page 3842)	This is constant IBExtract.priv_UPDATE.
PrivTypes (see page 3843)	This is constant IBExtract.PrivTypes.
SubTypes (see page 3843)	This is constant IBExtract.SubTypes.
TriggerTypes (see page 3843)	This is constant IBExtract.TriggerTypes.

Structs, Records, Enums

	Name	Description
	TExtractObjectTypes (see page 3823)	TExtractObjectTypes indicates the type of metadata to fetch from an InterBase server.
	TExtractType (see page 3824)	This is record IBExtract.TExtractType.
	TPrivTypes (see page 3825)	This is record IBExtract.TPrivTypes.
	TSQLType (see page 3825)	This is record IBExtract.TSQLType.

Types

Name	Description
TExtractTypes (see page 3827)	TExtractType (see page 3824) and TExtractTypes indicate the scope of metadata to extract from an InterBase server.
TSQLTypes (see page 3828)	This is type IBExtract.TSQLTypes.

23.1 Classes

The following table lists classes in this documentation.

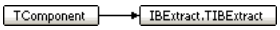
Classes

	Name	Description
	TIBExtract (see page 3811)	TIBExtract fetches metadata from an InterBase server.

23.1.1 TIBExtract Class

TIBExtract fetches metadata from an InterBase server.

Class Hierarchy



Pascal

```
TIBExtract = class(TComponent);
```

C++

```
class TIBExtract : public TComponent;
```

File

IBExtract

Description

Use a TIBExtract component to fetch metadata information such as a list of tables, views, roles, indexes, and so on. To use TIBExtract,

1. et the Database (see page 3821) and Transaction (see page 3822) properties to specify the database from which to fetch information and to provide a transaction to use when fetching that information.
2. all the ExtractObject (see page 3814) method, indicating the type of metadata you want to fetch.
3. ead the extracted information from the Items (see page 3821) property.

See Also

Getting Started with InterBase Express

23.1.1.1 TIBExtract Members

The following tables list the members exposed by TIBExtract.

Protected Methods

	Name	Description
	ExtractDDL (see page 3814)	This is ExtractDDL, a member of class TIBExtract.
	ExtractListTable (see page 3814)	This is ExtractListTable, a member of class TIBExtract.
	ExtractListView (see page 3814)	This is ExtractListView, a member of class TIBExtract.
	ListAllTables (see page 3816)	This is ListAllTables, a member of class TIBExtract.

	ListCheck (see page 3817)	This is ListCheck, a member of class TIBExtract.
	ListCreateDb (see page 3817)	This is ListCreateDb, a member of class TIBExtract.
	ListData (see page 3817)	This is ListData, a member of class TIBExtract.
	ListDomains (see page 3817)	This is ListDomains, a member of class TIBExtract.
	ListException (see page 3818)	This is ListException, a member of class TIBExtract.
	ListFilters (see page 3818)	This is ListFilters, a member of class TIBExtract.
	ListForeign (see page 3818)	This is ListForeign, a member of class TIBExtract.
	ListFunctions (see page 3818)	This is ListFunctions, a member of class TIBExtract.
	ListGenerators (see page 3818)	This is ListGenerators, a member of class TIBExtract.
	ListGrants (see page 3819)	This is ListGrants, a member of class TIBExtract.
	ListIndex (see page 3819)	This is ListIndex, a member of class TIBExtract.
	ListProcs (see page 3819)	This is ListProcs, a member of class TIBExtract.
	ListRoles (see page 3819)	This is ListRoles, a member of class TIBExtract.
	ListTriggers (see page 3820)	This is ListTriggers, a member of class TIBExtract.
	ListViews (see page 3820)	This is ListViews, a member of class TIBExtract.
	PrintSet (see page 3820)	This is PrintSet, a member of class TIBExtract.

Public Methods

	Name	Description
V	Create (see page 3812)	Creates an instance of TIBExtract (see page 3811).
V	Destroy (see page 3813)	Destroys an instance of TIBExtract (see page 3811).
	ExtractObject (see page 3814)	Fetches specified metadata from the InterBase server and writes it to the Items (see page 3821) property.
	GetArrayField (see page 3816)	Returns the dimensions of an array field as a printable string.
	GetCharacterSets (see page 3816)	Returns a specified character set and collation order as a printable string.
	GetFieldType (see page 3816)	Returns the full description of a specified field type as a printable string.
V	Notification (see page 3820)	Responds to notifications when components are created or freed.

Public Properties

	Name	Description
R	DatabaseInfo (see page 3821)	Represents information about the associated database.
R	Items (see page 3821)	Lists the information that the TIBExtract (see page 3811) object has fetched from the database server.

Published Properties

	Name	Description
	Database (see page 3821)	Specifies the database from which to fetch metadata.
	ShowSystem (see page 3822)	Reserved for future use.
	Transaction (see page 3822)	Specifies the transaction to use when fetching metadata from the server.

23.1.1.2 TIBExtract.Create Constructor

Creates an instance of TIBExtract (see page 3811).

Pascal

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

C++

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

Description

Call Create to create an instance of TIBExtract (see page 3811) at runtime. TIBExtract (see page 3811) components added to forms or data modules at design time are created automatically.

AOwner specifies the component that is responsible for freeing the TIBExtract (see page 3811) instance. If AOwner is a

TIBDatabase component, it is used to initialize the Database (see page 3821) property.

23.1.1.3 TIBExtract.Destroy Destructor

Destroys an instance of TIBExtract (see page 3811).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBExtract();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBExtract (see page 3811) object is not already freed and only then calls Destroy.

There is no need to explicitly free a TIBExtract (see page 3811) instance that is created with an Owner. The component's Owner handles the task of freeing it.

23.1.1.4 TIBExtract Methods

The methods of the TIBExtract class are listed here.

Protected Methods

	Name	Description
	ExtractDDL (see page 3814)	This is ExtractDDL, a member of class TIBExtract.
	ExtractListTable (see page 3814)	This is ExtractListTable, a member of class TIBExtract.
	ExtractListView (see page 3814)	This is ExtractListView, a member of class TIBExtract.
	ListAllTables (see page 3816)	This is ListAllTables, a member of class TIBExtract.
	ListCheck (see page 3817)	This is ListCheck, a member of class TIBExtract.
	ListCreateDb (see page 3817)	This is ListCreateDb, a member of class TIBExtract.
	ListData (see page 3817)	This is ListData, a member of class TIBExtract.
	ListDomains (see page 3817)	This is ListDomains, a member of class TIBExtract.
	ListException (see page 3818)	This is ListException, a member of class TIBExtract.
	ListFilters (see page 3818)	This is ListFilters, a member of class TIBExtract.
	ListForeign (see page 3818)	This is ListForeign, a member of class TIBExtract.
	ListFunctions (see page 3818)	This is ListFunctions, a member of class TIBExtract.
	ListGenerators (see page 3818)	This is ListGenerators, a member of class TIBExtract.
	ListGrants (see page 3819)	This is ListGrants, a member of class TIBExtract.
	ListIndex (see page 3819)	This is ListIndex, a member of class TIBExtract.
	ListProcs (see page 3819)	This is ListProcs, a member of class TIBExtract.
	ListRoles (see page 3819)	This is ListRoles, a member of class TIBExtract.
	ListTriggers (see page 3820)	This is ListTriggers, a member of class TIBExtract.
	ListViews (see page 3820)	This is ListViews, a member of class TIBExtract.
	PrintSet (see page 3820)	This is PrintSet, a member of class TIBExtract.

Public Methods

	Name	Description
	ExtractObject (see page 3814)	Fetches specified metadata from the InterBase server and writes it to the Items (see page 3821) property.
	GetArrayField (see page 3816)	Returns the dimensions of an array field as a printable string.
	GetCharacterSets (see page 3816)	Returns a specified character set and collation order as a printable string.
	GetFieldType (see page 3816)	Returns the full description of a specified field type as a printable string.

	Notification (see page 3820)	Responds to notifications when components are created or freed.

23.1.1.4.1 TIBExtract.ExtractDDL Method

Pascal

```
function ExtractDDL(Flag: Boolean; TableName: String): Boolean;
```

C++

```
__fastcall Boolean ExtractDDL(Boolean Flag, UnicodeString TableName);
```

Description

This is ExtractDDL, a member of class TIBExtract.

23.1.1.4.2 TIBExtract.ExtractListTable Method

Pascal

```
function ExtractListTable(RelationName: String; NewName: String; DomainFlag: Boolean): Boolean;
```

C++

```
__fastcall Boolean ExtractListTable(UnicodeString RelationName, UnicodeString NewName, Boolean
DomainFlag);
```

Description

This is ExtractListTable, a member of class TIBExtract.

23.1.1.4.3 TIBExtract.ExtractListView Method

Pascal

```
procedure ExtractListView(ViewName: String);
```

C++

```
__fastcall ExtractListView(UnicodeString ViewName);
```

Description

This is ExtractListView, a member of class TIBExtract.

23.1.1.4.4 TIBExtract.ExtractObject Method

Fetches specified metadata from the InterBase server and writes it to the Items (see page 3821) property.

Pascal

```
procedure ExtractObject(ObjectType: TExtractObjectTypes; ObjectName: String = '';
ExtractTypes: TExtractTypes = []);
```

C++

```
__fastcall ExtractObject(TExtractObjectTypes ObjectType, UnicodeString ObjectName = '',
TExtractTypes ExtractTypes = []);
```

Description

Call ExtractObject to fetch metadata from the database specified by the Database (see page 3821) property and write it to the Items (see page 3821) property.

ObjectType indicates the desired type of metadata. Its meaning is modified by the ObjectName and ExtractTypes parameters, as

indicated in the following table:

Value Meaning

eoDatabase Lists all objects defined on the database. This is the list that would be fetched by calling ExtractObject for each of the other object types in turn (plus the create database command that is supported), with no limits placed by the ObjectName and ExtractTypes parameters.

eoDomain Lists all domains. If ExtractTypes includes etTable, this list includes only domains for the table specified by ObjectName.

eoTable If ObjectName is an empty string, lists all the tables defined on the server. If ObjectName is non-empty, lists all the columns and types of the table named by ObjectName, plus any information specified by ExtractTypes that pertains to that table. This can include domains (etDomain), indexes (etIndex), foreign key constrains (etForeign), check constrains (etCheck), triggers (etTrigger), user permissions (etGrant), and data (etData).

eoView If ObjectName is an empty string, lists the text of all views. Otherwise, lists the text of the view named by the ObjectName parameter.

eoProcedure If ObjectName is an empty string, lists all the stored procedures. Otherwise, lists the text of the stored procedure named by the ObjectName parameter. To handle the situation where procedures reference each other, ExtractObject creates dummy procedures of the correct name and then alters these to their correct form. If ExtractTypes includes etAlterProc, only the corrected form is fetched.

eoFunction Lists all external functions. If ObjectName is non-empty, the list is filtered to include only those functions that match the specified name.

eoGenerator Lists all non-system generators. If ObjectName is non-empty, the list is filtered to include only those generators that match the specified name.

eoException Lists all exceptions that the database defines. If ObjectName is non-empty, the list is filtered to include only those exceptions that match the specified name.

eoBLOBFilter Lists all blob filters. If ObjectName is non-empty, the list is filtered to include only those filters whose function name matches the specified name.

eoRole Lists all roles. If ObjectName is non-empty, the list is filtered to include only those roles whose name matches the specified string.

eoTrigger If ExtractTypes includes etTable, lists all triggers defined for the table specified by ObjectName. Otherwise, lists all triggers or, if ObjectName is nonempty, all triggers that match the string specified by ObjectName.

eoForeign If ExtractTypes includes etTable, lists all foreign key constraints that pertain to the table specified by ObjectName. Otherwise, list all foreign key constraints, or, if ObjectName is nonempty, all foreign key constraints that have a name that matches ObjectName.

eoIndexes If ExtractTypes includes etTable, lists all indexes defined on the table specified by ObjectName. Otherwise, lists all indexes, or, if ObjectName is nonempty, all indexes that have a name that matches ObjectName.

eoChecks If ExtractTypes includes etTable, lists all non-foreign check constraints on the table specified by ObjectName. Otherwise, lists all non-foreign check constraints, or, if ObjectName is nonempty, all check constraints that have a name that matches ObjectName.

eoData Lists all the data in the object named by the ObjectName parameter.

ObjectName specifies the name of an object (such as a table or stored procedure) about which you want information. The type of this object and the way it limits the fetched information depends on the values of ObjectType and ExtractTypes.

ExtractTypes modifies the interpretation of ObjectType, to filter the type of information fetched.

23.1.1.4.5 TIBExtract.GetArrayField Method

Returns the dimensions of an array field as a printable string.

Pascal

```
function GetArrayField(FieldName: String): String;
```

C++

```
__fastcall UnicodeString GetArrayField(UnicodeString FieldName);
```

Description

GetArrayField returns the dimensions of the array field with the name specified by the FieldName parameter. The resulting string has the form

```
[lower:upper, lower:upper,..]
```

23.1.1.4.6 TIBExtract.GetCharacterSets Method

Returns a specified character set and collation order as a printable string.

Pascal

```
function GetCharacterSets(CharSetId: Short; Collation: Short; CollateOnly: Boolean): String;
```

C++

```
__fastcall UnicodeString GetCharacterSets(Short CharSetId, Short Collation, Boolean
CollateOnly);
```

Description

GetCharacterSets looks up a specified character set and collation order, and returns their names.

CharSetId is the identifier of the desired character set.

Collation is the identifier of the desired collation order.

CollateOnly indicates that the return string should include the name of the collation order but not the name of the character set.

23.1.1.4.7 TIBExtract.GetFieldType Method

Returns the full description of a specified field type as a printable string.

Pascal

```
function GetFieldType(FieldType: Integer; FieldSubType: Integer; FieldScale: Integer;
FieldSize: Integer; FieldPrec: Integer; FieldLen: Integer): String;
```

C++

```
__fastcall UnicodeString GetFieldType(int FieldType, int FieldSubType, int FieldScale, int
FieldSize, int FieldPrec, int FieldLen);
```

Description

GetFieldType returns the string description with the specified field type, subtype, scale, size, precision, and length.

23.1.1.4.8 TIBExtract.ListAllTables Method

Pascal

```
procedure ListAllTables(flag: Boolean);
```

C++

```
__fastcall ListAllTables(Boolean flag);
```

Description

This is ListAllTables, a member of class TIBExtract.

23.1.1.4.9 TIBExtract.ListCheck Method

Pascal

```
procedure ListCheck(ObjectName: String = ''; ExtractType: TExtractType = etCheck);
```

C++

```
__fastcall ListCheck(UnicodeString ObjectName = '', TExtractType ExtractType = etCheck);
```

Description

This is ListCheck, a member of class TIBExtract.

23.1.1.4.10 TIBExtract.ListCreateDb Method

Pascal

```
procedure ListCreateDb(TargetDb: String = '');
```

C++

```
__fastcall ListCreateDb(UnicodeString TargetDb = '');
```

Description

This is ListCreateDb, a member of class TIBExtract.

23.1.1.4.11 TIBExtract.ListData Method

Pascal

```
procedure ListData(ObjectName: String);
```

C++

```
__fastcall ListData(UnicodeString ObjectName);
```

Description

This is ListData, a member of class TIBExtract.

23.1.1.4.12 TIBExtract.ListDomains Method

Pascal

```
procedure ListDomains(ObjectName: String = ''; ExtractType: TExtractType = etDomain);
```

C++

```
__fastcall ListDomains(UnicodeString ObjectName = '', TExtractType ExtractType = etDomain);
```

Description

This is ListDomains, a member of class TIBExtract.

23.1.1.4.13 TIBExtract.ListException Method

Pascal

```
procedure ListException(ExceptionName: String = '');
```

C++

```
__fastcall ListException(UnicodeString ExceptionName = '');
```

Description

This is ListException, a member of class TIBExtract.

23.1.1.4.14 TIBExtract.ListFilters Method

Pascal

```
procedure ListFilters(FilterName: String = '');
```

C++

```
__fastcall ListFilters(UnicodeString FilterName = '');
```

Description

This is ListFilters, a member of class TIBExtract.

23.1.1.4.15 TIBExtract.ListForeign Method

Pascal

```
procedure ListForeign(ObjectName: String = ''; ExtractType: TExtractType = etForeign);
```

C++

```
__fastcall ListForeign(UnicodeString ObjectName = '', TExtractType ExtractType = etForeign);
```

Description

This is ListForeign, a member of class TIBExtract.

23.1.1.4.16 TIBExtract.ListFunctions Method

Pascal

```
procedure ListFunctions(FunctionName: String = '');
```

C++

```
__fastcall ListFunctions(UnicodeString FunctionName = '');
```

Description

This is ListFunctions, a member of class TIBExtract.

23.1.1.4.17 TIBExtract.ListGenerators Method

Pascal

```
procedure ListGenerators(GeneratorName: String = '');
```

C++

```
__fastcall ListGenerators(UnicodeString GeneratorName = '');
```

Description

This is ListGenerators, a member of class TIBExtract.

23.1.1.4.18 TIBExtract.ListGrants Method

Pascal

```
procedure ListGrants;
```

C++

```
__fastcall ListGrants();
```

Description

This is ListGrants, a member of class TIBExtract.

23.1.1.4.19 TIBExtract.ListIndex Method

Pascal

```
procedure ListIndex(ObjectName: String = ''; ExtractType: TExtractType = etIndex);
```

C++

```
__fastcall ListIndex(UnicodeString ObjectName = '', TExtractType ExtractType = etIndex);
```

Description

This is ListIndex, a member of class TIBExtract.

23.1.1.4.20 TIBExtract.ListProcs Method

Pascal

```
procedure ListProcs(ProcedureName: String = ''; AlterOnly: Boolean = false);
```

C++

```
__fastcall ListProcs(UnicodeString ProcedureName = '', Boolean AlterOnly = false);
```

Description

This is ListProcs, a member of class TIBExtract.

23.1.1.4.21 TIBExtract.ListRoles Method

Pascal

```
procedure ListRoles(ObjectName: String = '');
```

C++

```
__fastcall ListRoles(UnicodeString ObjectName = '');
```

Description

This is ListRoles, a member of class TIBExtract.

23.1.1.4.22 TIBExtract.ListTriggers Method

Pascal

```
procedure ListTriggers(ObjectName: String = ''; ExtractType: TExtractType = etTrigger);
```

C++

```
__fastcall ListTriggers(UnicodeString ObjectName = '', TExtractType ExtractType = etTrigger);
```

Description

This is ListTriggers, a member of class TIBExtract.

23.1.1.4.23 TIBExtract.ListViews Method

Pascal

```
procedure ListViews(ViewName: String = '');
```

C++

```
__fastcall ListViews(UnicodeString ViewName = '');
```

Description

This is ListViews, a member of class TIBExtract.

23.1.1.4.24 TIBExtract.Notification Method

Responds to notifications when components are created or freed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when objects are created or freed so that TIBExtract (see page 3811) can make any necessary adjustments.

AComponent is the component that was just created or that is about to be destroyed.

Operation indicates whether AComponent was just created (opInsert) or is about to be freed (opRemove).

In TIBExtract (see page 3811), Notification checks whether the components that implement the Database (see page 3821) and Transaction (see page 3822) properties are about to be freed, and if so, sets the appropriate property to nil (Delphi) or NULL (C++).

23.1.1.4.25 TIBExtract.PrintSet Method

Pascal

```
function PrintSet(var Used: Boolean): String;
```

C++

```
__fastcall UnicodeString PrintSet(Boolean Used);
```

Description

This is PrintSet, a member of class TIBExtract.

23.1.1.5 TIBExtract Properties

The properties of the TIBExtract class are listed here.

Public Properties

	Name	Description
	DatabaseInfo (see page 3821)	Represents information about the associated database.
	Items (see page 3821)	Lists the information that the TIBExtract (see page 3811) object has fetched from the database server.

Published Properties

	Name	Description
	Database (see page 3821)	Specifies the database from which to fetch metadata.
	ShowSystem (see page 3822)	Reserved for future use.
	Transaction (see page 3822)	Specifies the transaction to use when fetching metadata from the server.

23.1.1.5.1 TIBExtract.Database Property

Specifies the database from which to fetch metadata.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use Database to specify the database from which to fetch metadata.

23.1.1.5.2 TIBExtract.DatabaseInfo Property

Represents information about the associated database.

Pascal

```
property DatabaseInfo: TIBDatabaseInfo;
```

C++

```
__property TIBDatabaseInfo DatabaseInfo;
```

Description

Read DatabaseInfo to obtain information about the database that is specified by the Database (see page 3821) property. DatabaseInfo provides a variety of information about the database itself, such as the version of the online disk structure (ODS) used by the attachment, the number of database cache buffers allocated, the number of database pages read from or written to, and write-ahead log information.

23.1.1.5.3 TIBExtract.Items Property

Lists the information that the TIBExtract (see page 3811) object has fetched from the database server.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use the Items property to read the metadata information that was fetched from the database server. The ExtractObject (see page 3814) method writes information to the Items property every time you call it.

Tip: TIBExtract (see page 3811) does not clear the Items property when you make a new call to ExtractObject (see page 3814). You may want to clear this list of strings between calls to ExtractObject (see page 3814) so that the list does not get too long.

23.1.1.5.4 TIBExtract.ShowSystem Property

Reserved for future use.

Pascal

```
property ShowSystem: Boolean;
```

C++

```
__property Boolean ShowSystem;
```

Description

Currently, ShowSystem is not implemented.

23.1.1.5.5 TIBExtract.Transaction Property

Specifies the transaction to use when fetching metadata from the server.

Pascal

```
property Transaction: TIBTransaction;
```

C++

```
__property TIBTransaction Transaction;
```

Description

Set Transaction to specify a transaction to use when fetching metadata from the server. The ExtractObject (see page 3814) method executes all calls to the server within the context of this transaction.

23.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TExtractObjectTypes (see page 3823)	TExtractObjectTypes indicates the type of metadata to fetch from an InterBase server.
	TExtractType (see page 3824)	This is record IBExtract.TExtractType.

Records

	Name	Description
	TPrivTypes (see page 3825)	This is record IBExtract.TPrivTypes.
	TSQLType (see page 3825)	This is record IBExtract.TSQLType.

23.2.1 IBExtract.TExtractObjectTypes Enumeration

TExtractObjectTypes indicates the type of metadata to fetch from an InterBase server.

Pascal

```
TExtractObjectTypes = (
  eoDatabase,
  eoDomain,
  eoTable,
  eoView,
  eoProcedure,
  eoFunction,
  eoGenerator,
  eoException,
  eoBLOBFilter,
  eoRole,
  eoTrigger,
  eoForeign,
  eoIndexes,
  eoChecks,
  eoData
);
```

C++

```
enum TExtractObjectTypes {
  eoDatabase,
  eoDomain,
  eoTable,
  eoView,
  eoProcedure,
  eoFunction,
  eoGenerator,
  eoException,
  eoBLOBFilter,
  eoRole,
  eoTrigger,
  eoForeign,
  eoIndexes,
  eoChecks,
```

```
    eoData
  };
```

File

IBExtract

Description

Value Meaning

eoDatabase Fetch metadata about all objects defined on the database.

eoDomain List all domains, either for the entire database or only those that pertain to a specified table.

eoTable List all tables or information about a specified table.

eoView List all views in the database or the text of a specified view.

eoProcedure List all stored procedures or the text of a specified stored procedure.

eoFunction List all external functions, or list those external functions that match a specified name.

eoGenerator List all non-system generators, or only those generators that match a specified name.

eoException List all exceptions that the database defines, or only those exceptions that match a specified name.

eoBLOBFilter List all blob filters or only those filters whose function name matches a specified name.

eoRole List all roles or only those roles whose name matches a specified string.

eoTrigger List all triggers. The scope can be the entire database, those triggers that match a specified string, or those triggers defined for a specified table.

eoForeign List all foreign key constraints. The scope can be the entire database, those foreign key constraints with a specified name, or those that pertain to a specified table.

eoIndexes List all indexes. The scope can be the entire database, those indexes with a specified name, or those that are defined for a specified table.

eoChecks List all non-foreign check constraints. The scope can be the entire database, those check constraints with a specified name, or those that pertain to a specified table.

eoData List all the data a specified object.

23.2.2 IBExtract.TExtractType Enumeration

Pascal

```
TExtractType = (
  etDomain,
  etTable,
  etRole,
  etTrigger,
  etForeign,
  etIndex,
  etData,
  etGrant,
  etCheck,
  etAlterProc
);
```

C++

```
enum TExtractType {
  etDomain,
  etTable,
  etRole,
  etTrigger,
  etForeign,
  etIndex,
  etData,
  etGrant,
  etCheck,
  etAlterProc
};
```

File

IBExtract

Description

This is record IBExtract.TExtractType.

23.2.3 IBExtract.TPrivTypes Record

Pascal

```
TPrivTypes = record
  PrivFlag: Integer;
  PrivString: String;
end;
```

C++

```
struct TPrivTypes {
  int PrivFlag;
  UnicodeString PrivString;
};
```

File

IBExtract

Description

This is record IBExtract.TPrivTypes.

23.2.4 IBExtract.TSQLType Record

Pascal

```
TSQLType = record
  SqlType: Integer;
  TypeName: String;
end;
```

C++

```
struct TSQLType {
  int SqlType;
  UnicodeString TypeName;
```

```
};
```

File

IBExtract

Description

This is record IBExtract.TSQLType.

23.3 Types

The following table lists types in this documentation.

Types

Name	Description
TExtractTypes (see page 3827)	TExtractType (see page 3824) and TExtractTypes indicate the scope of metadata to extract from an InterBase server.
TSQLTypes (see page 3828)	This is type IBExtract.TSQLTypes.

23.3.1 IBExtract.TExtractTypes Type

TExtractType (see page 3824) and TExtractTypes indicate the scope of metadata to extract from an InterBase server.

Pascal

```
TExtractTypes = set of TExtractType;
```

C++

```
TExtractType TExtractTypes;
```

File

IBExtract

Description

TExtractTypes is a set of TExtractType (see page 3824) values. The following table lists the possible values:

Value Meaning

etDomain When fetching metadata about a database table, include information about all domains that pertain to the table.

etTable Fetch metadata of the target type that pertains only to a specified named table.

etRole When fetching metadata about a database table, include information about all roles that have access to the table.

etTrigger When fetching metadata about a database table, include information about all triggers that pertain to the table.

etForeign When fetching metadata about a database table, include information about foreign key constraints.

etIndex When fetching metadata about a database table, include information about indexes defined for the table.

etData When fetching metadata about a database table, include the data values.

etGrant When fetching metadata about a database table, include information about user permissions.

etCheck When fetching metadata about a database table, include information about check constraints.

etAlterProc When fetching metadata about stored procedures, the extraction may involve the creation of dummy procedures to handle the situation where procedures reference each other. These dummy procedures, which stand in for a referenced stored procedure, are later altered to their correct form. This flag indicates that only the corrected form should be fetched.

23.3.2 IBExtract.TSQLTypes Type

Pascal

```
TSQLTypes = Array[0..13] of TSQLType;
```

C++

```
Array[0..13] of TSQLType TSQLTypes;
```

File

IBExtract

Description

This is type IBExtract.TSQLTypes.

23.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ColumnTypes (see page 3830)	This is constant IBExtract.ColumnTypes.
FILE_cache (see page 3830)	This is constant IBExtract.FILE_cache.
FILE_conditional (see page 3830)	This is constant IBExtract.FILE_conditional.
FILE_inactive (see page 3831)	This is constant IBExtract.FILE_inactive.
FILE_manual (see page 3831)	This is constant IBExtract.FILE_manual.
FILE_shadow (see page 3831)	This is constant IBExtract.FILE_shadow.
IntegralSubtypes (see page 3832)	This is constant IBExtract.IntegralSubtypes.
LOG_default (see page 3832)	This is constant IBExtract.LOG_default.
LOG_overflow (see page 3832)	This is constant IBExtract.LOG_overflow.
LOG_raw (see page 3833)	This is constant IBExtract.LOG_raw.
LOG_serial (see page 3833)	This is constant IBExtract.LOG_serial.
MAX_INTSUBTYPES (see page 3833)	This is constant IBExtract.MAX_INTSUBTYPES.
MAXSUBTYPES (see page 3834)	This is constant IBExtract.MAXSUBTYPES.
obj_computed (see page 3834)	This is constant IBExtract.obj_computed.
obj_count (see page 3834)	This is constant IBExtract.obj_count.
obj_exception (see page 3835)	This is constant IBExtract.obj_exception.
obj_expression_index (see page 3835)	This is constant IBExtract.obj_expression_index.
obj_field (see page 3835)	This is constant IBExtract.obj_field.
obj_index (see page 3836)	This is constant IBExtract.obj_index.
obj_procedure (see page 3836)	This is constant IBExtract.obj_procedure.
obj_relation (see page 3836)	This is constant IBExtract.obj_relation.
obj_sql_role (see page 3837)	This is constant IBExtract.obj_sql_role.
obj_trigger (see page 3837)	This is constant IBExtract.obj_trigger.
obj_user (see page 3837)	This is constant IBExtract.obj_user.
obj_user_group (see page 3838)	This is constant IBExtract.obj_user_group.
obj_validation (see page 3838)	This is constant IBExtract.obj_validation.
obj_view (see page 3838)	This is constant IBExtract.obj_view.
ODS_VERSION10 (see page 3839)	This is constant IBExtract.ODS_VERSION10.
ODS_VERSION6 (see page 3839)	This is constant IBExtract.ODS_VERSION6.
ODS_VERSION7 (see page 3839)	This is constant IBExtract.ODS_VERSION7.
ODS_VERSION8 (see page 3840)	This is constant IBExtract.ODS_VERSION8.
ODS_VERSION9 (see page 3840)	This is constant IBExtract.ODS_VERSION9.
priv_DELETE (see page 3840)	This is constant IBExtract.priv_DELETE.
priv_EXECUTE (see page 3841)	This is constant IBExtract.priv_EXECUTE.
priv_INSERT (see page 3841)	This is constant IBExtract.priv_INSERT.
priv_REFERENCES (see page 3841)	This is constant IBExtract.priv_REFERENCES.
priv_SELECT (see page 3842)	This is constant IBExtract.priv_SELECT.
priv_UNKNOWN (see page 3842)	This is constant IBExtract.priv_UNKNOWN.
priv_UPDATE (see page 3842)	This is constant IBExtract.priv_UPDATE.
PrivTypes (see page 3843)	This is constant IBExtract.PrivTypes.
SubTypes (see page 3843)	This is constant IBExtract.SubTypes.
TriggerTypes (see page 3843)	This is constant IBExtract.TriggerTypes.

23.4.1 IBExtract.ColumnTypes Constant

Pascal

```
ColumnTypes: TSQLTypes = ( (SqlType : blr_short; TypeName : 'SMALLINT'), (SqlType : blr_long;
TypeName : 'INTEGER'), (SqlType : blr_quad; TypeName : 'QUAD'), (SqlType : blr_float; TypeName
: 'FLOAT'), (SqlType : blr_text; TypeName : 'CHAR'), (SqlType : blr_double; TypeName : 'DOUBLE
PRECISION'), (SqlType : blr_varying; TypeName : 'VARCHAR'), (SqlType : blr_cstring; TypeName :
'CSTRING'), (SqlType : blr_blob_id; TypeName : 'BLOB_ID'), (SqlType : blr_blob; TypeName :
'BLOB'), (SqlType : blr_sql_time; TypeName : 'TIME'), (SqlType : blr_sql_date; TypeName :
'DATE'), (SqlType : blr_timestamp; TypeName : 'TIMESTAMP'), (SqlType : blr_int64; TypeName :
'INT64'));
```

C++

```
TSQLTypes ColumnTypes = ( (SqlType : blr_short; TypeName : 'SMALLINT'), (SqlType : blr_long;
TypeName : 'INTEGER'), (SqlType : blr_quad; TypeName : 'QUAD'), (SqlType : blr_float; TypeName
: 'FLOAT'), (SqlType : blr_text; TypeName : 'CHAR'), (SqlType : blr_double; TypeName : 'DOUBLE
PRECISION'), (SqlType : blr_varying; TypeName : 'VARCHAR'), (SqlType : blr_cstring; TypeName :
'CSTRING'), (SqlType : blr_blob_id; TypeName : 'BLOB_ID'), (SqlType : blr_blob; TypeName :
'BLOB'), (SqlType : blr_sql_time; TypeName : 'TIME'), (SqlType : blr_sql_date; TypeName :
'DATE'), (SqlType : blr_timestamp; TypeName : 'TIMESTAMP'), (SqlType : blr_int64; TypeName :
'INT64'));
```

File

IBExtract

Description

This is constant IBExtract.ColumnTypes.

23.4.2 IBExtract.FILE_cache Constant

Pascal

```
FILE_cache = 8;
```

C++

```
FILE_cache = 8;
```

File

IBExtract

Description

This is constant IBExtract.FILE_cache.

23.4.3 IBExtract.FILE_conditional Constant

Pascal

```
FILE_conditional = 16;
```

C++

```
FILE_conditional = 16;
```

File

IBExtract

Description

This is constant IBExtract.FILE_conditional.

23.4.4 IBExtract.FILE_inactive Constant

Pascal

```
FILE_inactive = 2;
```

C++

```
FILE_inactive = 2;
```

File

IBExtract

Description

This is constant IBExtract.FILE_inactive.

23.4.5 IBExtract.FILE_manual Constant

Pascal

```
FILE_manual = 4;
```

C++

```
FILE_manual = 4;
```

File

IBExtract

Description

This is constant IBExtract.FILE_manual.

23.4.6 IBExtract.FILE_shadow Constant

Pascal

```
FILE_shadow = 1;
```

C++

```
FILE_shadow = 1;
```

File

IBExtract

Description

This is constant IBExtract.FILE_shadow.

23.4.7 IBExtract.IntegralSubtypes Constant

Pascal

```
IntegralSubtypes: Array[0..2] of String = ( 'UNKNOWN', 'NUMERIC', 'DECIMAL');
```

C++

```
Array[0..2] of String IntegralSubtypes = ( 'UNKNOWN', 'NUMERIC', 'DECIMAL');
```

File

IBExtract

Description

This is constant IBExtract.IntegralSubtypes.

23.4.8 IBExtract.LOG_default Constant

Pascal

```
LOG_default = 2;
```

C++

```
LOG_default = 2;
```

File

IBExtract

Description

This is constant IBExtract.LOG_default.

23.4.9 IBExtract.LOG_overflow Constant

Pascal

```
LOG_overflow = 8;
```

C++

```
LOG_overflow = 8;
```

File

IBExtract

Description

This is constant IBExtract.LOG_overflow.

23.4.10 IBExtract.LOG_raw Constant

Pascal

```
LOG_raw = 4;
```

C++

```
LOG_raw = 4;
```

File

IBExtract

Description

This is constant IBExtract.LOG_raw.

23.4.11 IBExtract.LOG_serial Constant

Pascal

```
LOG_serial = 1;
```

C++

```
LOG_serial = 1;
```

File

IBExtract

Description

This is constant IBExtract.LOG_serial.

23.4.12 IBExtract.MAX_INTSUBTYPES Constant

Pascal

```
MAX_INTSUBTYPES = 2;
```

C++

```
MAX_INTSUBTYPES = 2;
```

File

IBExtract

Description

This is constant IBExtract.MAX_INTSUBTYPES.

23.4.13 IBExtract.MAXSUBTYPES Constant

Pascal

```
MAXSUBTYPES = 8;
```

C++

```
MAXSUBTYPES = 8;
```

File

IBExtract

Description

This is constant IBExtract.MAXSUBTYPES.

23.4.14 IBExtract.obj_computed Constant

Pascal

```
obj_computed = 3;
```

C++

```
obj_computed = 3;
```

File

IBExtract

Description

This is constant IBExtract.obj_computed.

23.4.15 IBExtract.obj_count Constant

Pascal

```
obj_count = 11;
```

C++

```
obj_count = 11;
```

File

IBExtract

Description

This is constant IBExtract.obj_count.

23.4.16 IBExtract.obj_exception Constant

Pascal

```
obj_exception = 7;
```

C++

```
obj_exception = 7;
```

File

IBExtract

Description

This is constant IBExtract.obj_exception.

23.4.17 IBExtract.obj_expression_index Constant

Pascal

```
obj_expression_index = 6;
```

C++

```
obj_expression_index = 6;
```

File

IBExtract

Description

This is constant IBExtract.obj_expression_index.

23.4.18 IBExtract.obj_field Constant

Pascal

```
obj_field = 9;
```

C++

```
obj_field = 9;
```

File

IBExtract

Description

This is constant IBExtract.obj_field.

23.4.19 IBExtract.obj_index Constant

Pascal

```
obj_index = 10;
```

C++

```
obj_index = 10;
```

File

IBExtract

Description

This is constant IBExtract.obj_index.

23.4.20 IBExtract.obj_procedure Constant

Pascal

```
obj_procedure = 5;
```

C++

```
obj_procedure = 5;
```

File

IBExtract

Description

This is constant IBExtract.obj_procedure.

23.4.21 IBExtract.obj_relation Constant

Pascal

```
obj_relation = 0;
```

C++

```
obj_relation = 0;
```

File

IBExtract

Description

This is constant IBExtract.obj_relation.

23.4.22 IBExtract.obj_sql_role Constant

Pascal

```
obj_sql_role = 13;
```

C++

```
obj_sql_role = 13;
```

File

IBExtract

Description

This is constant IBExtract.obj_sql_role.

23.4.23 IBExtract.obj_trigger Constant

Pascal

```
obj_trigger = 2;
```

C++

```
obj_trigger = 2;
```

File

IBExtract

Description

This is constant IBExtract.obj_trigger.

23.4.24 IBExtract.obj_user Constant

Pascal

```
obj_user = 8;
```

C++

```
obj_user = 8;
```

File

IBExtract

Description

This is constant IBExtract.obj_user.

23.4.25 IBExtract.obj_user_group Constant

Pascal

```
obj_user_group = 12;
```

C++

```
obj_user_group = 12;
```

File

IBExtract

Description

This is constant IBExtract.obj_user_group.

23.4.26 IBExtract.obj_validation Constant

Pascal

```
obj_validation = 4;
```

C++

```
obj_validation = 4;
```

File

IBExtract

Description

This is constant IBExtract.obj_validation.

23.4.27 IBExtract.obj_view Constant

Pascal

```
obj_view = 1;
```

C++

```
obj_view = 1;
```

File

IBExtract

Description

This is constant IBExtract.obj_view.

23.4.28 IBExtract.ODS_VERSION10 Constant

Pascal

```
ODS_VERSION10 = 10;
```

C++

```
ODS_VERSION10 = 10;
```

File

IBExtract

Description

This is constant IBExtract.ODS_VERSION10.

23.4.29 IBExtract.ODS_VERSION6 Constant

Pascal

```
ODS_VERSION6 = 6;
```

C++

```
ODS_VERSION6 = 6;
```

File

IBExtract

Description

This is constant IBExtract.ODS_VERSION6.

23.4.30 IBExtract.ODS_VERSION7 Constant

Pascal

```
ODS_VERSION7 = 7;
```

C++

```
ODS_VERSION7 = 7;
```

File

IBExtract

Description

This is constant IBExtract.ODS_VERSION7.

23.4.31 IBExtract.ODS_VERSION8 Constant

Pascal

```
ODS_VERSION8 = 8;
```

C++

```
ODS_VERSION8 = 8;
```

File

IBExtract

Description

This is constant IBExtract.ODS_VERSION8.

23.4.32 IBExtract.ODS_VERSION9 Constant

Pascal

```
ODS_VERSION9 = 9;
```

C++

```
ODS_VERSION9 = 9;
```

File

IBExtract

Description

This is constant IBExtract.ODS_VERSION9.

23.4.33 IBExtract.priv_DELETE Constant

Pascal

```
priv_DELETE = 16;
```

C++

```
priv_DELETE = 16;
```

File

IBExtract

Description

This is constant IBExtract.priv_DELETE.

23.4.34 IBExtract.priv_EXECUTE Constant

Pascal

```
priv_EXECUTE = 32;
```

C++

```
priv_EXECUTE = 32;
```

File

IBExtract

Description

This is constant IBExtract.priv_EXECUTE.

23.4.35 IBExtract.priv_INSERT Constant

Pascal

```
priv_INSERT = 4;
```

C++

```
priv_INSERT = 4;
```

File

IBExtract

Description

This is constant IBExtract.priv_INSERT.

23.4.36 IBExtract.priv_REFERENCES Constant

Pascal

```
priv_REFERENCES = 64;
```

C++

```
priv_REFERENCES = 64;
```

File

IBExtract

Description

This is constant IBExtract.priv_REFERENCES.

23.4.37 IBExtract.priv_SELECT Constant

Pascal

```
priv_SELECT = 2;
```

C++

```
priv_SELECT = 2;
```

File

IBExtract

Description

This is constant IBExtract.priv_SELECT.

23.4.38 IBExtract.priv_UNKNOWN Constant

Pascal

```
priv_UNKNOWN = 1;
```

C++

```
priv_UNKNOWN = 1;
```

File

IBExtract

Description

This is constant IBExtract.priv_UNKNOWN.

23.4.39 IBExtract.priv_UPDATE Constant

Pascal

```
priv_UPDATE = 8;
```

C++

```
priv_UPDATE = 8;
```

File

IBExtract

Description

This is constant IBExtract.priv_UPDATE.

23.4.40 IBExtract.PrivTypes Constant

Pascal

```
PrivTypes: Array[0..5] of TPrivTypes = ( (PrivFlag : priv_DELETE; PrivString : 'DELETE' ),
(PrivFlag : priv_EXECUTE; PrivString : 'EXECUTE' ), (PrivFlag : priv_INSERT; PrivString :
'INSERT' ), (PrivFlag : priv_SELECT; PrivString : 'SELECT' ), (PrivFlag : priv_UPDATE;
PrivString : 'UPDATE' ), (PrivFlag : priv_REFERENCES; PrivString : 'REFERENCES'));
```

C++

```
Array[0..5] of TPrivTypes PrivTypes = ( (PrivFlag : priv_DELETE; PrivString : 'DELETE' ),
(PrivFlag : priv_EXECUTE; PrivString : 'EXECUTE' ), (PrivFlag : priv_INSERT; PrivString :
'INSERT' ), (PrivFlag : priv_SELECT; PrivString : 'SELECT' ), (PrivFlag : priv_UPDATE;
PrivString : 'UPDATE' ), (PrivFlag : priv_REFERENCES; PrivString : 'REFERENCES'));
```

File

IBExtract

Description

This is constant IBExtract.PrivTypes.

23.4.41 IBExtract.SubTypes Constant

Pascal

```
SubTypes: Array[0..8] of String = ( 'UNKNOWN', 'TEXT', 'BLR', 'ACL', 'RANGES', 'SUMMARY',
'FORMAT', 'TRANSACTION_DESCRIPTION', 'EXTERNAL_FILE_DESCRIPTION');
```

C++

```
Array[0..8] of String SubTypes = ( 'UNKNOWN', 'TEXT', 'BLR', 'ACL', 'RANGES', 'SUMMARY',
'FORMAT', 'TRANSACTION_DESCRIPTION', 'EXTERNAL_FILE_DESCRIPTION');
```

File

IBExtract

Description

This is constant IBExtract.SubTypes.

23.4.42 IBExtract.TriggerTypes Constant

Pascal

```
TriggerTypes: Array[0..6] of String = ( '', 'BEFORE INSERT', 'AFTER INSERT', 'BEFORE UPDATE',
'AFTER UPDATE', 'BEFORE DELETE', 'AFTER DELETE');
```

C++

```
Array[0..6] of String TriggerTypes = ( '', 'BEFORE INSERT', 'AFTER INSERT', 'BEFORE UPDATE',
'AFTER UPDATE', 'BEFORE DELETE', 'AFTER DELETE');
```

File

IBExtract

Description

This is constant IBExtract.TriggerTypes.

24 IBInstall Namespace

This is namespace IBInstall.

Topics

Name	Description
Classes (see page 3847)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3868)	The following table lists structs, records, enums in this documentation.
Types (see page 3873)	The following table lists types in this documentation.

Classes

	Name	Description
	EIBInstall (see page 3847)	EIBInstall catches all of the installation exceptions.
	EIBInstallerError (see page 3849)	This is class IBInstall.EIBInstallerError.
	EIBInstallError (see page 3850)	This is class IBInstall.EIBInstallError.
	TIBInstall (see page 3850)	Use TIBInstall to set up an InterBase installation component.
	TIBSetup (see page 3857)	TIBSetup is the ancestor object from which TIBInstall (see page 3850) and TIBUnInstall (see page 3862) are derived.
	TIBUnInstall (see page 3862)	Use TIBUnInstall to set up an uninstall component.
	TInstallOptions (see page 3865)	TInstallOptions configures the options of a TIBInstall (see page 3850) instance.

Structs, Records, Enums

	Name	Description
	TCmdOption (see page 3868)	TCmdOption and TCmdOptions (see page 3873) indicate which of the command-line tools are installed.
	TConnectivityOption (see page 3869)	TConnectivityOption and TConnectivityOptions (see page 3873) identify InterBase connectivity clients.
	TErrorResult (see page 3869)	This is record IBInstall.TErrorResult.
	TExamplesOption (see page 3870)	TExamplesOption and TExamplesOptions (see page 3874) identify InterBase Examples clients.
	TIBInstallerError (see page 3870)	This is record IBInstall.TIBInstallerError.
	TMainOption (see page 3871)	TMainOption and TMainOptions (see page 3876) identify the main InterBase components.
	TStatusResult (see page 3872)	This is record IBInstall.TStatusResult.
	TWarningResult (see page 3872)	This is record IBInstall.TWarningResult.

Types

Name	Description
TCmdOptions (see page 3873)	This is type IBInstall.TCmdOptions.
TConnectivityOptions (see page 3873)	This is type IBInstall.TConnectivityOptions.
TExamplesOptions (see page 3874)	This is type IBInstall.TExamplesOptions.
TIBSetupOnError (see page 3874)	TIBSetupOnError is the type of event handlers that respond to installation errors.
TIBSetupOnStatus (see page 3874)	TIBSetupOnStatus is the type of event handlers that respond during the InterBase installation or de-installation process.
TIBSetupOnWarning (see page 3875)	TIBSetupOnWarning is the type of event handlers that respond to installation warnings.
TIscError (see page 3875)	This is type IBInstall.TIscError.
TMainOptions (see page 3876)	This is type IBInstall.TMainOptions.

24.1 Classes

The following table lists classes in this documentation.

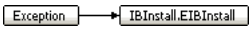
Classes

	Name	Description
	EIBInstall (see page 3847)	EIBInstall catches all of the installation exceptions.
	EIBInstallerError (see page 3849)	This is class IBInstall.EIBInstallerError.
	EIBInstallError (see page 3850)	This is class IBInstall.EIBInstallError.
	TIBInstall (see page 3850)	Use TIBInstall to set up an InterBase installation component.
	TIBSetup (see page 3857)	TIBSetup is the ancestor object from which TIBInstall (see page 3850) and TIBUnInstall (see page 3862) are derived.
	TIBUnInstall (see page 3862)	Use TIBUnInstall to set up an uninstall component.
	TInstallOptions (see page 3865)	TInstallOptions configures the options of a TIBInstall (see page 3850) instance.

24.1.1 EIBInstall Class

EIBInstall catches all of the installation exceptions.

Class Hierarchy



Pascal

```
EIBInstall = class(Exception);
```

C++

```
class EIBInstall : public Exception;
```

File

IBInstall

Description

Use EIBInstall to catch installation exceptions from InterBase and InterBase Express (IBX).

Note: You must install InterBase 6 to use this feature.

24.1.1.1 EIBInstall Members

The following tables list the members exposed by EIBInstall.

Public Methods

	Name	Description
	Create (see page 3848)	This is the overview for the Create constructor overload.

Public Properties

	Name	Description
	InstallerError (see page 3848)	Returns IBX error messages.

	InstallError (see page 3849)	Returns InterBase installation error messages.

24.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	EIBInstall.Create (MSG_NO, string) (see page 3848)	Creates an instance of an EIBInstall (see page 3847) object.
	EIBInstall.Create (TIBInstallerError, string) (see page 3848)	Creates an instance of an EIBInstall (see page 3847) object.

24.1.1.2.1 EIBInstall.Create Constructor (MSG_NO, string)

Creates an instance of an EIBInstall (see page 3847) object.

Pascal

```
constructor Create(IscCode: MSG_NO; IscMessage: string); virtual; overload;
constructor Create(ECode: TIBInstallerError; EMessage: string); virtual; overload;
```

C++

```
virtual __fastcall EIBInstall(MSG_NO IscCode, UnicodeString IscMessage);
virtual __fastcall EIBInstall(TIBInstallerError ECode, UnicodeString EMessage);
```

Description

Call Create to create an instance of an EIBInstall (see page 3847) exception object.

Note: You must install InterBase 6 to use this feature.

24.1.1.3 EIBInstall Properties

The properties of the EIBInstall class are listed here.

Public Properties

	Name	Description
	InstallerError (see page 3848)	Returns IBX error messages.
	InstallError (see page 3849)	Returns InterBase installation error messages.

24.1.1.3.1 EIBInstall.InstallerError Property

Returns IBX error messages.

Pascal

```
property InstallerError: TIBInstallerError;
```

C++

```
__property TIBInstallerError InstallerError;
```

Description

Use InstallerError to return error messages generated by the IBX install components. InstallerError can be one of:

ieSuccess	Installation is successful
ieDelphiException	Installation fails due to a Delphi exception
ieNoOptionsSet	Installation fails because no installation options are set
ieNoDestinationDirectory	Installation fails because no destination directory is set
ieNoSourceDirectory	Installation fails because no source directory is set
ieNoUninstallFile	Installation fails because no uninstall file is named
ieOptionNeedsClient	Installation fails because the installation needs the InterBase Client component to succeed
ieOptionNeedsServer	Installation fails because the installation needs the InterBase Server component to succeed
ieInvalidOption	Option is invalid
ieInvalidOnErrorResult	The OnError event result is invalid
ieInvalidOnStatusResult	The OnStatus event result is invalid

Note: You must install InterBase 6 to use this feature.

24.1.1.3.2 EIBInstall.InstallError Property

Returns InterBase installation error messages.

Pascal

```
property InstallError: MSG_NO;
```

C++

```
__property MSG_NO InstallError;
```

Description

Use InstallError to return error messages generated by the InterBase API.

Note: You must install InterBase 6 to use this feature.

24.1.2 EIBInstallerError Class

Class Hierarchy

Pascal

```
EIBInstallerError = class(EIBInstall);
```

C++

```
class EIBInstallerError : public EIBInstall;
```

File

IBInstall

Description

This is class IBInstall.EIBInstallerError.

24.1.3 EIBInstallError Class

Class Hierarchy

Pascal

```
EIBInstallError = class(EIBInstall);
```

C++

```
class EIBInstallError : public EIBInstall;
```

File

IBInstall

Description

This is class IBInstall.EIBInstallError.

24.1.4 TIBInstall Class

Use TIBInstall to set up an InterBase installation component.

Class Hierarchy

Pascal

```
TIBInstall = class(TIBSetup);
```

C++

```
class TIBInstall : public TIBSetup;
```

File

IBInstall

Description

Use TIBInstall to set up an InterBase installation component.

Note: To use this component, you must install InterBase 6 and you must place ibinstall.dll in the path.

See Also

Getting Started with InterBase Express

24.1.4.1 TIBInstall Members

The following tables list the members exposed by TIBInstall.

Protected Methods

	Name	Description
	Call (see page 3858)	This is Call, a member of class TIBSetup.
	ErrorInternal (see page 3858)	This is ErrorInternal, a member of class TIBSetup.
	GetInstallMessage (see page 3859)	This is GetInstallMessage, a member of class TIBSetup.
	IBInstallError (see page 3859)	This is IBInstallError, a member of class TIBSetup.
	StatusInternal (see page 3859)	This is StatusInternal, a member of class TIBSetup.

Public Methods

	Name	Description
V	Create (see page 3851)	Creates an installation component.
V	Destroy (see page 3852)	Destroys the instance of TIBInstall (see page 3850).
	GetOptionDescription (see page 3852)	This is the overview for the GetOptionDescription method overload.
	GetOptionName (see page 3853)	This is the overview for the GetOptionName method overload.
	GetOptionSpaceRequired (see page 3853)	This is the overview for the GetOptionSpaceRequired method overload.
	InstallCheck (see page 3854)	Checks if the installation is ready to be performed.
	InstallExecute (see page 3854)	Executes the installation.

Public Properties

	Name	Description
	ErrorContext (see page 3859)	Sets or returns the error context.
	MsgFilePath (see page 3860)	Sets or returns the installation message file path.
R	Progress (see page 3860)	Returns the installation progress.
R	RebootToComplete (see page 3860)	Sets whether it is necessary to reboot to complete the installation.
	StatusContext (see page 3861)	Sets or returns the status context.
R	SuggestedDestination (see page 3856)	Returns the suggested installation target destination.
R	UnInstallFile (see page 3856)	Returns the name and path of the uninstall file.

Published Events

	Name	Description
	OnError (see page 3861)	Occurs when an installation error occurs.
	OnStatusChange (see page 3862)	Occurs when a change in installation status occurs.
	OnWarning (see page 3862)	Occurs when an installation warning occurs.

Published Properties

	Name	Description
	DestinationDirectory (see page 3855)	Sets or returns the installation target path.
	InstallOptions (see page 3855)	Sets or displays the installation options.
	SourceDirectory (see page 3856)	Sets or returns the path of the InterBase installation source files.

24.1.4.2 TIBInstall.Create Constructor

Creates an installation component.

Pascal

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

C++

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

Description

Call (see page 3858) Create to create an installation component.

Note: You must install InterBase 6 to use this feature.

24.1.4.3 TIBInstall.Destroy Destructor

Destroys the instance of TIBInstall (see page 3850).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBInstall();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the install object has not already been freed before it calls Destroy.

Note: You must install InterBase 6 to use this feature.

24.1.4.4 TIBInstall Methods

The methods of the TIBInstall class are listed here.

Public Methods

	Name	Description
	GetOptionDescription (see page 3852)	This is the overview for the GetOptionDescription method overload.
	GetOptionName (see page 3853)	This is the overview for the GetOptionName method overload.
	GetOptionSpaceRequired (see page 3853)	This is the overview for the GetOptionSpaceRequired method overload.
	InstallCheck (see page 3854)	Checks if the installation is ready to be performed.
	InstallExecute (see page 3854)	Executes the installation.

24.1.4.4.1 GetOptionDescription Method

This is the overview for the GetOptionDescription method overload.

Overload List

	Name	Description
	TIBInstall.GetOptionDescription (TCmdOption) (see page 3852)	Returns a description of a specified option.
	TIBInstall.GetOptionDescription (TConnectivityOption) (see page 3852)	Returns a description of a specified option.
	TIBInstall.GetOptionDescription (TExamplesOption) (see page 3852)	Returns a description of a specified option.
	TIBInstall.GetOptionDescription (TMainOption) (see page 3852)	Returns a description of a specified option.

24.1.4.4.1.1 TIBInstall.GetOptionDescription Method (TCmdOption)

Returns a description of a specified option.

Pascal

```
function GetOptionDescription(Option: TExamplesOption): string; overload;
function GetOptionDescription(Option: TMainOption): string; overload;
```

```
function GetOptionDescription(Option: TConnectivityOption): string; overload;
function GetOptionDescription(Option: TCmdOption): string; overload;
```

C++

```
__fastcall UnicodeString GetOptionDescription(TExamplesOption Option);
__fastcall UnicodeString GetOptionDescription(TMainOption Option);
__fastcall UnicodeString GetOptionDescription(TConnectivityOption Option);
__fastcall UnicodeString GetOptionDescription(TCmdOption Option);
```

Description

Call (see page 3858) GetOptionDescription to return a description of the indicated option. If the specified option is not valid, an error is returned.

Note: You must install InterBase 6 to use this feature.

24.1.4.4.2 GetOptionName Method

This is the overview for the GetOptionName method overload.

Overload List

	Name	Description
	TIBInstall.GetOptionName (TCmdOption) (see page 3853)	Returns the option name.
	TIBInstall.GetOptionName (TConnectivityOption) (see page 3853)	Returns the option name.
	TIBInstall.GetOptionName (TExamplesOption) (see page 3853)	Returns the option name.
	TIBInstall.GetOptionName (TMainOption) (see page 3853)	Returns the option name.

24.1.4.4.2.1 TIBInstall.GetOptionName Method (TCmdOption)

Returns the option name.

Pascal

```
function GetOptionName(Option: TExamplesOption): string; overload;
function GetOptionName(Option: TMainOption): string; overload;
function GetOptionName(Option: TConnectivityOption): string; overload;
function GetOptionName(Option: TCmdOption): string; overload;
```

C++

```
__fastcall UnicodeString GetOptionName(TExamplesOption Option);
__fastcall UnicodeString GetOptionName(TMainOption Option);
__fastcall UnicodeString GetOptionName(TConnectivityOption Option);
__fastcall UnicodeString GetOptionName(TCmdOption Option);
```

Description

Call (see page 3858) GetOptionName to return the name of the indicated option.

Note: You must install InterBase 6 to use this feature.

24.1.4.4.3 GetOptionSpaceRequired Method

This is the overview for the GetOptionSpaceRequired method overload.

Overload List

	Name	Description
	TIBInstall.GetOptionSpaceRequired (TCmdOption) (see page 3854)	Returns the space required to install the indicated option.
	TIBInstall.GetOptionSpaceRequired (TConnectivityOption) (see page 3854)	Returns the space required to install the indicated option.
	TIBInstall.GetOptionSpaceRequired (TExamplesOption) (see page 3854)	Returns the space required to install the indicated option.
	TIBInstall.GetOptionSpaceRequired (TMainOption) (see page 3854)	Returns the space required to install the indicated option.

24.1.4.4.3.1 TIBInstall.GetOptionSpaceRequired Method (TCmdOption)

Returns the space required to install the indicated option.

Pascal

```
function GetOptionSpaceRequired(Option: TExamplesOption): longword; overload;
function GetOptionSpaceRequired(Option: TMainOption): longword; overload;
function GetOptionSpaceRequired(Option: TConnectivityOption): longword; overload;
function GetOptionSpaceRequired(Option: TCmdOption): longword; overload;
```

C++

```
__fastcall longword GetOptionSpaceRequired(TExamplesOption Option);
__fastcall longword GetOptionSpaceRequired(TMainOption Option);
__fastcall longword GetOptionSpaceRequired(TConnectivityOption Option);
__fastcall longword GetOptionSpaceRequired(TCmdOption Option);
```

Description

Call (see page 3858) GetOptionSpaceRequired to return the space required in kilobytes for the indicated option.

Note: You must install InterBase 6 to use this feature.

24.1.4.4.4 TIBInstall.InstallCheck Method

Checks if the installation is ready to be performed.

Pascal

```
procedure InstallCheck;
```

C++

```
__fastcall InstallCheck();
```

Description

Call (see page 3858) InstallCheck to check if everything is ready for the installation to be performed.

Note: You must install InterBase 6 to use this feature.

24.1.4.4.5 TIBInstall.InstallExecute Method

Executes the installation.

Pascal

```
procedure InstallExecute;
```

C++

```
__fastcall InstallExecute();
```

Description

Call (see page 3858) InstallExecute to execute the installation.

Note: You must install InterBase 6 to use this feature.

24.1.4.5 TIBInstall Properties

The properties of the TIBInstall class are listed here.

Public Properties

	Name	Description
	SuggestedDestination (see page 3856)	Returns the suggested installation target destination.
	UnInstallFile (see page 3856)	Returns the name and path of the uninstall file.

Published Properties

	Name	Description
	DestinationDirectory (see page 3855)	Sets or returns the installation target path.
	InstallOptions (see page 3855)	Sets or displays the installation options.
	SourceDirectory (see page 3856)	Sets or returns the path of the InterBase installation source files.

24.1.4.5.1 TIBInstall.DestinationDirectory Property

Sets or returns the installation target path.

Pascal

```
property DestinationDirectory: string;
```

C++

```
__property UnicodeString DestinationDirectory;
```

Description

Use DestinationDirectory to set or returns the installation target path. Initially, this will be the same as the SuggestedDestination (see page 3856).

Note: You must install InterBase 6 to use this feature.

24.1.4.5.2 TIBInstall.InstallOptions Property

Sets or displays the installation options.

Pascal

```
property InstallOptions: TInstallOptions;
```

C++

```
__property TInstallOptions InstallOptions;
```

Description

Use InstallOptions to set or display the installation options (TInstallOptions (see page 3865)). TInstallOptions (see page 3865) are:

Option	Installs:
CmdLineTools	The InterBase command line tools
ConnectivityClients	The InterBase connectivity clients
Examples	The InterBase examples
MainComponents	The main InterBase components

Note: You must install InterBase 6 to use this feature.

24.1.4.5.3 TIBInstall.SourceDirectory Property

Sets or returns the path of the InterBase installation source files.

Pascal

```
property SourceDirectory: string;
```

C++

```
__property UnicodeString SourceDirectory;
```

Description

Use SourceDirectory to set or return the path of the installation source files. In most cases, this will be a path on the InterBase CDROM.

Note: You must install InterBase 6 to use this feature.

24.1.4.5.4 TIBInstall.SuggestedDestination Property

Returns the suggested installation target destination.

Pascal

```
property SuggestedDestination: string;
```

C++

```
__property UnicodeString SuggestedDestination;
```

Description

Use SuggestedDestination to return suggested installation target destination.

Note: You must install InterBase 6 to use this feature.

24.1.4.5.5 TIBInstall.UnInstallFile Property

Returns the name and path of the uninstall file.

Pascal

```
property UnInstallFile: string;
```

C++

```
__property UnicodeString UnInstallFile;
```

Description

Use UnInstallFile to return the binary log file name and path of the uninstall file. The uninstall file contains information on which options were installed. This information lets the TIBUnInstall (see page 3862) component know what needs to be removed.

Note: You must install InterBase 6 to use this feature.

24.1.5 TIBSetup Class

TIBSetup is the ancestor object from which TIBInstall (see page 3850) and TIBUnInstall (see page 3862) are derived.

Class Hierarchy

Pascal

```
TIBSetup = class(TComponent);
```

C++

```
class TIBSetup : public TComponent;
```

File

IBInstall

Description

TIBSetup is the ancestor object from which TIBInstall (see page 3850) and TIBUnInstall (see page 3862) are derived.

Note: You must install InterBase 6 to use this feature.

24.1.5.1 TIBSetup Members

The following tables list the members exposed by TIBSetup.

Protected Methods

	Name	Description
	Call (see page 3858)	This is Call, a member of class TIBSetup.
	ErrorInternal (see page 3858)	This is ErrorInternal, a member of class TIBSetup.
	GetInstallMessage (see page 3859)	This is GetInstallMessage, a member of class TIBSetup.
	IBInstallError (see page 3859)	This is IBInstallError, a member of class TIBSetup.
	StatusInternal (see page 3859)	This is StatusInternal, a member of class TIBSetup.

Public Methods

	Name	Description
V	Create (see page 3858)	Creates an installation component.

Public Properties

	Name	Description
	ErrorContext (see page 3859)	Sets or returns the error context.
	MsgFilePath (see page 3860)	Sets or returns the installation message file path.
R	Progress (see page 3860)	Returns the installation progress.
R	RebootToComplete (see page 3860)	Sets whether it is necessary to reboot to complete the installation.
	StatusContext (see page 3861)	Sets or returns the status context.

Published Events

	Name	Description
	OnError (see page 3861)	Occurs when an installation error occurs.
	OnStatusChange (see page 3862)	Occurs when a change in installation status occurs.

	OnWarning (see page 3862)	Occurs when an installation warning occurs.

24.1.5.2 TIBSetup.Create Constructor

Creates an installation component.

Pascal

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

C++

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

Description

Call (see page 3858) Create to create an installation component.

Note: You must install InterBase 6 to use this feature.

24.1.5.3 TIBSetup Methods

The methods of the TIBSetup class are listed here.

Protected Methods

	Name	Description
	Call (see page 3858)	This is Call, a member of class TIBSetup.
	ErrorInternal (see page 3858)	This is ErrorInternal, a member of class TIBSetup.
	GetInstallMessage (see page 3859)	This is GetInstallMessage, a member of class TIBSetup.
	IBInstallError (see page 3859)	This is IBInstallError, a member of class TIBSetup.
	StatusInternal (see page 3859)	This is StatusInternal, a member of class TIBSetup.

24.1.5.3.1 TIBSetup.Call Method

Pascal

```
procedure Call(IscCode: MSG_NO);
```

C++

```
__fastcall Call(MSG_NO IscCode);
```

Description

This is Call, a member of class TIBSetup.

24.1.5.3.2 TIBSetup.ErrorInternal Method

Pascal

```
function ErrorInternal(IscCode: MSG_NO; const ActionDescription: TEXT): Integer;
```

C++

```
__fastcall int ErrorInternal(MSG_NO IscCode, const TEXT ActionDescription);
```

Description

This is ErrorInternal, a member of class TIBSetup.

24.1.5.3.3 TIBSetup.GetInstallMessage Method

Pascal

```
function GetInstallMessage(IscCode: MSG_NO): string;
```

C++

```
__fastcall UnicodeString GetInstallMessage(MSG_NO IscCode);
```

Description

This is GetInstallMessage, a member of class TIBSetup.

24.1.5.3.4 TIBSetup.IBInstallError Method

Pascal

```
procedure IBInstallError(IscCode: MSG_NO);
```

C++

```
__fastcall IBInstallError(MSG_NO IscCode);
```

Description

This is IBInstallError, a member of class TIBSetup.

24.1.5.3.5 TIBSetup.StatusInternal Method

Pascal

```
function StatusInternal(Status: Integer; const ActionDescription: TEXT): Integer;
```

C++

```
__fastcall int StatusInternal(int Status, const TEXT ActionDescription);
```

Description

This is StatusInternal, a member of class TIBSetup.

24.1.5.4 TIBSetup Properties

The properties of the TIBSetup class are listed here.

Public Properties

	Name	Description
	ErrorContext (see page 3859)	Sets or returns the error context.
	MsgFilePath (see page 3860)	Sets or returns the installation message file path.
R	Progress (see page 3860)	Returns the installation progress.
R	RebootToComplete (see page 3860)	Sets whether it is necessary to reboot to complete the installation.
	StatusContext (see page 3861)	Sets or returns the status context.

24.1.5.4.1 TIBSetup.ErrorContext Property

Sets or returns the error context.

Pascal

```
property ErrorContext: Pointer;
```

C++

```
__property void * ErrorContext;
```

Description

Use ErrorContext to set or return the user-defined data for the error function.

Note: You must install InterBase 6 to use this feature.

24.1.5.4.2 TIBSetup.MsgFilePath Property

Sets or returns the installation message file path.

Pascal

```
property MsgFilePath: string;
```

C++

```
__property UnicodeString MsgFilePath;
```

Description

Use MsgFilePath to set or return the full directory path to the filename.msg file. Most of the time this is the same as the source and path where the InterBase product is installed.

This property allows you to supply your own installation messages. It is also possible to write the setup program in such a way that you could switch languages on demand, by providing several filename.msg files at different locations.

The filename.msg file normally contains:

- Error messages
- Option names and descriptions
- Actions descriptions and status messages that are logged to the ibinstall.log file

Note: You must install InterBase 6 to use this feature.

24.1.5.4.3 TIBSetup.Progress Property

Returns the installation progress.

Pascal

```
property Progress: Integer;
```

C++

```
__property int Progress;
```

Description

Use Progress to return the installation progress as an integer between 0 and 100. Progress is updated by the OnStatusChange (see page 3862) event.

Note: You must install InterBase 6 to use this feature.

24.1.5.4.4 TIBSetup.RebootToComplete Property

Sets whether it is necessary to reboot to complete the installation.

Pascal

```
property RebootToComplete: Boolean;
```

C++

```
__property Boolean RebootToComplete;
```

Description

Use RebootToComplete to set whether it is necessary to reboot the system to complete the installation or uninstallation. By default, this is not necessary (False).

Note: You must install InterBase 6 to use this feature.

24.1.5.4.5 TIBSetup.StatusContext Property

Sets or returns the status context.

Pascal

```
property StatusContext: Pointer;
```

C++

```
__property void * StatusContext;
```

Description

Use StatusContext to set or return the user-defined data for the status function.

Note: You must install InterBase 6 to use this feature.

24.1.5.5 TIBSetup Events

The events of the TIBSetup class are listed here.

Published Events

	Name	Description
	OnError (see page 3861)	Occurs when an installation error occurs.
	OnStatusChange (see page 3862)	Occurs when a change in installation status occurs.
	OnWarning (see page 3862)	Occurs when an installation warning occurs.

24.1.5.5.1 TIBSetup.OnError Event

Occurs when an installation error occurs.

Pascal

```
property OnError: TIBSetupOnError;
```

C++

```
__property TIBSetupOnError OnError;
```

Description

Write an OnError event handler to take specific actions when an error occurs during installation.

Note: You must install InterBase 6 to use this feature.

24.1.5.5.2 TIBSetup.OnStatusChange Event

Occurs when a change in installation status occurs.

Pascal

```
property OnStatusChange: TIBSetupOnStatus;
```

C++

```
__property TIBSetupOnStatus OnStatusChange;
```

Description

Write an OnStatusChange event handler to take specific actions when a status change occurs during installation.

Status events are generated from time to time when InstallExecute is called. The event informs the user of the installation progress. The event handler might want to use the StatusContext (see page 3861) property to display information about the current installation status. The progress is returned by the Progress (see page 3860) property as an integer from 1-100.

Note: You must install InterBase 6 to use this feature.

24.1.5.5.3 TIBSetup.OnWarning Event

Occurs when an installation warning occurs.

Pascal

```
property OnWarning: TIBSetupOnWarning;
```

C++

```
__property TIBSetupOnWarning OnWarning;
```

Description

Write an OnWarning event handler to take specific actions when a warning occurs.

An OnWarning event is generated whenever warning is reported, and must return either isc_fp_install_continue or isc_install_abort.

Note: You must install InterBase 6 to use this feature.

24.1.6 TIBUnInstall Class

Use TIBUnInstall to set up an uninstall component.

Class Hierarchy

Pascal

```
TIBUnInstall = class(TIBSetup);
```

C++

```
class TIBUnInstall : public TIBSetup;
```

File

IBInstall

Description

Use TIBUnInstall to set up an uninstall component.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

24.1.6.1 TIBUnInstall Members

The following tables list the members exposed by TIBUnInstall.

Protected Methods

	Name	Description
	Call (see page 3858)	This is Call, a member of class TIBSetup.
	ErrorInternal (see page 3858)	This is ErrorInternal, a member of class TIBSetup.
	GetInstallMessage (see page 3859)	This is GetInstallMessage, a member of class TIBSetup.
	IBInstallError (see page 3859)	This is IBInstallError, a member of class TIBSetup.
	StatusInternal (see page 3859)	This is StatusInternal, a member of class TIBSetup.

Public Methods

	Name	Description
V	Create (see page 3858)	Creates an installation component.
	UnInstallCheck (see page 3864)	Checks to see if uninstallation is possible.
	UnInstallExecute (see page 3864)	Executes the uninstallation.

Public Properties

	Name	Description
	ErrorContext (see page 3859)	Sets or returns the error context.
	MsgFilePath (see page 3860)	Sets or returns the installation message file path.
R	Progress (see page 3860)	Returns the installation progress.
R	RebootToComplete (see page 3860)	Sets whether it is necessary to reboot to complete the installation.
	StatusContext (see page 3861)	Sets or returns the status context.
	UnInstallFile (see page 3864)	Returns the name and path of the InterBase uninstall file.

Published Events

	Name	Description
	OnError (see page 3861)	Occurs when an installation error occurs.
	OnStatusChange (see page 3862)	Occurs when a change in installation status occurs.
	OnWarning (see page 3862)	Occurs when an installation warning occurs.

24.1.6.2 TIBUnInstall Methods

The methods of the TIBUnInstall class are listed here.

Public Methods

	Name	Description
	UnInstallCheck (see page 3864)	Checks to see if uninstallation is possible.
	UnInstallExecute (see page 3864)	Executes the uninstallation.

24.1.6.2.1 TIBUnInstall.UnInstallCheck Method

Checks to see if uninstallation is possible.

Pascal

```
procedure UnInstallCheck;
```

C++

```
__fastcall UnInstallCheck();
```

Description

Call (see page 3858) UnInstallCheck to check if uninstallation is possible.

Note: You must install InterBase 6 to use this feature.

24.1.6.2.2 TIBUnInstall.UnInstallExecute Method

Executes the uninstallation.

Pascal

```
procedure UnInstallExecute;
```

C++

```
__fastcall UnInstallExecute();
```

Description

Call (see page 3858) UnInstallExecute to execute the uninstallation.

Note: You must install InterBase 6 to use this feature.

24.1.6.3 TIBUnInstall Properties

The properties of the TIBUnInstall class are listed here.

Public Properties

	Name	Description
	UnInstallFile (see page 3864)	Returns the name and path of the InterBase uninstall file.

24.1.6.3.1 TIBUnInstall.UnInstallFile Property

Returns the name and path of the InterBase uninstall file.

Pascal

```
property UnInstallFile: string;
```

C++

```
__property UnicodeString UnInstallFile;
```

Description

Use UnInstallFile to return the name and path of the InterBase uninstall file.

Note: You must install InterBase 6 to use this feature.

24.1.7 TInstallOptions Class

TInstallOptions configures the options of a TIBInstall (see page 3850) instance.

Class Hierarchy

Pascal

```
TInstallOptions = class(TPersistent);
```

C++

```
class TInstallOptions : public TPersistent;
```

File

IBInstall

Description

Use TInstallOptions to indicate the options to use when installing InterBase. This is the type of the Options property on TIBInstall (see page 3850).

Note: You must install InterBase 6 to use this feature.

24.1.7.1 TInstallOptions Members

The following tables list the members exposed by TInstallOptions.

Published Properties

	Name	Description
	CmdLineTools (see page 3865)	Installs the command line tools for InterBase.
	ConnectivityClients (see page 3866)	Installs the connectivity clients for InterBase.
	Examples (see page 3866)	Installs all the InterBase examples.
	MainComponents (see page 3866)	Installs the InterBase main components.

24.1.7.2 TInstallOptions Properties

The properties of the TInstallOptions class are listed here.

Published Properties

	Name	Description
	CmdLineTools (see page 3865)	Installs the command line tools for InterBase.
	ConnectivityClients (see page 3866)	Installs the connectivity clients for InterBase.
	Examples (see page 3866)	Installs all the InterBase examples.
	MainComponents (see page 3866)	Installs the InterBase main components.

24.1.7.2.1 TInstallOptions.CmdLineTools Property

Installs the command line tools for InterBase.

Pascal

```
property CmdLineTools: TCmdOptions;
```

C++

```
__property TCmdOptions CmdLineTools;
```

Description

Use CmdLineTools to indicate which of the command line tools for InterBase should be installed. The installer issues a warning for any of these if the MainComponents (see page 3866) property does not include moClient.

Note: You must install InterBase 6 to use this feature.

24.1.7.2.2 TInstallOptions.ConnectivityClients Property

Installs the connectivity clients for InterBase.

Pascal

```
property ConnectivityClients: TConnectivityOptions;
```

C++

```
__property TConnectivityOptions ConnectivityClients;
```

Description

Use ConnectivityClients to install the InterBase connectivity clients.

Note: You must install InterBase 6 to use this feature.

24.1.7.2.3 TInstallOptions.Examples Property

Installs all the InterBase examples.

Pascal

```
property Examples: TExamplesOptions;
```

C++

```
__property TExamplesOptions Examples;
```

Description

Use Examples to indicates which of the InterBase examples should be installed.

When Examples includes exAPI, the MainComponents (see page 3866) property must include moClient and moDevelopment. When Examples includes exDev, MainComponents (see page 3866) must include moServer.

Note: You must install InterBase 6 to use this feature.

24.1.7.2.4 TInstallOptions.MainComponents Property

Installs the InterBase main components.

Pascal

```
property MainComponents: TMainOptions;
```

C++

```
__property TMainOptions MainComponents;
```

Description

Use MainComponents to indicate which of the InterBase main components to install. These can include the client, server, development tools, documentation, and ODBC driver and driver manager.

Note: You must install InterBase 6 to use this feature.

24.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCmdOption (see page 3868)	TCmdOption and TCmdOptions (see page 3873) indicate which of the command-line tools are installed.
	TConnectivityOption (see page 3869)	TConnectivityOption and TConnectivityOptions (see page 3873) identify InterBase connectivity clients.
	TErrorResult (see page 3869)	This is record IBInstall.TErrorResult.
	TExamplesOption (see page 3870)	TExamplesOption and TExamplesOptions (see page 3874) identify InterBase Examples clients.
	TIBInstallerError (see page 3870)	This is record IBInstall.TIBInstallerError.
	TMainOption (see page 3871)	TMainOption and TMainOptions (see page 3876) identify the main InterBase components.
	TStatusResult (see page 3872)	This is record IBInstall.TStatusResult.
	TWarningResult (see page 3872)	This is record IBInstall.TWarningResult.

24.2.1 IBInstall.TCmdOption Enumeration

TCmdOption and TCmdOptions (see page 3873) indicate which of the command-line tools are installed.

Pascal

```
TCmdOption = (
  cmDBMgmt,
  cmDBQuery,
  cmUsrMgmt
);
```

C++

```
enum TCmdOption {
  cmDBMgmt,
  cmDBQuery,
  cmUsrMgmt
};
```

File

IBInstall

Description

TCmdOption indicates a particular command-line tool for InterBase. TCmdOptions (see page 3873) is a set of TCmdOption values that indicates the tools that are installed. The following table lists the TCmdOption values:

Value Meaning

cmDBMgmt gbak and gfix database management tools.

cmDBQuery isql, gdef, and qli database query tools.

cmUsrMgmt gsec user management tool.

24.2.2 IBInstall.TConnectivityOption Enumeration

TConnectivityOption and TConnectivityOptions (see page 3873) identify InterBase connectivity clients.

Pascal

```
TConnectivityOption = (
  cnODBC,
  cnOLEDB,
  cnJDBC
);
```

C++

```
enum TConnectivityOption {
  cnODBC,
  cnOLEDB,
  cnJDBC
};
```

File

IBInstall

Description

TConnectivityOption indicates a particular connectivity client. TConnectivityOptions (see page 3873) is a set of TConnectivityOption values that indicates the clients that are installed. The following table lists the TConnectivityOption values:

Value Meaning

cnODBC The Open Database Connectivity client.

cnOLEDB The OLE DB connectivity client.

cnJDBC The InterClient Java (see page 4529) connectivity client.

24.2.3 IBInstall.TErrorResult Enumeration

Pascal

```
TErrorResult = (
  erAbort,
  erContinue,
  erRetry
);
```

C++

```
enum TErrorResult {
  erAbort,
  erContinue,
  erRetry
};
```

File

IBInstall

Description

This is record IBInstall.TErrorResult.

24.2.4 IBInstall.TExamplesOption Enumeration

TExamplesOption and TExamplesOptions (see page 3874) identify InterBase Examples clients.

Pascal

```
TExamplesOption = (
  exDB,
  exAPI
);
```

C++

```
enum TExamplesOption {
  exDB,
  exAPI
};
```

File

IBInstall

Description

TExamplesOption indicates a particular example that ships with InterBase. TExamplesOptions (see page 3874) is a set of TExamplesOption values that indicates the examples that are installed. The following table lists the TExamplesOption values:

Value Meaning

exAPI API, SQL, DSQL, and ESQL example files.

exDB Database examples.

24.2.5 IBInstall.TIBInstallerError Enumeration

Pascal

```
TIBInstallerError = (
  ieSuccess,
  ieDelphiException,
  ieNoOptionsSet,
  ieNoDestinationDirectory,
  ieNosourceDirectory,
  ieNoUninstallFile,
  ieOptionNeedsClient,
  ieOptionNeedsServer,
  ieInvalidOption,
  ieInvalidOnErrorResult,
  ieInvalidOnStatusResult
);
```

C++

```
enum TIBInstallerError {
  ieSuccess,
  ieDelphiException,
```

```
    ieNoOptionsSet,
    ieNoDestinationDirectory,
    ieNosourceDirectory,
    ieNoUninstallFile,
    ieOptionNeedsClient,
    ieOptionNeedsServer,
    ieInvalidOption,
    ieInvalidOnErrorResult,
    ieInvalidOnStatusResult
  };
```

File

IBInstall

Description

This is record IBInstall.TIBInstallerError.

24.2.6 IBInstall.TMainOption Enumeration

TMainOption and TMainOptions (see page 3876) identify the main InterBase components.

Pascal

```
  TMainOption = (
    moServer,
    moClient,
    moConServer,
    moGuiTools,
    moDocumentation,
    moDevelopment
  );
```

C++

```
  enum TMainOption {
    moServer,
    moClient,
    moConServer,
    moGuiTools,
    moDocumentation,
    moDevelopment
  };
```

File

IBInstall

Description

TMainOption indicates one of the main InterBase components. TMainOptions (see page 3876) is a set of TMainOption values that indicates the components that are installed. The following table lists the TMainOption values:

Value Meaning

moServer Server component, including: the server, license file (if present), message file, the Guardian, server configuration tool, the database information tools, UDF library, International character set library, and help files (Windows only).

moClient Client component, including the client library, license file, and message file.

moConServer Connectivity server, for future implementation.

moGuiTools Graphical interface tools, including IBConsole.

moDocumentation Documentation, including PDF files and indexes.

moDevelopment Development tools, including gpre, the import libraries, and the header files.

24.2.7 IBInstall.TStatusResult Enumeration

Pascal

```
TStatusResult = (
  srAbort,
  srContinue
);
```

C++

```
enum TStatusResult {
  srAbort,
  srContinue
};
```

File

IBInstall

Description

This is record IBInstall.TStatusResult.

24.2.8 IBInstall.TWarningResult Enumeration

Pascal

```
TWarningResult = (
  wrAbort,
  wrContinue
);
```

C++

```
enum TWarningResult {
  wrAbort,
  wrContinue
};
```

File

IBInstall

Description

This is record IBInstall.TWarningResult.

24.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCmdOptions (see page 3873)	This is type IBInstall.TCmdOptions.
TConnectivityOptions (see page 3873)	This is type IBInstall.TConnectivityOptions.
TExamplesOptions (see page 3874)	This is type IBInstall.TExamplesOptions.
TIBSetupOnError (see page 3874)	TIBSetupOnError is the type of event handlers that respond to installation errors.
TIBSetupOnStatus (see page 3874)	TIBSetupOnStatus is the type of event handlers that respond during the InterBase installation or de-installation process.
TIBSetupOnWarning (see page 3875)	TIBSetupOnWarning is the type of event handlers that respond to installation warnings.
TIscError (see page 3875)	This is type IBInstall.TIscError.
TMainOptions (see page 3876)	This is type IBInstall.TMainOptions.

24.3.1 IBInstall.TCmdOptions Type

Pascal

```
TCmdOptions = set of TCmdOption;
```

C++

```
TCmdOption TCmdOptions;
```

File

IBInstall

Description

This is type IBInstall.TCmdOptions.

24.3.2 IBInstall.TConnectivityOptions Type

Pascal

```
TConnectivityOptions = set of TConnectivityOption;
```

C++

```
TConnectivityOption TConnectivityOptions;
```

File

IBInstall

Description

This is type IBInstall.TConnectivityOptions.

24.3.3 IBInstall.TExamplesOptions Type

Pascal

```
TExamplesOptions = set of TExamplesOption;
```

C++

```
TExamplesOption TExamplesOptions;
```

File

IBInstall

Description

This is type IBInstall.TExamplesOptions.

24.3.4 IBInstall.TIBSetupOnError Type

TIBSetupOnError is the type of event handlers that respond to installation errors.

Pascal

```
TIBSetupOnError = function (Sender : TObject; IscCode : TIscError; ErrorMessage, ErrorComment
: string): TErrorResult of object;
```

C++

```
(Sender : TObject; IscCode : TIscError; ErrorMessage, ErrorComment : string): TErrorResult (
TIBSetupOnError)();
```

File

IBInstall

Description

Sender is the TIBSetup (see page 3857) component that encounters the exception.

IscCode is the error code for the problem that was encounted.

ErrorMessage is the error message fro the problem that occurred.

ErrorComment returns a string that will be displayed to the user.

The event handler is expected to return isc_fp_install_continue, isc_install_abort or isc_fp_install_retry.

24.3.5 IBInstall.TIBSetupOnStatus Type

TIBSetupOnStatus is the type of event handlers that respond during the InterBase installation or de-installation process.

Pascal

```
TIBSetupOnStatus = function (Sender : TObject; StatusComment : string): TStatusResult of
object;
```

C++

```
(Sender : TObject; StatusComment : string): TStatusResult ( TIBSetupOnStatus)();
```

File

IBInstall

Description

Sender is the TIBSetup (see page 3857) component that is installing or uninstalling.

StatusComment is a short string that indicates what operation is currently occurring. This can be used to supply the user with feedback on the installation process.

The event handler can return isc_fp_install_continue if the installation should continue, or isc_install_abort if the user cancels the installation process.

24.3.6 IBInstall.TIBSetupOnWarning Type

TIBSetupOnWarning is the type of event handlers that respond to installation warnings.

Pascal

```
TIBSetupOnWarning = function (Sender :TObject; WarningCode: TIscError; WarningMessage :
string): TWarningResult of object;
```

C++

```
(Sender :TObject; WarningCode: TIscError; WarningMessage : string): TWarningResult (
TIBSetupOnWarning)();
```

File

IBInstall

Description

Sender is the TIBSetup (see page 3857) component that encounters the warning.

WarningCode is the warning code for the situation that was encounted.

WarningMessage is the warning message for the situation that occurred.

The event handler must return either isc_fp_install_continue or isc_install_abort.

24.3.7 IBInstall.TIscError Type

Pascal

```
TIscError = MSG_NO;
```

C++

```
MSG_NO TIscError;
```

File

IBInstall

Description

This is type IBInstall.TIscError.

24.3.8 IBInstall.TMainOptions Type

Pascal

```
TMainOptions = set of TMainOption;
```

C++

```
TMainOption TMainOptions;
```

File

IBInstall

Description

This is type IBInstall.TMainOptions.

25 IBIntf Namespace

This is namespace IBIntf.

Topics

Name	Description
Functions (see page 3880)	The following table lists functions in this documentation.
Variables (see page 3891)	The following table lists variables in this documentation.

Functions

	Name	Description
	CheckIBInstallLoaded (see page 3880)	Attempts to load IBInstall.DLL into memory if it is not already loaded.
	CheckIBLoaded (see page 3881)	Attempts to load GDS32.DLL into memory if it is not already loaded.
	FreeIBInstallLibrary (see page 3881)	Frees the IBInstall.DLL from memory.
	FreeIBLibrary (see page 3881)	Frees the GDS32.DLL from memory.
	GetIBClientVersion (see page 3882)	Returns the major version number of the InterBase client.
	isc_decode_sql_date_stub (see page 3882)	This is function IBIntf.isc_decode_sql_date_stub.
	isc_decode_sql_time_stub (see page 3882)	This is function IBIntf.isc_decode_sql_time_stub.
	isc_decode_timestamp_stub (see page 3883)	This is function IBIntf.isc_decode_timestamp_stub.
	isc_encode_sql_date_stub (see page 3883)	This is function IBIntf.isc_encode_sql_date_stub.
	isc_encode_sql_time_stub (see page 3883)	This is function IBIntf.isc_encode_sql_time_stub.
	isc_encode_timestamp_stub (see page 3884)	This is function IBIntf.isc_encode_timestamp_stub.
	isc_install_clear_options_stub (see page 3884)	This is function IBIntf.isc_install_clear_options_stub.
	isc_install_execute_stub (see page 3884)	This is function IBIntf.isc_install_execute_stub.
	isc_install_get_info_stub (see page 3885)	This is function IBIntf.isc_install_get_info_stub.
	isc_install_get_message_stub (see page 3885)	This is function IBIntf.isc_install_get_message_stub.
	isc_install_load_external_text_stub (see page 3885)	This is function IBIntf.isc_install_load_external_text_stub.
	isc_install_precheck_stub (see page 3886)	This is function IBIntf.isc_install_precheck_stub.
	isc_install_set_option_stub (see page 3886)	This is function IBIntf.isc_install_set_option_stub.
	isc_install_unset_option_stub (see page 3886)	This is function IBIntf.isc_install_unset_option_stub.
	isc_rollback_retaining_stub (see page 3887)	This is function IBIntf.isc_rollback_retaining_stub.
	isc_service_attach_stub (see page 3887)	This is function IBIntf.isc_service_attach_stub.
	isc_service_detach_stub (see page 3887)	This is function IBIntf.isc_service_detach_stub.
	isc_service_query_stub (see page 3888)	This is function IBIntf.isc_service_query_stub.
	isc_service_start_stub (see page 3888)	This is function IBIntf.isc_service_start_stub.
	isc_uninstall_execute_stub (see page 3888)	This is function IBIntf.isc_uninstall_execute_stub.
	isc_uninstall_precheck_stub (see page 3889)	This is function IBIntf.isc_uninstall_precheck_stub.

	LoadIBInstallLibrary (see page 3889)	Loads IBInstall.DLL into memory.
	LoadIBLibrary (see page 3889)	Loads GDS32.DLL into memory.
	TryIBLoad (see page 3890)	This is function IBIntf.TryIBLoad.

Variables

Name	Description
BLOB_get (see page 3892)	This is variable IBIntf.BLOB_get.
BLOB_put (see page 3892)	This is variable IBIntf.BLOB_put.
isc_add_user (see page 3893)	This is variable IBIntf.isc_add_user.
isc_attach_database (see page 3893)	This is variable IBIntf.isc_attach_database.
isc_blob_info (see page 3893)	This is variable IBIntf.isc_blob_info.
isc_cancel_events (see page 3894)	This is variable IBIntf.isc_cancel_events.
isc_close_blob (see page 3894)	This is variable IBIntf.isc_close_blob.
isc_commit_retaining (see page 3894)	This is variable IBIntf.isc_commit_retaining.
isc_commit_transaction (see page 3895)	This is variable IBIntf.isc_commit_transaction.
isc_create_blob2 (see page 3895)	This is variable IBIntf.isc_create_blob2.
isc_database_info (see page 3895)	This is variable IBIntf.isc_database_info.
isc_decode_date (see page 3896)	This is variable IBIntf.isc_decode_date.
isc_decode_sql_date (see page 3896)	This is variable IBIntf.isc_decode_sql_date.
isc_decode_sql_time (see page 3896)	This is variable IBIntf.isc_decode_sql_time.
isc_decode_timestamp (see page 3897)	This is variable IBIntf.isc_decode_timestamp.
isc_delete_user (see page 3897)	This is variable IBIntf.isc_delete_user.
isc_detach_database (see page 3897)	This is variable IBIntf.isc_detach_database.
isc_drop_database (see page 3898)	This is variable IBIntf.isc_drop_database.
isc_dsql_alloc_statement2 (see page 3898)	This is variable IBIntf.isc_dsql_alloc_statement2.
isc_dsql_describe (see page 3898)	This is variable IBIntf.isc_dsql_describe.
isc_dsql_describe_bind (see page 3899)	This is variable IBIntf.isc_dsql_describe_bind.
isc_dsql_execute (see page 3899)	This is variable IBIntf.isc_dsql_execute.
isc_dsql_execute_immediate (see page 3899)	This is variable IBIntf.isc_dsql_execute_immediate.
isc_dsql_execute2 (see page 3900)	This is variable IBIntf.isc_dsql_execute2.
isc_dsql_fetch (see page 3900)	This is variable IBIntf.isc_dsql_fetch.
isc_dsql_free_statement (see page 3900)	This is variable IBIntf.isc_dsql_free_statement.
isc_dsql_prepare (see page 3901)	This is variable IBIntf.isc_dsql_prepare.
isc_dsql_set_cursor_name (see page 3901)	This is variable IBIntf.isc_dsql_set_cursor_name.
isc_dsql_sql_info (see page 3901)	This is variable IBIntf.isc_dsql_sql_info.
isc_encode_date (see page 3902)	This is variable IBIntf.isc_encode_date.
isc_encode_sql_date (see page 3902)	This is variable IBIntf.isc_encode_sql_date.
isc_encode_sql_time (see page 3902)	This is variable IBIntf.isc_encode_sql_time.
isc_encode_timestamp (see page 3903)	This is variable IBIntf.isc_encode_timestamp.
isc_event_block (see page 3903)	This is variable IBIntf.isc_event_block.
isc_event_counts (see page 3903)	This is variable IBIntf.isc_event_counts.
isc_free (see page 3904)	This is variable IBIntf.isc_free.
isc_get_segment (see page 3904)	This is variable IBIntf.isc_get_segment.
isc_install_clear_options (see page 3904)	This is variable IBIntf.isc_install_clear_options.
isc_install_execute (see page 3905)	This is variable IBIntf.isc_install_execute.
isc_install_get_info (see page 3905)	This is variable IBIntf.isc_install_get_info.
isc_install_get_message (see page 3905)	This is variable IBIntf.isc_install_get_message.
isc_install_load_external_text (see page 3906)	This is variable IBIntf.isc_install_load_external_text.
isc_install_precheck (see page 3906)	This is variable IBIntf.isc_install_precheck.
isc_install_set_option (see page 3906)	This is variable IBIntf.isc_install_set_option.
isc_install_unset_option (see page 3907)	This is variable IBIntf.isc_install_unset_option.
isc_interprete (see page 3907)	This is variable IBIntf.isc_interprete.
isc_modify_user (see page 3907)	This is variable IBIntf.isc_modify_user.
isc_open_blob2 (see page 3908)	This is variable IBIntf.isc_open_blob2.

isc_prepare_transaction (see page 3908)	This is variable IBIntf.isc_prepare_transaction.
isc_prepare_transaction2 (see page 3908)	This is variable IBIntf.isc_prepare_transaction2.
isc_put_segment (see page 3909)	This is variable IBIntf.isc_put_segment.
isc_que_events (see page 3909)	This is variable IBIntf.isc_que_events.
isc_rollback_retaining (see page 3909)	This is variable IBIntf.isc_rollback_retaining.
isc_rollback_transaction (see page 3910)	This is variable IBIntf.isc_rollback_transaction.
isc_service_attach (see page 3910)	This is variable IBIntf.isc_service_attach.
isc_service_detach (see page 3910)	This is variable IBIntf.isc_service_detach.
isc_service_query (see page 3911)	This is variable IBIntf.isc_service_query.
isc_service_start (see page 3911)	This is variable IBIntf.isc_service_start.
isc_sql_interprete (see page 3911)	This is variable IBIntf.isc_sql_interprete.
isc_sqlcode (see page 3912)	This is variable IBIntf.isc_sqlcode.
isc_start_multiple (see page 3912)	This is variable IBIntf.isc_start_multiple.
isc_uninstall_execute (see page 3912)	This is variable IBIntf.isc_uninstall_execute.
isc_uninstall_precheck (see page 3913)	This is variable IBIntf.isc_uninstall_precheck.
isc_vax_integer (see page 3913)	This is variable IBIntf.isc_vax_integer.

25.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckIBInstallLoaded (see page 3880)	Attempts to load IBInstall.DLL into memory if it is not already loaded.
	CheckIBLoaded (see page 3881)	Attempts to load GDS32.DLL into memory if it is not already loaded.
	FreeIBInstallLibrary (see page 3881)	Frees the IBInstall.DLL from memory.
	FreeIBLibrary (see page 3881)	Frees the GDS32.DLL from memory.
	GetIBClientVersion (see page 3882)	Returns the major version number of the InterBase client.
	isc_decode_sql_date_stub (see page 3882)	This is function IBIntf.isc_decode_sql_date_stub.
	isc_decode_sql_time_stub (see page 3882)	This is function IBIntf.isc_decode_sql_time_stub.
	isc_decode_timestamp_stub (see page 3883)	This is function IBIntf.isc_decode_timestamp_stub.
	isc_encode_sql_date_stub (see page 3883)	This is function IBIntf.isc_encode_sql_date_stub.
	isc_encode_sql_time_stub (see page 3883)	This is function IBIntf.isc_encode_sql_time_stub.
	isc_encode_timestamp_stub (see page 3884)	This is function IBIntf.isc_encode_timestamp_stub.
	isc_install_clear_options_stub (see page 3884)	This is function IBIntf.isc_install_clear_options_stub.
	isc_install_execute_stub (see page 3884)	This is function IBIntf.isc_install_execute_stub.
	isc_install_get_info_stub (see page 3885)	This is function IBIntf.isc_install_get_info_stub.
	isc_install_get_message_stub (see page 3885)	This is function IBIntf.isc_install_get_message_stub.
	isc_install_load_external_text_stub (see page 3885)	This is function IBIntf.isc_install_load_external_text_stub.
	isc_install_precheck_stub (see page 3886)	This is function IBIntf.isc_install_precheck_stub.
	isc_install_set_option_stub (see page 3886)	This is function IBIntf.isc_install_set_option_stub.
	isc_install_unset_option_stub (see page 3886)	This is function IBIntf.isc_install_unset_option_stub.
	isc_rollback_retaining_stub (see page 3887)	This is function IBIntf.isc_rollback_retaining_stub.
	isc_service_attach_stub (see page 3887)	This is function IBIntf.isc_service_attach_stub.
	isc_service_detach_stub (see page 3887)	This is function IBIntf.isc_service_detach_stub.
	isc_service_query_stub (see page 3888)	This is function IBIntf.isc_service_query_stub.
	isc_service_start_stub (see page 3888)	This is function IBIntf.isc_service_start_stub.
	isc_uninstall_execute_stub (see page 3888)	This is function IBIntf.isc_uninstall_execute_stub.
	isc_uninstall_precheck_stub (see page 3889)	This is function IBIntf.isc_uninstall_precheck_stub.
	LoadIBInstallLibrary (see page 3889)	Loads IBInstall.DLL into memory.
	LoadIBLibrary (see page 3889)	Loads GDS32.DLL into memory.
	TryIBLoad (see page 3890)	This is function IBIntf.TryIBLoad.

25.1.1 IBIntf.CheckIBInstallLoaded Function

Attempts to load IBInstall.DLL into memory if it is not already loaded.

Pascal

```
procedure CheckIBInstallLoaded;
```

C++

```
CheckIBInstallLoaded();
```

File

IBIntf

Description

Use the CheckIBInstallLibrary procedure to load IBInstall.DLL into memory if it is not already loaded. If unsuccessful, this procedure raises an exception.

25.1.2 IBIntf.CheckIBLoaded Function

Attempts to load GDS32.DLL into memory if it is not already loaded.

Pascal

```
procedure CheckIBLoaded;
```

C++

```
CheckIBLoaded();
```

File

IBIntf

Description

Use the CheckIBInstallLibrary procedure to load GDS32.DLL into memory if it is not already loaded. If unsuccessful, this procedure raises an exception.

25.1.3 IBIntf.FreeIBInstallLibrary Function

Frees the IBInstall.DLL from memory.

Pascal

```
procedure FreeIBInstallLibrary;
```

C++

```
FreeIBInstallLibrary();
```

File

IBIntf

Description

Use the FreeIBInstallLibrary procedure to free the previously loaded the IBInstall.DLL from memory.

25.1.4 IBIntf.FreeIBLibrary Function

Frees the GDS32.DLL from memory.

Pascal

```
procedure FreeIBLibrary;
```

C++

```
FreeIBLibrary();
```

File

IBIntf

Description

Use the FreeIBInstallLibrary (see page 3881) procedure to free the previously loaded the GDS32.DLL from memory.

25.1.5 IBIntf.GetIBClientVersion Function

Returns the major version number of the InterBase client.

Pascal

```
function GetIBClientVersion: Integer;
```

C++

```
int GetIBClientVersion();
```

File

IBIntf

Description

Use the GetIBClientVersion function to retrieve the major version number of the InterBase client.

25.1.6 IBIntf.isc_decode_sql_date_stub Function

Pascal

```
procedure isc_decode_sql_date_stub(ib_date: PISC_DATE; tm_date: PCTimeStructure); stdcall;
```

C++

```
__stdcall isc_decode_sql_date_stub(PISC_DATE ib_date, PCTimeStructure tm_date);
```

File

IBIntf

Description

This is function IBIntf.isc_decode_sql_date_stub.

25.1.7 IBIntf.isc_decode_sql_time_stub Function

Pascal

```
procedure isc_decode_sql_time_stub(ib_time: PISC_TIME; tm_date: PCTimeStructure); stdcall;
```

C++

```
__stdcall isc_decode_sql_time_stub(PISC_TIME ib_time, PCTimeStructure tm_date);
```

File

IBIntf

Description

This is function IBIntf.isc_decode_sql_time_stub.

25.1.8 IBIntf.isc_decode_timestamp_stub Function

Pascal

```
procedure isc_decode_timestamp_stub(ib_timestamp: PISC_TIMESTAMP; tm_date: PCTimeStructure);
stdcall;
```

C++

```
__stdcall isc_decode_timestamp_stub(PISC_TIMESTAMP ib_timestamp, PCTimeStructure tm_date);
```

File

IBIntf

Description

This is function IBIntf.isc_decode_timestamp_stub.

25.1.9 IBIntf.isc_encode_sql_date_stub Function

Pascal

```
procedure isc_encode_sql_date_stub(tm_date: PCTimeStructure; ib_date: PISC_DATE); stdcall;
```

C++

```
__stdcall isc_encode_sql_date_stub(PCTimeStructure tm_date, PISC_DATE ib_date);
```

File

IBIntf

Description

This is function IBIntf.isc_encode_sql_date_stub.

25.1.10 IBIntf.isc_encode_sql_time_stub Function

Pascal

```
procedure isc_encode_sql_time_stub(tm_date: PCTimeStructure; ib_time: PISC_TIME); stdcall;
```

C++

```
__stdcall isc_encode_sql_time_stub(PCTimeStructure tm_date, PISC_TIME ib_time);
```

File

IBIntf

Description

This is function IBIntf.isc_encode_sql_time_stub.

25.1.11 IBIntf.isc_encode_timestamp_stub Function

Pascal

```
procedure isc_encode_timestamp_stub(tm_date: PCTimeStructure; ib_timestamp: PISC_TIMESTAMP);
stdcall;
```

C++

```
__stdcall isc_encode_timestamp_stub(PCTimeStructure tm_date, PISC_TIMESTAMP ib_timestamp);
```

File

IBIntf

Description

This is function IBIntf.isc_encode_timestamp_stub.

25.1.12 IBIntf.isc_install_clear_options_stub Function

Pascal

```
function isc_install_clear_options_stub(hOption: POPTIONS_HANDLE): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_clear_options_stub(POPTIONS_HANDLE hOption);
```

File

IBIntf

Description

This is function IBIntf.isc_install_clear_options_stub.

25.1.13 IBIntf.isc_install_execute_stub Function

Pascal

```
function isc_install_execute_stub(hOption: OPTIONS_HANDLE; src_dir: TEXT; dest_dir: TEXT;
status_func: FP_STATUS; status_data: pointer; error_func: FP_ERROR; error_data: pointer;
uninstal_file_name: TEXT): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_execute_stub(OPTIONS_HANDLE hOption, TEXT src_dir, TEXT dest_dir,
FP_STATUS status_func, void * status_data, FP_ERROR error_func, void * error_data, TEXT
uninstal_file_name);
```

File

IBIntf

Description

This is function IBIntf.isc_install_execute_stub.

25.1.14 IBIntf.isc_install_get_info_stub Function

Pascal

```
function isc_install_get_info_stub(info_type: integer; option: OPT; info_buffer: Pointer;
buf_len: Cardinal): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_get_info_stub(int info_type, OPT option, void * info_buffer,
unsigned buf_len);
```

File

IBIntf

Description

This is function IBIntf.isc_install_get_info_stub.

25.1.15 IBIntf.isc_install_get_message_stub Function

Pascal

```
function isc_install_get_message_stub(hOption: OPTIONS_HANDLE; message_no: MSG_NO;
message_txt: Pointer; message_len: Cardinal): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_get_message_stub(OPTIONS_HANDLE hOption, MSG_NO message_no, void
* message_txt, unsigned message_len);
```

File

IBIntf

Description

This is function IBIntf.isc_install_get_message_stub.

25.1.16 IBIntf.isc_install_load_external_text_stub Function

Pascal

```
function isc_install_load_external_text_stub(msg_file_name: TEXT): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_load_external_text_stub(TEXT msg_file_name);
```

File

IBIntf

Description

This is function IBIntf.isc_install_load_external_text_stub.

25.1.17 IBIntf.isc_install_precheck_stub Function

Pascal

```
function isc_install_precheck_stub(hOption: OPTIONS_HANDLE; src_dir: TEXT; dest_dir: TEXT):
MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_precheck_stub(OPTIONS_HANDLE hOption, TEXT src_dir, TEXT
dest_dir);
```

File

IBIntf

Description

This is function IBIntf.isc_install_precheck_stub.

25.1.18 IBIntf.isc_install_set_option_stub Function

Pascal

```
function isc_install_set_option_stub(hOption: POPTIONS_HANDLE; option: OPT): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_set_option_stub(POPTIONS_HANDLE hOption, OPT option);
```

File

IBIntf

Description

This is function IBIntf.isc_install_set_option_stub.

25.1.19 IBIntf.isc_install_unset_option_stub Function

Pascal

```
function isc_install_unset_option_stub(hOption: POPTIONS_HANDLE; option: OPT): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_install_unset_option_stub(POPTIONS_HANDLE hOption, OPT option);
```

File

IBIntf

Description

This is function IBIntf.isc_install_unset_option_stub.

25.1.20 IBIntf.isc_rollback_retaining_stub Function

Pascal

```
function isc_rollback_retaining_stub(status_vector: PISC_STATUS; tran_handle: PISC_TR_HANDLE):
ISC_STATUS; stdcall;
```

C++

```
__stdcall ISC_STATUS isc_rollback_retaining_stub(PISC_STATUS status_vector, PISC_TR_HANDLE
tran_handle);
```

File

IBIntf

Description

This is function IBIntf.isc_rollback_retaining_stub.

25.1.21 IBIntf.isc_service_attach_stub Function

Pascal

```
function isc_service_attach_stub(status_vector: PISC_STATUS; isc_arg2: UShort; isc_arg3:
PChar; service_handle: PISC_SVC_HANDLE; isc_arg5: UShort; isc_arg6: PChar): ISC_STATUS;
stdcall;
```

C++

```
__stdcall ISC_STATUS isc_service_attach_stub(PISC_STATUS status_vector, UShort isc_arg2, const
char * isc_arg3, PISC_SVC_HANDLE service_handle, UShort isc_arg5, const char * isc_arg6);
```

File

IBIntf

Description

This is function IBIntf.isc_service_attach_stub.

25.1.22 IBIntf.isc_service_detach_stub Function

Pascal

```
function isc_service_detach_stub(status_vector: PISC_STATUS; service_handle: PISC_SVC_HANDLE):
ISC_STATUS; stdcall;
```

C++

```
__stdcall ISC_STATUS isc_service_detach_stub(PISC_STATUS status_vector, PISC_SVC_HANDLE
service_handle);
```

File

IBIntf

Description

This is function IBIntf.isc_service_detach_stub.

25.1.23 IBIntf.isc_service_query_stub Function

Pascal

```
function isc_service_query_stub(status_vector: PISC_STATUS; service_handle: PISC_SVC_HANDLE;
recv_handle: PISC_SVC_HANDLE; isc_arg4: UShort; isc_arg5: PChar; isc_arg6: UShort; isc_arg7:
PChar; isc_arg8: UShort; isc_arg9: PChar): ISC_STATUS; stdcall;
```

C++

```
__stdcall ISC_STATUS isc_service_query_stub(PISC_STATUS status_vector, PISC_SVC_HANDLE
service_handle, PISC_SVC_HANDLE recv_handle, UShort isc_arg4, const char * isc_arg5, UShort
isc_arg6, const char * isc_arg7, UShort isc_arg8, const char * isc_arg9);
```

File

IBIntf

Description

This is function IBIntf.isc_service_query_stub.

25.1.24 IBIntf.isc_service_start_stub Function

Pascal

```
function isc_service_start_stub(status_vector: PISC_STATUS; service_handle: PISC_SVC_HANDLE;
recv_handle: PISC_SVC_HANDLE; isc_arg4: UShort; isc_arg5: PChar): ISC_STATUS; stdcall;
```

C++

```
__stdcall ISC_STATUS isc_service_start_stub(PISC_STATUS status_vector, PISC_SVC_HANDLE
service_handle, PISC_SVC_HANDLE recv_handle, UShort isc_arg4, const char * isc_arg5);
```

File

IBIntf

Description

This is function IBIntf.isc_service_start_stub.

25.1.25 IBIntf.isc_uninstall_execute_stub Function

Pascal

```
function isc_uninstall_execute_stub(uninstall_file_name: TEXT; status_func: FP_STATUS;
status_data: pointer; error_func: FP_ERROR; error_data: pointer): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_uninstall_execute_stub(TEXT uninstall_file_name, FP_STATUS status_func,
void * status_data, FP_ERROR error_func, void * error_data);
```

File

IBIntf

Description

This is function IBIntf.isc_uninstall_execute_stub.

25.1.26 IBIntf.isc_uninstall_precheck_stub Function

Pascal

```
function isc_uninstall_precheck_stub(uninstall_file_name: TEXT): MSG_NO; stdcall;
```

C++

```
__stdcall MSG_NO isc_uninstall_precheck_stub(TEXT uninstall_file_name);
```

File

IBIntf

Description

This is function IBIntf.isc_uninstall_precheck_stub.

25.1.27 IBIntf.LoadIBInstallLibrary Function

Loads IBInstall.DLL into memory.

Pascal

```
procedure LoadIBInstallLibrary;
```

C++

```
LoadIBInstallLibrary();
```

File

IBIntf

Description

Use the LoadIBInstallLibrary procedure to load the IBInstall.DLL into memory and resolve the respective entry points into to the respective procedure pointer in the unit.

25.1.28 IBIntf.LoadIBLibrary Function

Loads GDS32.DLL into memory.

Pascal

```
procedure LoadIBLibrary;
```

C++

```
LoadIBLibrary();
```

File

IBIntf

Description

Use the LoadIBLibrary procedure to load the GDS32.DLL into memory and resolve the respective entry points into to the respective procedure pointer in the unit. In addition, it loads the InterBase entry points, if they are available.

25.1.29 IBIntf.TryIBLoad Function

Pascal

```
function TryIBLoad: Boolean;
```

C++

```
Boolean TryIBLoad();
```

File

IBIntf

Description

This is function IBIntf.TryIBLoad.

25.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BLOB_get (see page 3892)	This is variable IBIntf.BLOB_get.
BLOB_put (see page 3892)	This is variable IBIntf.BLOB_put.
isc_add_user (see page 3893)	This is variable IBIntf.isc_add_user.
isc_attach_database (see page 3893)	This is variable IBIntf.isc_attach_database.
isc_blob_info (see page 3893)	This is variable IBIntf.isc_blob_info.
isc_cancel_events (see page 3894)	This is variable IBIntf.isc_cancel_events.
isc_close_blob (see page 3894)	This is variable IBIntf.isc_close_blob.
isc_commit_retaining (see page 3894)	This is variable IBIntf.isc_commit_retaining.
isc_commit_transaction (see page 3895)	This is variable IBIntf.isc_commit_transaction.
isc_create_blob2 (see page 3895)	This is variable IBIntf.isc_create_blob2.
isc_database_info (see page 3895)	This is variable IBIntf.isc_database_info.
isc_decode_date (see page 3896)	This is variable IBIntf.isc_decode_date.
isc_decode_sql_date (see page 3896)	This is variable IBIntf.isc_decode_sql_date.
isc_decode_sql_time (see page 3896)	This is variable IBIntf.isc_decode_sql_time.
isc_decode_timestamp (see page 3897)	This is variable IBIntf.isc_decode_timestamp.
isc_delete_user (see page 3897)	This is variable IBIntf.isc_delete_user.
isc_detach_database (see page 3897)	This is variable IBIntf.isc_detach_database.
isc_drop_database (see page 3898)	This is variable IBIntf.isc_drop_database.
isc_dsql_alloc_statement2 (see page 3898)	This is variable IBIntf.isc_dsql_alloc_statement2.
isc_dsql_describe (see page 3898)	This is variable IBIntf.isc_dsql_describe.
isc_dsql_describe_bind (see page 3899)	This is variable IBIntf.isc_dsql_describe_bind.
isc_dsql_execute (see page 3899)	This is variable IBIntf.isc_dsql_execute.
isc_dsql_execute_immediate (see page 3899)	This is variable IBIntf.isc_dsql_execute_immediate.
isc_dsql_execute2 (see page 3900)	This is variable IBIntf.isc_dsql_execute2.
isc_dsql_fetch (see page 3900)	This is variable IBIntf.isc_dsql_fetch.
isc_dsql_free_statement (see page 3900)	This is variable IBIntf.isc_dsql_free_statement.
isc_dsql_prepare (see page 3901)	This is variable IBIntf.isc_dsql_prepare.
isc_dsql_set_cursor_name (see page 3901)	This is variable IBIntf.isc_dsql_set_cursor_name.
isc_dsql_sql_info (see page 3901)	This is variable IBIntf.isc_dsql_sql_info.
isc_encode_date (see page 3902)	This is variable IBIntf.isc_encode_date.
isc_encode_sql_date (see page 3902)	This is variable IBIntf.isc_encode_sql_date.
isc_encode_sql_time (see page 3902)	This is variable IBIntf.isc_encode_sql_time.
isc_encode_timestamp (see page 3903)	This is variable IBIntf.isc_encode_timestamp.
isc_event_block (see page 3903)	This is variable IBIntf.isc_event_block.
isc_event_counts (see page 3903)	This is variable IBIntf.isc_event_counts.
isc_free (see page 3904)	This is variable IBIntf.isc_free.
isc_get_segment (see page 3904)	This is variable IBIntf.isc_get_segment.
isc_install_clear_options (see page 3904)	This is variable IBIntf.isc_install_clear_options.
isc_install_execute (see page 3905)	This is variable IBIntf.isc_install_execute.
isc_install_get_info (see page 3905)	This is variable IBIntf.isc_install_get_info.
isc_install_get_message (see page 3905)	This is variable IBIntf.isc_install_get_message.
isc_install_load_external_text (see page 3906)	This is variable IBIntf.isc_install_load_external_text.
isc_install_precheck (see page 3906)	This is variable IBIntf.isc_install_precheck.
isc_install_set_option (see page 3906)	This is variable IBIntf.isc_install_set_option.

isc_install_unset_option (see page 3907)	This is variable IBIntf.isc_install_unset_option.
isc_interprete (see page 3907)	This is variable IBIntf.isc_interprete.
isc_modify_user (see page 3907)	This is variable IBIntf.isc_modify_user.
isc_open_blob2 (see page 3908)	This is variable IBIntf.isc_open_blob2.
isc_prepare_transaction (see page 3908)	This is variable IBIntf.isc_prepare_transaction.
isc_prepare_transaction2 (see page 3908)	This is variable IBIntf.isc_prepare_transaction2.
isc_put_segment (see page 3909)	This is variable IBIntf.isc_put_segment.
isc_que_events (see page 3909)	This is variable IBIntf.isc_que_events.
isc_rollback_retaining (see page 3909)	This is variable IBIntf.isc_rollback_retaining.
isc_rollback_transaction (see page 3910)	This is variable IBIntf.isc_rollback_transaction.
isc_service_attach (see page 3910)	This is variable IBIntf.isc_service_attach.
isc_service_detach (see page 3910)	This is variable IBIntf.isc_service_detach.
isc_service_query (see page 3911)	This is variable IBIntf.isc_service_query.
isc_service_start (see page 3911)	This is variable IBIntf.isc_service_start.
isc_sql_interprete (see page 3911)	This is variable IBIntf.isc_sql_interprete.
isc_sqlcode (see page 3912)	This is variable IBIntf.isc_sqlcode.
isc_start_multiple (see page 3912)	This is variable IBIntf.isc_start_multiple.
isc_uninstall_execute (see page 3912)	This is variable IBIntf.isc_uninstall_execute.
isc_uninstall_precheck (see page 3913)	This is variable IBIntf.isc_uninstall_precheck.
isc_vax_integer (see page 3913)	This is variable IBIntf.isc_vax_integer.

25.2.1 IBIntf.BLOB_get Variable

Pascal

```
BLOB_get: TBLOB_get;
```

C++

```
TBLOB_get BLOB_get;
```

File

IBIntf

Description

This is variable IBIntf.BLOB_get.

25.2.2 IBIntf.BLOB_put Variable

Pascal

```
BLOB_put: TBLOB_put;
```

C++

```
TBLOB_put BLOB_put;
```

File

IBIntf

Description

This is variable IBIntf.BLOB_put.

25.2.3 IBIntf.isc_add_user Variable

Pascal

```
isc_add_user: Tisc_add_user;
```

C++

```
Tisc_add_user isc_add_user;
```

File

IBIntf

Description

This is variable IBIntf.isc_add_user.

25.2.4 IBIntf.isc_attach_database Variable

Pascal

```
isc_attach_database: Tisc_attach_database;
```

C++

```
Tisc_attach_database isc_attach_database;
```

File

IBIntf

Description

This is variable IBIntf.isc_attach_database.

25.2.5 IBIntf.isc_blob_info Variable

Pascal

```
isc_blob_info: Tisc_blob_info;
```

C++

```
Tisc_blob_info isc_blob_info;
```

File

IBIntf

Description

This is variable IBIntf.isc_blob_info.

25.2.6 IBIntf.isc_cancel_events Variable

Pascal

```
isc_cancel_events: Tisc_cancel_events;
```

C++

```
Tisc_cancel_events isc_cancel_events;
```

File

IBIntf

Description

This is variable IBIntf.isc_cancel_events.

25.2.7 IBIntf.isc_close_blob Variable

Pascal

```
isc_close_blob: Tisc_close_blob;
```

C++

```
Tisc_close_blob isc_close_blob;
```

File

IBIntf

Description

This is variable IBIntf.isc_close_blob.

25.2.8 IBIntf.isc_commit_retaining Variable

Pascal

```
isc_commit_retaining: Tisc_commit_retaining;
```

C++

```
Tisc_commit_retaining isc_commit_retaining;
```

File

IBIntf

Description

This is variable IBIntf.isc_commit_retaining.

25.2.9 IBIntf.isc_commit_transaction Variable

Pascal

```
isc_commit_transaction: Tisc_commit_transaction;
```

C++

```
Tisc_commit_transaction isc_commit_transaction;
```

File

IBIntf

Description

This is variable IBIntf.isc_commit_transaction.

25.2.10 IBIntf.isc_create_blob2 Variable

Pascal

```
isc_create_blob2: Tisc_create_blob2;
```

C++

```
Tisc_create_blob2 isc_create_blob2;
```

File

IBIntf

Description

This is variable IBIntf.isc_create_blob2.

25.2.11 IBIntf.isc_database_info Variable

Pascal

```
isc_database_info: Tisc_database_info;
```

C++

```
Tisc_database_info isc_database_info;
```

File

IBIntf

Description

This is variable IBIntf.isc_database_info.

25.2.12 IBIntf.isc_decode_date Variable

Pascal

```
isc_decode_date: Tisc_decode_date;
```

C++

```
Tisc_decode_date isc_decode_date;
```

File

IBIntf

Description

This is variable IBIntf.isc_decode_date.

25.2.13 IBIntf.isc_decode_sql_date Variable

Pascal

```
isc_decode_sql_date: Tisc_decode_sql_date;
```

C++

```
Tisc_decode_sql_date isc_decode_sql_date;
```

File

IBIntf

Description

This is variable IBIntf.isc_decode_sql_date.

25.2.14 IBIntf.isc_decode_sql_time Variable

Pascal

```
isc_decode_sql_time: Tisc_decode_sql_time;
```

C++

```
Tisc_decode_sql_time isc_decode_sql_time;
```

File

IBIntf

Description

This is variable IBIntf.isc_decode_sql_time.

25.2.15 IBIntf.isc_decode_timestamp Variable

Pascal

```
isc_decode_timestamp: Tisc_decode_timestamp;
```

C++

```
Tisc_decode_timestamp isc_decode_timestamp;
```

File

IBIntf

Description

This is variable IBIntf.isc_decode_timestamp.

25.2.16 IBIntf.isc_delete_user Variable

Pascal

```
isc_delete_user: Tisc_delete_user;
```

C++

```
Tisc_delete_user isc_delete_user;
```

File

IBIntf

Description

This is variable IBIntf.isc_delete_user.

25.2.17 IBIntf.isc_detach_database Variable

Pascal

```
isc_detach_database: Tisc_detach_database;
```

C++

```
Tisc_detach_database isc_detach_database;
```

File

IBIntf

Description

This is variable IBIntf.isc_detach_database.

25.2.18 IBIntf.isc_drop_database Variable

Pascal

```
isc_drop_database: Tisc_drop_database;
```

C++

```
Tisc_drop_database isc_drop_database;
```

File

IBIntf

Description

This is variable IBIntf.isc_drop_database.

25.2.19 IBIntf.isc_dsql_alloc_statement2 Variable

Pascal

```
isc_dsql_alloc_statement2: Tisc_dsql_alloc_statement2;
```

C++

```
Tisc_dsql_alloc_statement2 isc_dsql_alloc_statement2;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_alloc_statement2.

25.2.20 IBIntf.isc_dsql_describe Variable

Pascal

```
isc_dsql_describe: Tisc_dsql_describe;
```

C++

```
Tisc_dsql_describe isc_dsql_describe;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_describe.

25.2.21 IBIntf.isc_dsql_describe_bind Variable

Pascal

```
isc_dsql_describe_bind: Tisc_dsql_describe_bind;
```

C++

```
Tisc_dsql_describe_bind isc_dsql_describe_bind;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_describe_bind.

25.2.22 IBIntf.isc_dsql_execute Variable

Pascal

```
isc_dsql_execute: Tisc_dsql_execute;
```

C++

```
Tisc_dsql_execute isc_dsql_execute;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_execute.

25.2.23 IBIntf.isc_dsql_execute_immediate Variable

Pascal

```
isc_dsql_execute_immediate: Tisc_dsql_execute_immediate;
```

C++

```
Tisc_dsql_execute_immediate isc_dsql_execute_immediate;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_execute_immediate.

25.2.24 IBIntf.isc_dsql_execute2 Variable

Pascal

```
isc_dsql_execute2: Tisc_dsql_execute2;
```

C++

```
Tisc_dsql_execute2 isc_dsql_execute2;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_execute2.

25.2.25 IBIntf.isc_dsql_fetch Variable

Pascal

```
isc_dsql_fetch: Tisc_dsql_fetch;
```

C++

```
Tisc_dsql_fetch isc_dsql_fetch;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_fetch.

25.2.26 IBIntf.isc_dsql_free_statement Variable

Pascal

```
isc_dsql_free_statement: Tisc_dsql_free_statement;
```

C++

```
Tisc_dsql_free_statement isc_dsql_free_statement;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_free_statement.

25.2.27 IBIntf.isc_dsql_prepare Variable

Pascal

```
isc_dsql_prepare: Tisc_dsql_prepare;
```

C++

```
Tisc_dsql_prepare isc_dsql_prepare;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_prepare.

25.2.28 IBIntf.isc_dsql_set_cursor_name Variable

Pascal

```
isc_dsql_set_cursor_name: Tisc_dsql_set_cursor_name;
```

C++

```
Tisc_dsql_set_cursor_name isc_dsql_set_cursor_name;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_set_cursor_name.

25.2.29 IBIntf.isc_dsql_sql_info Variable

Pascal

```
isc_dsql_sql_info: Tisc_dsql_sql_info;
```

C++

```
Tisc_dsql_sql_info isc_dsql_sql_info;
```

File

IBIntf

Description

This is variable IBIntf.isc_dsql_sql_info.

25.2.30 IBIntf.isc_encode_date Variable

Pascal

```
isc_encode_date: Tisc_encode_date;
```

C++

```
Tisc_encode_date isc_encode_date;
```

File

IBIntf

Description

This is variable IBIntf.isc_encode_date.

25.2.31 IBIntf.isc_encode_sql_date Variable

Pascal

```
isc_encode_sql_date: Tisc_encode_sql_date;
```

C++

```
Tisc_encode_sql_date isc_encode_sql_date;
```

File

IBIntf

Description

This is variable IBIntf.isc_encode_sql_date.

25.2.32 IBIntf.isc_encode_sql_time Variable

Pascal

```
isc_encode_sql_time: Tisc_encode_sql_time;
```

C++

```
Tisc_encode_sql_time isc_encode_sql_time;
```

File

IBIntf

Description

This is variable IBIntf.isc_encode_sql_time.

25.2.33 IBIntf.isc_encode_timestamp Variable

Pascal

```
isc_encode_timestamp: Tisc_encode_timestamp;
```

C++

```
Tisc_encode_timestamp isc_encode_timestamp;
```

File

IBIntf

Description

This is variable IBIntf.isc_encode_timestamp.

25.2.34 IBIntf.isc_event_block Variable

Pascal

```
isc_event_block: Tisc_event_block;
```

C++

```
Tisc_event_block isc_event_block;
```

File

IBIntf

Description

This is variable IBIntf.isc_event_block.

25.2.35 IBIntf.isc_event_counts Variable

Pascal

```
isc_event_counts: Tisc_event_counts;
```

C++

```
Tisc_event_counts isc_event_counts;
```

File

IBIntf

Description

This is variable IBIntf.isc_event_counts.

25.2.36 IBIntf.isc_free Variable

Pascal

```
isc_free: Tisc_free;
```

C++

```
Tisc_free isc_free;
```

File

IBIntf

Description

This is variable IBIntf.isc_free.

25.2.37 IBIntf.isc_get_segment Variable

Pascal

```
isc_get_segment: Tisc_get_segment;
```

C++

```
Tisc_get_segment isc_get_segment;
```

File

IBIntf

Description

This is variable IBIntf.isc_get_segment.

25.2.38 IBIntf.isc_install_clear_options Variable

Pascal

```
isc_install_clear_options: Tisc_install_clear_options;
```

C++

```
Tisc_install_clear_options isc_install_clear_options;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_clear_options.

25.2.39 IBIntf.isc_install_execute Variable

Pascal

```
isc_install_execute: Tisc_install_execute;
```

C++

```
Tisc_install_execute isc_install_execute;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_execute.

25.2.40 IBIntf.isc_install_get_info Variable

Pascal

```
isc_install_get_info: Tisc_install_get_info;
```

C++

```
Tisc_install_get_info isc_install_get_info;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_get_info.

25.2.41 IBIntf.isc_install_get_message Variable

Pascal

```
isc_install_get_message: Tisc_install_get_message;
```

C++

```
Tisc_install_get_message isc_install_get_message;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_get_message.

25.2.42 IBIntf.isc_install_load_external_text Variable

Pascal

```
isc_install_load_external_text: Tisc_install_load_external_text;
```

C++

```
Tisc_install_load_external_text isc_install_load_external_text;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_load_external_text.

25.2.43 IBIntf.isc_install_precheck Variable

Pascal

```
isc_install_precheck: Tisc_install_precheck;
```

C++

```
Tisc_install_precheck isc_install_precheck;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_precheck.

25.2.44 IBIntf.isc_install_set_option Variable

Pascal

```
isc_install_set_option: Tisc_install_set_option;
```

C++

```
Tisc_install_set_option isc_install_set_option;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_set_option.

25.2.45 IBIntf.isc_install_unset_option Variable

Pascal

```
isc_install_unset_option: Tisc_install_unset_option;
```

C++

```
Tisc_install_unset_option isc_install_unset_option;
```

File

IBIntf

Description

This is variable IBIntf.isc_install_unset_option.

25.2.46 IBIntf.isc_interprete Variable

Pascal

```
isc_interprete: Tisc_interprete;
```

C++

```
Tisc_interprete isc_interprete;
```

File

IBIntf

Description

This is variable IBIntf.isc_interprete.

25.2.47 IBIntf.isc_modify_user Variable

Pascal

```
isc_modify_user: Tisc_modify_user;
```

C++

```
Tisc_modify_user isc_modify_user;
```

File

IBIntf

Description

This is variable IBIntf.isc_modify_user.

25.2.48 IBIntf.isc_open_blob2 Variable

Pascal

```
isc_open_blob2: Tisc_open_blob2;
```

C++

```
Tisc_open_blob2 isc_open_blob2;
```

File

IBIntf

Description

This is variable IBIntf.isc_open_blob2.

25.2.49 IBIntf.isc_prepare_transaction Variable

Pascal

```
isc_prepare_transaction: Tisc_prepare_transaction;
```

C++

```
Tisc_prepare_transaction isc_prepare_transaction;
```

File

IBIntf

Description

This is variable IBIntf.isc_prepare_transaction.

25.2.50 IBIntf.isc_prepare_transaction2 Variable

Pascal

```
isc_prepare_transaction2: Tisc_prepare_transaction2;
```

C++

```
Tisc_prepare_transaction2 isc_prepare_transaction2;
```

File

IBIntf

Description

This is variable IBIntf.isc_prepare_transaction2.

25.2.51 IBIntf.isc_put_segment Variable

Pascal

```
isc_put_segment: Tisc_put_segment;
```

C++

```
Tisc_put_segment isc_put_segment;
```

File

IBIntf

Description

This is variable IBIntf.isc_put_segment.

25.2.52 IBIntf.isc_que_events Variable

Pascal

```
isc_que_events: Tisc_que_events;
```

C++

```
Tisc_que_events isc_que_events;
```

File

IBIntf

Description

This is variable IBIntf.isc_que_events.

25.2.53 IBIntf.isc_rollback_retaining Variable

Pascal

```
isc_rollback_retaining: Tisc_rollback_retaining;
```

C++

```
Tisc_rollback_retaining isc_rollback_retaining;
```

File

IBIntf

Description

This is variable IBIntf.isc_rollback_retaining.

25.2.54 IBIntf.isc_rollback_transaction Variable

Pascal

```
isc_rollback_transaction: Tisc_rollback_transaction;
```

C++

```
Tisc_rollback_transaction isc_rollback_transaction;
```

File

IBIntf

Description

This is variable IBIntf.isc_rollback_transaction.

25.2.55 IBIntf.isc_service_attach Variable

Pascal

```
isc_service_attach: Tisc_service_attach;
```

C++

```
Tisc_service_attach isc_service_attach;
```

File

IBIntf

Description

This is variable IBIntf.isc_service_attach.

25.2.56 IBIntf.isc_service_detach Variable

Pascal

```
isc_service_detach: Tisc_service_detach;
```

C++

```
Tisc_service_detach isc_service_detach;
```

File

IBIntf

Description

This is variable IBIntf.isc_service_detach.

25.2.57 IBIntf.isc_service_query Variable

Pascal

```
isc_service_query: Tisc_service_query;
```

C++

```
Tisc_service_query isc_service_query;
```

File

IBIntf

Description

This is variable IBIntf.isc_service_query.

25.2.58 IBIntf.isc_service_start Variable

Pascal

```
isc_service_start: Tisc_service_start;
```

C++

```
Tisc_service_start isc_service_start;
```

File

IBIntf

Description

This is variable IBIntf.isc_service_start.

25.2.59 IBIntf.isc_sql_interprete Variable

Pascal

```
isc_sql_interprete: Tisc_sql_interprete;
```

C++

```
Tisc_sql_interprete isc_sql_interprete;
```

File

IBIntf

Description

This is variable IBIntf.isc_sql_interprete.

25.2.60 IBIntf.isc_sqlcode Variable

Pascal

```
isc_sqlcode: Tisc_sqlcode;
```

C++

```
Tisc_sqlcode isc_sqlcode;
```

File

IBIntf

Description

This is variable IBIntf.isc_sqlcode.

25.2.61 IBIntf.isc_start_multiple Variable

Pascal

```
isc_start_multiple: Tisc_start_multiple;
```

C++

```
Tisc_start_multiple isc_start_multiple;
```

File

IBIntf

Description

This is variable IBIntf.isc_start_multiple.

25.2.62 IBIntf.isc_uninstall_execute Variable

Pascal

```
isc_uninstall_execute: Tisc_uninstall_execute;
```

C++

```
Tisc_uninstall_execute isc_uninstall_execute;
```

File

IBIntf

Description

This is variable IBIntf.isc_uninstall_execute.

25.2.63 IBIntf.isc_uninstall_precheck Variable

Pascal

```
isc_uninstall_precheck: Tisc_uninstall_precheck;
```

C++

```
Tisc_uninstall_precheck isc_uninstall_precheck;
```

File

IBIntf

Description

This is variable IBIntf.isc_uninstall_precheck.

25.2.64 IBIntf.isc_vax_integer Variable

Pascal

```
isc_vax_integer: Tisc_vax_integer;
```

C++

```
Tisc_vax_integer isc_vax_integer;
```

File

IBIntf

Description

This is variable IBIntf.isc_vax_integer.

26 IBQuery Namespace

This is namespace IBQuery.

Topics

Name	Description
Classes (see page 3916)	The following table lists classes in this documentation.

Classes

	Name	Description
	TIBQuery (see page 3916)	TIBQuery executes an InterBase SQL (see page 3932) statement.

26.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBQuery (see page 3916)	TIBQuery executes an InterBase SQL (see page 3932) statement.

26.1.1 TIBQuery Class

TIBQuery executes an InterBase SQL (see page 3932) statement.

Class Hierarchy

Pascal

```
TIBQuery = class(TIBCustomDataSet);
```

C++

```
class TIBQuery : public TIBCustomDataSet;
```

File

IBQuery

Description

Use TIBQuery to access one or more tables in a database using SQL (see page 3932) statements. The TIBQuery component provides a read-only dataset, and adapts well to the InterBase client/server environment. To update the result set that TIBQuery represents, use this component in conjunction with a TIBUpdateSQL component.

Query components are useful because they can

- Access more than one table at a time (called a 'join' in SQL (see page 3932)).
- Automatically access a subset of rows and columns in its underlying table(s), rather than always returning all rows and columns.

Note: TIBQuery is of particular importance to the development of scalable database applications. If there is any chance that an application built to run against local databases will be scaled to a remote SQL (see page 3932) database server in the future, use TIBQuery components from the start to ensure easier scaling later.

See Also

Types of Datasets

Getting Started with InterBase Express

26.1.1.1 TIBQuery Members

The following tables list the members exposed by TIBQuery.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.
	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.
V	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
V	DefineProperties (see page 3922)	This is DefineProperties, a member of class TIBQuery.
V	Disconnect (see page 3923)	This is Disconnect, a member of class TIBQuery.
V	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
V	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
V	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
V	DoOnNewRecord (see page 3614)	This is DoOnNewRecord, a member of class TIBCustomDataSet.
V	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GenerateQueryForLiveUpdate (see page 3923)	This is GenerateQueryForLiveUpdate, a member of class TIBQuery.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
V	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
V	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
V	GetCanModify (see page 3615)	This is GetCanModify, a member of class TIBCustomDataSet.
V	GetDataSource (see page 3616)	This is GetDataSource, a member of class TIBCustomDataSet.
V	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
	GetParamsCount (see page 3924)	This is GetParamsCount, a member of class TIBQuery.
V	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
V	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
V	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
V	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
V	InitFieldDefs (see page 3924)	This is InitFieldDefs, a member of class TIBQuery.
V	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
V	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
V	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
V	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.
V	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
V	InternalClose (see page 3619)	This is InternalClose, a member of class TIBCustomDataSet.
V	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
V	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
V	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
V	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
V	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
V	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
V	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
V	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
V	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
V	InternalOpen (see page 3924)	This is InternalOpen, a member of class TIBQuery.
V	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
V	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
V	InternalRefresh (see page 3622)	This is InternalRefresh, a member of class TIBCustomDataSet.
V	InternalRefreshRow (see page 3622)	This is InternalRefreshRow, a member of class TIBCustomDataSet.
V	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
V	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.

	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.
	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.
	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
	PSExecute (see page 3925)	This is PSExecute, a member of class TIBQuery.
	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
	PSGetParams (see page 3925)	This is PSGetParams, a member of class TIBQuery.
	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
	PSGetTableName (see page 3925)	This is PSGetTableName, a member of class TIBQuery.
	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.
	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
	PSSetCommandText (see page 3926)	This is PSSetCommandText, a member of class TIBQuery.
	PSSetParams (see page 3926)	This is PSSetParams, a member of class TIBQuery.
	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 3926)	This is SetFiltered, a member of class TIBQuery.
	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.

Protected Properties

	Name	Description
	DeleteSQL (see page 3639)	Holds the SQL statement used to delete rows from the dataset.
	InsertSQL (see page 3640)	Holds the SQL statement used to insert rows into the dataset.
R	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
	ModifySQL (see page 3641)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
R	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.
R	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
R	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
R	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
R	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	RefreshSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
R	SelectStmtHandle (see page 3645)	Gets the handle for the SELECT statement.
R	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Methods

	Name	Description
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.

	BatchInput (see page 3922)	Executes the parameterized query in SQL (see page 3932) for input in the referenced input object.
	BatchOutput (see page 3922)	Outputs the selected query in SQL (see page 3932) to the referenced OutputObject.
V	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.
	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
V	CompareBookmarks (see page 3611)	Compares two bookmarks.
V	Create (see page 3921)	Creates an instance of a query component.
V	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
V	Destroy (see page 3921)	Destroys the instance of a query.
	ExecSQL (see page 3923)	Executes the SQL (see page 3932) statement for the query.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
V	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
V	GetDetailLinkFields (see page 3923)	Fills lists with the master and detail fields of the link.
V	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.
V	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
V	Locate (see page 3624)	Searches the dataset for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
V	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.
	ParamByName (see page 3924)	Accesses parameter information based on a specified parameter name.
V	Post (see page 3626)	Writes a modified record to the database server.
	Prepare (see page 3925)	Sends a query to the server for optimization prior to execution.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
	UnPrepare (see page 3926)	Frees the resources allocated for a previously prepared query.
V	UpdateStatus (see page 3632)	Reports the update status for the current record.

Public Properties

	Name	Description
R	DBHandle (see page 3639)	Specifies the database handle for the dataset.
	GenerateParamNames (see page 3929)	Generates a list of parameter names for the query.
	LiveMode (see page 3929)	Indicates what types of edits, if any, can be made to the data in the dataset.
R	ParamCount (see page 3930)	Indicates the current number of parameters for the query.
R	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
	Prepared (see page 3931)	Determines whether or not a query is prepared for execution.
R	RowsAffected (see page 3931)	Returns the number of rows operated upon by the latest query execution.
	StatementType (see page 3932)	Returns the statement type of the QSelect query.
R	StmtHandle (see page 3932)	Identifies the statement handle for the query.
R	Text (see page 3932)	Points to the actual text of the SQL (see page 3932) query.
R	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 3934)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3934)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3934)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3935)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 3935)	Occurs when the database associated with this dataset is freed.
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.
	TransactionFree (see page 3935)	Occurs when the transaction that includes this dataset's query is freed.

Published Properties

	Name	Description
	Active (see page 3927)	This is Active, a member of class TIBQuery.
	AfterCancel (see page 3634)	This is AfterCancel, a member of class TIBCustomDataSet.
	AfterClose (see page 3634)	This is AfterClose, a member of class TIBCustomDataSet.
	AfterDelete (see page 3634)	This is AfterDelete, a member of class TIBCustomDataSet.
	AfterEdit (see page 3634)	This is AfterEdit, a member of class TIBCustomDataSet.
	AfterInsert (see page 3635)	This is AfterInsert, a member of class TIBCustomDataSet.
	AfterOpen (see page 3635)	This is AfterOpen, a member of class TIBCustomDataSet.
	AfterPost (see page 3635)	This is AfterPost, a member of class TIBCustomDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3635)	This is AfterScroll, a member of class TIBCustomDataSet.
	AutoCalcFields (see page 3636)	This is AutoCalcFields, a member of class TIBCustomDataSet.
	BeforeCancel (see page 3636)	This is BeforeCancel, a member of class TIBCustomDataSet.
	BeforeClose (see page 3636)	This is BeforeClose, a member of class TIBCustomDataSet.
	BeforeDelete (see page 3636)	This is BeforeDelete, a member of class TIBCustomDataSet.
	BeforeEdit (see page 3637)	This is BeforeEdit, a member of class TIBCustomDataSet.
	BeforeInsert (see page 3637)	This is BeforeInsert, a member of class TIBCustomDataSet.
	BeforeOpen (see page 3637)	This is BeforeOpen, a member of class TIBCustomDataSet.
	BeforePost (see page 3637)	This is BeforePost, a member of class TIBCustomDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3638)	This is BeforeScroll, a member of class TIBCustomDataSet.
	BufferChunks (see page 3927)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3928)	Specifies whether cached updates are enabled for a dataset.
	Constraints (see page 3928)	This is Constraints, a member of class TIBQuery.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	DataSource (see page 3928)	This is DataSource, a member of class TIBQuery.
	Filtered (see page 3929)	Specifies whether filtering is active for a dataset.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	GeneratorField (see page 3929)	Specifies whether the dataset should refresh its data every time it posts a record.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3641)	This is OnCalcFields, a member of class TIBCustomDataSet.
	OnDeleteError (see page 3642)	This is OnDeleteError, a member of class TIBCustomDataSet.
	OnEditError (see page 3642)	This is OnEditError, a member of class TIBCustomDataSet.
	OnFilterRecord (see page 3930)	This is OnFilterRecord, a member of class TIBQuery.
	OnNewRecord (see page 3642)	This is OnNewRecord, a member of class TIBCustomDataSet.
	OnPostError (see page 3642)	This is OnPostError, a member of class TIBCustomDataSet.
	ParamCheck (see page 3930)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
	Params (see page 3931)	Contains the parameters for a query's SQL (see page 3932) statement.
	SQL (see page 3932)	Contains the text of the SQL statement to execute for the query.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.

	UniDirectional (see page 3933)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 3933)	Specifies the update object component used to update a read-only result set.

26.1.1.2 TIBQuery.Create Constructor

Creates an instance of a query component.

Pascal

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

C++

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

Description

Call Create to instantiate a query at runtime. Query components placed in forms or data modules at design time are created automatically.

Create calls its inherited Create constructor, creates an empty SQL (see page 3932) statement list, creates an empty parameter list, sets the OnChange event handler for the SQL (see page 3932) statement list, sets the ParamCheck (see page 3930) property to True, and sets the RowsAffected (see page 3931) property to -1.

26.1.1.3 TIBQuery.Destroy Destructor

Destroys the instance of a query.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBQuery();
```

Description

Do not call Destroy directly. Instead call Free to verify that the query is not already freed before calling Destroy. Destroy disconnects from the server, frees the SQL (see page 3932) statement list and the parameter list, and then calls its inherited destructor.

26.1.1.4 TIBQuery Methods

The methods of the TIBQuery class are listed here.

Protected Methods

	Name	Description
	DefineProperties (see page 3922)	This is DefineProperties, a member of class TIBQuery.
	Disconnect (see page 3923)	This is Disconnect, a member of class TIBQuery.
	GenerateQueryForLiveUpdate (see page 3923)	This is GenerateQueryForLiveUpdate, a member of class TIBQuery.
	GetParamsCount (see page 3924)	This is GetParamsCount, a member of class TIBQuery.
	InitFieldDefs (see page 3924)	This is InitFieldDefs, a member of class TIBQuery.
	InternalOpen (see page 3924)	This is InternalOpen, a member of class TIBQuery.
	PSExecute (see page 3925)	This is PSExecute, a member of class TIBQuery.
	PSGetParams (see page 3925)	This is PSGetParams, a member of class TIBQuery.
	PSGetTableName (see page 3925)	This is PSGetTableName, a member of class TIBQuery.

	PSSetCommandText (see page 3926)	This is PSSetCommandText, a member of class TIBQuery.
	PSSetParams (see page 3926)	This is PSSetParams, a member of class TIBQuery.
	SetFiltered (see page 3926)	This is SetFiltered, a member of class TIBQuery.

Public Methods

	Name	Description
	BatchInput (see page 3922)	Executes the parameterized query in SQL (see page 3932) for input in the referenced input object.
	BatchOutput (see page 3922)	Outputs the selected query in SQL (see page 3932) to the referenced OutputObject.
	ExecSQL (see page 3923)	Executes the SQL (see page 3932) statement for the query.
	GetDetailLinkFields (see page 3923)	Fills lists with the master and detail fields of the link.
	ParamByName (see page 3924)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 3925)	Sends a query to the server for optimization prior to execution.
	UnPrepare (see page 3926)	Frees the resources allocated for a previously prepared query.

26.1.1.4.1 TIBQuery.BatchInput Method

Executes the parameterized query in SQL (see page 3932) for input in the referenced input object.

Pascal

```
procedure BatchInput(InputObject: TIBBatchInput);
```

C++

```
__fastcall BatchInput(TIBBatchInput InputObject);
```

Description

Call BatchInput to execute the parameterized query in SQL (see page 3932) for input in the referenced input object.

26.1.1.4.2 TIBQuery.BatchOutput Method

Outputs the selected query in SQL (see page 3932) to the referenced OutputObject.

Pascal

```
procedure BatchOutput(OutputObject: TIBBatchOutput);
```

C++

```
__fastcall BatchOutput(TIBBatchOutput OutputObject);
```

Description

Call BatchOutput to output the selected query in SQL (see page 3932) to the referenced OutputObject.

26.1.1.4.3 TIBQuery.DefineProperties Method

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

This is DefineProperties, a member of class TIBQuery.

26.1.1.4.4 TIBQuery.Disconnect Method

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TIBQuery.

26.1.1.4.5 TIBQuery.ExecSQL Method

Executes the SQL (see page 3932) statement for the query.

Pascal

```
procedure ExecSQL;
```

C++

```
__fastcall ExecSQL();
```

Description

Call ExecSQL to execute the SQL (see page 3932) statement currently assigned to the SQL (see page 3932) property. Use ExecSQL to execute queries that do not return a cursor to data (such as INSERT, UPDATE, DELETE, and CREATE TABLE).

Note: For SELECT statements, call Open instead of ExecSQL.

ExecSQL prepares the statement in SQL (see page 3932) property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare (see page 3925) before calling ExecSQL for the first time.

26.1.1.4.6 TIBQuery.GenerateQueryForLiveUpdate Method

Pascal

```
function GenerateQueryForLiveUpdate: Boolean;
```

C++

```
__fastcall Boolean GenerateQueryForLiveUpdate();
```

Description

This is GenerateQueryForLiveUpdate, a member of class TIBQuery.

26.1.1.4.7 TIBQuery.GetDetailLinkFields Method

Fills lists with the master and detail fields of the link.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

Creates two lists of TFields from the master-detail relationship between two tables; one containing the master fields, and the

other containing the detail fields.

26.1.1.4.8 TIBQuery.GetParamsCount Method

Pascal

```
function GetParamsCount: Word;
```

C++

```
__fastcall Word GetParamsCount();
```

Description

This is GetParamsCount, a member of class TIBQuery.

26.1.1.4.9 TIBQuery.InitFieldDefs Method

Pascal

```
procedure InitFieldDefs; override;
```

C++

```
virtual __fastcall InitFieldDefs();
```

Description

This is InitFieldDefs, a member of class TIBQuery.

26.1.1.4.10 TIBQuery.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TIBQuery.

26.1.1.4.11 TIBQuery.ParamByName Method

Accesses parameter information based on a specified parameter name.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const UnicodeString Value);
```

Description

Call ParamByName to set or use parameter information for a specific parameter based on its name.

Value is the name of the parameter for which to retrieve information.

ParamByName is primarily used to set an parameter's value at runtime. For example, the following statement retrieves the current value of a parameter called 'Contact' into an edit box:

```
Edit1.Text := Query1.ParamByName('Contact').AsString;
```

```
Edit1->Text = Query1->ParamByName("Contact")->AsString;
```

26.1.1.4.12 TIBQuery.Prepare Method

Sends a query to the server for optimization prior to execution.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call Prepare to have the remote database server allocate resources for the query and to perform additional optimizations. Calling Prepare before executing a query improves application performance.

A query is automatically prepared before being executed. A query that was automatically prepared will then be automatically unprepared after it executes. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The UnPrepare (see page 3926) method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

26.1.1.4.13 TIBQuery.PSExecute Method

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

This is PSExecute, a member of class TIBQuery.

26.1.1.4.14 TIBQuery.PSGetParams Method

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

This is PSGetParams, a member of class TIBQuery.

26.1.1.4.15 TIBQuery.PSGetTableName Method

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetTableName();
```

Description

This is PSGetTableName, a member of class TIBQuery.

26.1.1.4.16 TIBQuery.PSSetCommandText Method

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const UnicodeString CommandText);
```

Description

This is PSSetCommandText, a member of class TIBQuery.

26.1.1.4.17 TIBQuery.PSSetParams Method

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

This is PSSetParams, a member of class TIBQuery.

26.1.1.4.18 TIBQuery.SetFiltered Method

Pascal

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

C++

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

Description

This is SetFiltered, a member of class TIBQuery.

26.1.1.4.19 TIBQuery.UnPrepare Method

Frees the resources allocated for a previously prepared query.

Pascal

```
procedure UnPrepare;
```

C++

```
__fastcall UnPrepare();
```

Description

Call UnPrepare to free the resources allocated for a previously prepared query on the server and client sides.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

26.1.1.5 TIBQuery Properties

The properties of the TIBQuery class are listed here.

Public Properties

	Name	Description
	GenerateParamNames (see page 3929)	Generates a list of parameter names for the query.
	LiveMode (see page 3929)	Indicates what types of edits, if any, can be made to the data in the dataset.
R	ParamCount (see page 3930)	Indicates the current number of parameters for the query.
	Prepared (see page 3931)	Determines whether or not a query is prepared for execution.
R	RowsAffected (see page 3931)	Returns the number of rows operated upon by the latest query execution.
	StatementType (see page 3932)	Returns the statement type of the QSelect query.
R	StmtHandle (see page 3932)	Identifies the statement handle for the query.
R	Text (see page 3932)	Points to the actual text of the SQL (see page 3932) query.

Published Properties

	Name	Description
	Active (see page 3927)	This is Active, a member of class TIBQuery.
	BufferChunks (see page 3927)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3928)	Specifies whether cached updates are enabled for a dataset.
	Constraints (see page 3928)	This is Constraints, a member of class TIBQuery.
	DataSource (see page 3928)	This is DataSource, a member of class TIBQuery.
	Filtered (see page 3929)	Specifies whether filtering is active for a dataset.
	GeneratorField (see page 3929)	Specifies whether the dataset should refresh its data every time it posts a record.
	OnFilterRecord (see page 3930)	This is OnFilterRecord, a member of class TIBQuery.
	ParamCheck (see page 3930)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
	Params (see page 3931)	Contains the parameters for a query's SQL (see page 3932) statement.
	SQL (see page 3932)	Contains the text of the SQL statement to execute for the query.
	UniDirectional (see page 3933)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 3933)	Specifies the update object component used to update a read-only result set.

26.1.1.5.1 TIBQuery.Active Property

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

This is Active, a member of class TIBQuery.

26.1.1.5.2 TIBQuery.BufferChunks Property

Specifies the dataset buffer chunk size.

Pascal

```
property BufferChunks: Integer;
```

C++

```
__property int BufferChunks;
```

Description

Use BufferChunks to get or set the number of records for which the dataset allocates buffer space at any time. When the dataset's buffer is full, trying to fetch an additional record causes the dataset to reallocate the buffer so that it has enough memory to hold an additional BufferChunks records.

Note: When Unidirectional is true, BufferChunks has no meaning.

26.1.1.5.3 TIBQuery.CachedUpdates Property

Specifies whether cached updates are enabled for a dataset.

Pascal

```
property CachedUpdates: Boolean;
```

C++

```
__property Boolean CachedUpdates;
```

Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is true, cached updates are enabled. If CachedUpdates is false, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Note: **Note**: Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset.

26.1.1.5.4 TIBQuery.Constraints Property

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

This is Constraints, a member of class TIBQuery.

26.1.1.5.5 TIBQuery.DataSource Property

Pascal

```
property DataSource;
```

C++

```
__property DataSource;
```

Description

This is DataSource, a member of class TIBQuery.

26.1.1.5.6 TIBQuery.Filtered Property

Specifies whether filtering is active for a dataset.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether dataset filtering is in effect. If Filtered is true, then filtering is active. Otherwise Filtered is false. To apply filter conditions specified in the Filter property or the OnFilterRecord (see page 3930) event handler, set Filtered to true. Filtered is set to false by default.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

26.1.1.5.7 TIBQuery.GenerateParamNames Property

Generates a list of parameter names for the query.

Pascal

```
property GenerateParamNames: Boolean;
```

C++

```
__property Boolean GenerateParamNames;
```

Description

Set GenerateParamNames to true to have the query generate a list of parameter names.

26.1.1.5.8 TIBQuery.GeneratorField Property

Specifies whether the dataset should refresh its data every time it posts a record.

Pascal

```
property GeneratorField: TIBGeneratorField;
```

C++

```
__property TIBGeneratorField GeneratorField;
```

Description

Use GeneratorField to specify a generator that automatically provides the value for one of the fields in the dataset. Use the properties of the TIBGeneratorField object to indicate which field's value should be set and how that value should be calculated.

Warning: Do not set the value of GeneratorField. To change the value of this property, change the properties of the TIBGeneratorField object that is the value of this property.

26.1.1.5.9 TIBQuery.LiveMode Property

Indicates what types of edits, if any, can be made to the data in the dataset.

Pascal

```
property LiveMode: TLiveModes;
```

C++

```
__property TLiveModes LiveMode;
```

Description

Read LiveMode to determine what types of edits should be allowed. LiveMode is set automatically to restrict the capabilities of the dataset to reflect the user's rights.

Note: For read-only datasets, such as TIBQuery, LiveMode is always an empty set.

26.1.1.5.10 TIBQuery.OnFilterRecord Property

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

This is OnFilterRecord, a member of class TIBQuery.

26.1.1.5.11 TIBQuery.ParamCheck Property

Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether the Params property is cleared and regenerated if an application modifies the SQL in this dataset's query. By default ParamCheck is true, meaning that the Params property is automatically regenerated at runtime. When ParamCheck is true, the proper number of parameters is guaranteed to be generated for the current SQL statement.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement that are not parameters for the TIBCustomDataSet. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TIBCustomDataSet executing the DDL statement.

26.1.1.5.12 TIBQuery.ParamCount Property

Indicates the current number of parameters for the query.

Pascal

```
property ParamCount: Word;
```

C++

```
__property Word ParamCount;
```

Description

Inspect ParamCount to determine how many parameters are in the Params (see page 3931) property. If the ParamCheck (see page 3930) property is true, ParamCount always corresponds to the number of actual parameters in the SQL (see page 3932) statement for the query.

26.1.1.5.13 TIBQuery.Params Property

Contains the parameters for a query's SQL (see page 3932) statement.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the collection editor for the Params property to set parameter information). Params is a zero-based array of TParams parameter records. Index specifies the array element to access.

Note: An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName (see page 3924). ParamByName (see page 3924) cannot, however, be used to change a parameter's data type or name.

26.1.1.5.14 TIBQuery.Prepared Property

Determines whether or not a query is prepared for execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Examine Prepared to determine if a query is already prepared for execution. If Prepared is true, the query is prepared, and if Prepared is false, the query is not prepared. While a query need not be prepared before execution, execution performance is enhanced if the query is prepared beforehand, particularly if it is a parameterized query that is executed more than once using the same parameter values.

Note: An application can change the current setting of Prepared to prepare or unprepare a query. If Prepared is true, setting it to false calls the UnPrepare (see page 3926) method to unprepare the query. If Prepared is false, setting it to true calls the Prepare (see page 3925) method to prepare the query. Generally, however, it is better programming practice to call Prepare (see page 3925) and UnPrepare (see page 3926) directly. These methods automatically update the Prepared property.

26.1.1.5.15 TIBQuery.RowsAffected Property

Returns the number of rows operated upon by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

Check RowsAffected to determine how many rows were updated or deleted by the last query operation. If RowsAffected is -1, the query did not update or delete any rows.

26.1.1.5.16 TIBQuery.SQL Property

Contains the text of the SQL statement to execute for the query.

Pascal

```
property SQL: TStrings;
```

C++

```
__property TStrings SQL;
```

Description

Use SQL to provide the SQL statement that a query component executes when its ExecSQL (see page 3923) or Open method is called. At design time the SQL property can be edited by invoking the String List editor in the Object Inspector.

The SQL property may contain only one complete SQL statement at a time.

26.1.1.5.17 TIBQuery.StatementType Property

Returns the statement type of the QSelect query.

Pascal

```
property StatementType: TIBSQLTypes;
```

C++

```
__property TIBSQLTypes StatementType;
```

Description

Use StatementType to determine the statement type of a QSelect query.

26.1.1.5.18 TIBQuery.StmtHandle Property

Identifies the statement handle for the query.

Pascal

```
property StmtHandle: TISC_STMT_HANDLE;
```

C++

```
__property TISC_STMT_HANDLE StmtHandle;
```

Description

Retrieve StmtHandle if an application makes a direct call to the InterBase server, bypassing the methods of TIBQuery (see page 3916). Some API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

26.1.1.5.19 TIBQuery.Text Property

Points to the actual text of the SQL (see page 3932) query.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Text is a read-only property that can be examined to determine the actual contents of SQL (see page 3932) statement. For parameterized queries, Text contains the SQL (see page 3932) statement with parameters replaced by the parameter substitution symbol (?) in place of actual parameter values.

In general there should be no need to examine the Text property. To access or change the SQL (see page 3932) statement for the query, use the SQL (see page 3932) property. To examine or modify parameters, use the Params (see page 3931) property.

26.1.1.5.20 TIBQuery.UniDirectional Property

Determines whether bidirectional cursors are enabled for the dataset.

Pascal

```
property UniDirectional: Boolean;
```

C++

```
__property Boolean UniDirectional;
```

Description

Use UniDirectional to determine whether bidirectional cursors are enabled for this dataset.

Note: Do not confuse the UniDirectional property with the IsUniDirectional property inherited from TDataSet. IsUniDirectional indicates whether the dataset is a dbExpress dataset, which, in addition to using a unidirectional cursor, does not buffer records. UniDirectional is only concerned with whether the cursor permits backward or random navigation.

26.1.1.5.21 TIBQuery.UpdateObject Property

Specifies the update object component used to update a read-only result set.

Pascal

```
property UpdateObject: TIBDataSetUpdateObject;
```

C++

```
__property TIBDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TIBDataSetUpdateObject component to use in an application that must be able to update a read-only result set.

In some cases, such as when using TIBQuery, the dataset does not automatically update records. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

TIBDataSetUpdateObject is especially useful for complicated updates from an OnUpdateRecord event.

26.1.1.6 TIBQuery Events

The events of the TIBQuery class are listed here.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 3934)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 3934)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 3934)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 3935)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 3935)	Occurs when the database associated with this dataset is freed.
	TransactionFree (see page 3935)	Occurs when the transaction that includes this dataset's query is freed.

26.1.1.6.1 TIBQuery.AfterDatabaseDisconnect Event

Occurs immediately after the connection to the associated database is closed.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to respond when the dataset closes its connection to the database specified by the Database property.

26.1.1.6.2 TIBQuery.AfterTransactionEnd Event

Occurs immediately after a transaction that includes this dataset's query ends.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to respond when the transaction specified by the Transaction property ends.

26.1.1.6.3 TIBQuery.BeforeDatabaseDisconnect Event

Occurs immediately before the connection to the associated database is closed.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to respond when the dataset is about to close its connection to the database

specified by the Database property.

26.1.1.6.4 TIBQuery.BeforeTransactionEnd Event

Occurs immediately before a transaction that includes this dataset's query ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to respond when the transaction specified by the Transaction property is about to end.

26.1.1.6.5 TIBQuery.DatabaseFree Event

Occurs when the database associated with this dataset is freed.

Pascal

```
property DatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent DatabaseFree;
```

Description

Write a DatabaseFree event handler to respond when the database specified by the Database property is freed.

26.1.1.6.6 TIBQuery.TransactionFree Event

Occurs when the transaction that includes this dataset's query is freed.

Pascal

```
property TransactionFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent TransactionFree;
```

Description

Write a TransactionFree event handler to respond when the transaction specified by the Transaction property is freed.

27 IBServices Namespace

This is namespace IBServices.

Topics

Name	Description
Classes (see page 3939)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4018)	The following table lists structs, records, enums in this documentation.
Types (see page 4027)	The following table lists types in this documentation.
Constants (see page 4032)	The following table lists constants in this documentation.

Classes

	Name	Description
	TConfigFileData (see page 3939)	This is class IBServices.TConfigFileData.
	TConfigParams (see page 3941)	TConfigFileData (see page 3939) and TConfigParams hold configuration information about an InterBase server
	TDatabaseInfo (see page 3944)	TDatabaseInfo stores information about an InterBase database.
	TIBBackupRestoreService (see page 3946)	TIBBackupRestoreService is the ancestor object from which TIBBackupService (see page 3948) and TIBRestoreService (see page 3979) are derived.
	TIBBackupService (see page 3948)	TIBBackupService allows you to back up your database.
	TIBConfigService (see page 3952)	TIBConfigService configures the database parameters
	TIBControlAndQueryService (see page 3957)	TIBControlAndQueryService is the ancestor object from which the log, statistical, validation, security, and backup and restore IBService components descend.
	TIBControlService (see page 3961)	TIBControlService is the ancestor object from which the structured and unstructured IBService components descend.
	TIBCustomService (see page 3965)	TIBCustomService is the ancestor object from which all IBService components descend.
	TIBLicensingService (see page 3973)	TIBLicensingService configures the licensing parameters
	TIBLogService (see page 3977)	TIBLogService returns the contents of the interbase.log file from a server.
	TIBRestoreService (see page 3979)	TIBRestoreService allows you to restore your database.
	TIBSecurityService (see page 3983)	TIBSecurityService enables you to manage user access to the InterBase server.
	TIBServerProperties (see page 3992)	TIBServerProperties returns database server information.
	TIBStatisticalService (see page 3998)	TIBStatisticalService allows you to view database statistics.
	TIBValidationService (see page 4001)	TIBValidationService enables you to validate your database and reconcile your database transactions.
	TLicenseInfo (see page 4006)	TLicenseInfo stores information about licensed users.
	TLicenseMaskInfo (see page 4008)	TLicenseMaskInfo indicates the software activation certificate options enabled on the server.
	TLimboTransactionInfo (see page 4010)	TLimboTransactionInfo stores information about a limbo transaction.

	TUserInfo (see page 4013)	TUserInfo stores information about an InterBase user for the security service.
	TVersionInfo (see page 4015)	TVersionInfo represents version information about an InterBase server.

Constants

Name	Description
DefaultBufferSize (see page 4032)	This is constant IBServices.DefaultBufferSize.
SPBConstantNames (see page 4032)	This is constant IBServices.SPBConstantNames.
SPBConstantValues (see page 4033)	This is constant IBServices.SPBConstantValues.
SPBPrefix (see page 4033)	This is constant IBServices.SPBPrefix.

Structs, Records, Enums

	Name	Description
	TBackupOption (see page 4018)	This is record IBServices.TBackupOption.
	TLicensingAction (see page 4019)	TLicensingAction indicates whether to add or remove InterBase software activation certificates.
	TOutputBufferOption (see page 4019)	This is record IBServices.TOutputBufferOption.
	TPropertyOption (see page 4020)	This is record IBServices.TPropertyOption.
	TProtocol (see page 4020)	TProtocol identifies a network protocol.
	TRestoreOption (see page 4021)	This is record IBServices.TRestoreOption.
	TSecurityAction (see page 4021)	TSecurityAction indicates what operation the InterBase security service should perform.
	TSecurityModifyParam (see page 4022)	This is record IBServices.TSecurityModifyParam.
	TShutdownMode (see page 4022)	TShutdownMode indicates what steps to take when shutting down the InterBase database.
	TStatOption (see page 4023)	This is record IBServices.TStatOption.
	TTransactionAction (see page 4023)	TTransactionAction indicates the action a transaction can take when it times out.
	TTransactionAdvise (see page 4024)	This is record IBServices.TTransactionAdvise.
	TTransactionGlobalAction (see page 4024)	TTransactionGlobalAction indicates the action to take on a set of transactions.
	TTransactionState (see page 4025)	This is record IBServices.TTransactionState.
	TValidateOption (see page 4026)	This is record IBServices.TValidateOption.

Types

Name	Description
TBackupOptions (see page 4027)	TBackupOption (see page 4018) and TBackupOptions indicate how TIBBackupService (see page 3948) performs database backup.
TLoginEvent (see page 4028)	This is type IBServices.TLoginEvent.
TPropertyOptions (see page 4028)	TPropertyOption (see page 4020) and TPropertyOptions indicate groups of InterBase server properties.
TRestoreOptions (see page 4029)	TRestoreOption (see page 4021) and TRestoreOptions indicate what operations take place when restoring an InterBase database.
TSecurityModifyParams (see page 4030)	This is type IBServices.TSecurityModifyParams.
TStatOptions (see page 4030)	TStatOption (see page 4023) and TStatOptions indicate the different statistical information available about a database.
TValidateOptions (see page 4031)	TValidateOption (see page 4026) and TValidateOptions indicate what operations the validation service should perform when restoring a database.

27.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TConfigFileData (see page 3939)	This is class IBServices.TConfigFileData.
	TConfigParams (see page 3941)	TConfigFileData (see page 3939) and TConfigParams hold configuration information about an InterBase server
	TDatabaseInfo (see page 3944)	TDatabaseInfo stores information about an InterBase database.
	TIBBackupRestoreService (see page 3946)	TIBBackupRestoreService is the ancestor object from which TIBBackupService (see page 3948) and TIBRestoreService (see page 3979) are derived.
	TIBBackupService (see page 3948)	TIBBackupService allows you to back up your database.
	TIBConfigService (see page 3952)	TIBConfigService configures the database parameters
	TIBControlAndQueryService (see page 3957)	TIBControlAndQueryService is the ancestor object from which the log, statistical, validation, security, and backup and restore IBService components descend.
	TIBControlService (see page 3961)	TIBControlService is the ancestor object from which the structured and unstructured IBService components descend.
	TIBCustomService (see page 3965)	TIBCustomService is the ancestor object from which all IBService components descend.
	TIBLicensingService (see page 3973)	TIBLicensingService configures the licensing parameters
	TIBLogService (see page 3977)	TIBLogService returns the contents of the interbase.log file from a server.
	TIBRestoreService (see page 3979)	TIBRestoreService allows you to restore your database.
	TIBSecurityService (see page 3983)	TIBSecurityService enables you to manage user access to the InterBase server.
	TIBServerProperties (see page 3992)	TIBServerProperties returns database server information.
	TIBStatisticalService (see page 3998)	TIBStatisticalService allows you to view database statistics.
	TIBValidationService (see page 4001)	TIBValidationService enables you to validate your database and reconcile your database transactions.
	TLicenseInfo (see page 4006)	TLicenseInfo stores information about licensed users.
	TLicenseMaskInfo (see page 4008)	TLicenseMaskInfo indicates the software activation certificate options enabled on the server.
	TLimboTransactionInfo (see page 4010)	TLimboTransactionInfo stores information about a limbo transaction.
	TUserInfo (see page 4013)	TUserInfo stores information about an InterBase user for the security service.
	TVersionInfo (see page 4015)	TVersionInfo represents version information about an InterBase server.

27.1.1 TConfigFileData Class

Class Hierarchy

Pascal

```
TConfigFileData = class;
```

C++

```
class TConfigFileData;
```

File

IBServices

Description

This is class IBServices.TConfigFileData.

27.1.1.1 TConfigFileData Members

The following tables list the members exposed by TConfigFileData.

Public Fields

	Name	Description
	ConfigFileKey (see page 3941)	This is ConfigFileKey, a member of class TConfigFileData.
	ConfigFileValue (see page 3941)	This is ConfigFileValue, a member of class TConfigFileData.

Public Methods

	Name	Description
	Create (see page 3940)	This is Create, a member of class TConfigFileData.
	Destroy (see page 3940)	This is Destroy, a member of class TConfigFileData.

27.1.1.2 TConfigFileData.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TConfigFileData();
```

Description

This is Create, a member of class TConfigFileData.

27.1.1.3 TConfigFileData.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConfigFileData();
```

Description

This is Destroy, a member of class TConfigFileData.

27.1.1.4 TConfigFileData Fields

The fields of the TConfigFileData class are listed here.

Public Fields

	Name	Description
	ConfigFileKey (see page 3941)	This is ConfigFileKey, a member of class TConfigFileData.
	ConfigFileValue (see page 3941)	This is ConfigFileValue, a member of class TConfigFileData.

27.1.1.4.1 TConfigFileData.ConfigFileKey Field

Pascal

```
ConfigFileKey: array of integer;
```

C++

```
array of integer ConfigFileKey;
```

Description

This is ConfigFileKey, a member of class TConfigFileData.

27.1.1.4.2 TConfigFileData.ConfigFileValue Field

Pascal

```
ConfigFileValue: array of integer;
```

C++

```
array of integer ConfigFileValue;
```

Description

This is ConfigFileValue, a member of class TConfigFileData.

27.1.2 TConfigParams Class

TConfigFileData (see page 3939) and TConfigParams hold configuration information about an InterBase server

Class Hierarchy

Pascal

```
TConfigParams = class;
```

C++

```
class TConfigParams;
```

File

IBServices

Description

TConfigFileData (see page 3939) stored information about an InterBase configuration file. It is one of the parameters of the TConfigParams type, which stores server configuration settings.

27.1.2.1 TConfigParams Members

The following tables list the members exposed by TConfigParams.

Public Fields

	Name	Description
	BaseLocation (see page 3942)	This is BaseLocation, a member of class TConfigParams.

	ConfigFileData (see page 3943)	This is ConfigFileData, a member of class TConfigParams.
	ConfigFileParams (see page 3943)	This is ConfigFileParams, a member of class TConfigParams.
	LockFileLocation (see page 3943)	This is LockFileLocation, a member of class TConfigParams.
	MessageFileLocation (see page 3943)	This is MessageFileLocation, a member of class TConfigParams.
	SecurityDatabaseLocation (see page 3944)	This is SecurityDatabaseLocation, a member of class TConfigParams.

Public Methods

	Name	Description
	Create (see page 3942)	This is Create, a member of class TConfigParams.
V	Destroy (see page 3942)	This is Destroy, a member of class TConfigParams.

27.1.2.2 TConfigParams.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TConfigParams();
```

Description

This is Create, a member of class TConfigParams.

27.1.2.3 TConfigParams.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConfigParams();
```

Description

This is Destroy, a member of class TConfigParams.

27.1.2.4 TConfigParams Fields

The fields of the TConfigParams class are listed here.

Public Fields

	Name	Description
	BaseLocation (see page 3942)	This is BaseLocation, a member of class TConfigParams.
	ConfigFileData (see page 3943)	This is ConfigFileData, a member of class TConfigParams.
	ConfigFileParams (see page 3943)	This is ConfigFileParams, a member of class TConfigParams.
	LockFileLocation (see page 3943)	This is LockFileLocation, a member of class TConfigParams.
	MessageFileLocation (see page 3943)	This is MessageFileLocation, a member of class TConfigParams.
	SecurityDatabaseLocation (see page 3944)	This is SecurityDatabaseLocation, a member of class TConfigParams.

27.1.2.4.1 TConfigParams.BaseLocation Field

Pascal

```
BaseLocation: string;
```

C++

```
UnicodeString BaseLocation;
```

Description

This is BaseLocation, a member of class TConfigParams.

27.1.2.4.2 TConfigParams.ConfigFileData Field

Pascal

```
ConfigFileData: TConfigFileData;
```

C++

```
TConfigFileData ConfigFileData;
```

Description

This is ConfigFileData, a member of class TConfigParams.

27.1.2.4.3 TConfigParams.ConfigFileParams Field

Pascal

```
ConfigFileParams: array of string;
```

C++

```
array of string ConfigFileParams;
```

Description

This is ConfigFileParams, a member of class TConfigParams.

27.1.2.4.4 TConfigParams.LockFileLocation Field

Pascal

```
LockFileLocation: string;
```

C++

```
UnicodeString LockFileLocation;
```

Description

This is LockFileLocation, a member of class TConfigParams.

27.1.2.4.5 TConfigParams.MessageFileLocation Field

Pascal

```
MessageFileLocation: string;
```

C++

```
UnicodeString MessageFileLocation;
```

Description

This is MessageFileLocation, a member of class TConfigParams.

27.1.2.4.6 TConfigParams.SecurityDatabaseLocation Field

Pascal

```
SecurityDatabaseLocation: string;
```

C++

```
UnicodeString SecurityDatabaseLocation;
```

Description

This is SecurityDatabaseLocation, a member of class TConfigParams.

27.1.3 TDatabaseInfo Class

TDatabaseInfo stores information about an InterBase database.

Class Hierarchy

Pascal

```
TDatabaseInfo = class;
```

C++

```
class TDatabaseInfo;
```

File

IBServices

Description

TDatabaseInfo describes an InterBase database.

NoOfAttachments (see page 3945) is the number attachments.

NoOfDatabases (see page 3946) is the number of databases.

DbName (see page 3945) is the name of the database.

27.1.3.1 TDatabaseInfo Members

The following tables list the members exposed by TDatabaseInfo.

Public Fields

	Name	Description
	DbName (see page 3945)	This is DbName, a member of class TDatabaseInfo.
	NoOfAttachments (see page 3945)	This is NoOfAttachments, a member of class TDatabaseInfo.
	NoOfDatabases (see page 3946)	This is NoOfDatabases, a member of class TDatabaseInfo.

Public Methods

	Name	Description
	Create (see page 3945)	This is Create, a member of class TDatabaseInfo.
	Destroy (see page 3945)	This is Destroy, a member of class TDatabaseInfo.

27.1.3.2 TDatabaseInfo.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDatabaseInfo();
```

Description

This is Create, a member of class TDatabaseInfo.

27.1.3.3 TDatabaseInfo.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDatabaseInfo();
```

Description

This is Destroy, a member of class TDatabaseInfo.

27.1.3.4 TDatabaseInfo Fields

The fields of the TDatabaseInfo class are listed here.

Public Fields

	Name	Description
	DbName (see page 3945)	This is DbName, a member of class TDatabaseInfo.
	NoOfAttachments (see page 3945)	This is NoOfAttachments, a member of class TDatabaseInfo.
	NoOfDatabases (see page 3946)	This is NoOfDatabases, a member of class TDatabaseInfo.

27.1.3.4.1 TDatabaseInfo.DbName Field

Pascal

```
DbName: array of string;
```

C++

```
array of string DbName;
```

Description

This is DbName, a member of class TDatabaseInfo.

27.1.3.4.2 TDatabaseInfo.NoOfAttachments Field

Pascal

```
NoOfAttachments: Integer;
```

C++

```
int NoOfAttachments;
```

Description

This is NoOfAttachments, a member of class TDatabaseInfo.

27.1.3.4.3 TDatabaseInfo.NoOfDatabases Field

Pascal

```
NoOfDatabases: Integer;
```

C++

```
int NoOfDatabases;
```

Description

This is NoOfDatabases, a member of class TDatabaseInfo.

27.1.4 TIBBackupRestoreService Class

TIBBackupRestoreService is the ancestor object from which TIBBackupService (see page 3948) and TIBRestoreService (see page 3979) are derived.

Class Hierarchy

Pascal

```
TIBBackupRestoreService = class(TIBControlAndQueryService);
```

C++

```
class TIBBackupRestoreService : public TIBControlAndQueryService;
```

File

IBServices

Description

TIBBackupRestoreService is the ancestor object from which TIBBackupService (see page 3948) and TIBRestoreService (see page 3979) are derived.

Note: You must install InterBase 6 to use this feature.

27.1.4.1 TIBBackupRestoreService Members

The following tables list the members exposed by TIBBackupRestoreService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.

	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3964)	This is SetServiceStartOptions, a member of class TIBControlService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3959)	Creates an instance of a TIBControlAndQueryService (see page 3957).
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.
	Verbose (see page 3948)	Sets or returns the backup or restore in verbose mode.

27.1.4.2 TIBBackupRestoreService Properties

The properties of the TIBBackupRestoreService class are listed here.

Published Properties

	Name	Description
	Verbose (see page 3948)	Sets or returns the backup or restore in verbose mode.

27.1.4.2.1 TIBBackupRestoreService.Verbose Property

Sets or returns the backup or restore in verbose mode.

Pascal

```
property Verbose: Boolean;
```

C++

```
__property Boolean Verbose;
```

Description

Use Verbose to set or return the backup or restore in verbose mode.

Note: You must install InterBase 6 to use this feature.

27.1.5 TIBBackupService Class

TIBBackupService allows you to back up your database.

Class Hierarchy

Pascal

```
TIBBackupService = class(TIBBackupRestoreService);
```

C++

```
class TIBBackupService : public TIBBackupRestoreService;
```

File

IBServices

Description

Use a TIBBackupService object to back up your database.

See Also

Getting Started with InterBase Express

27.1.5.1 TIBBackupService Members

The following tables list the members exposed by TIBBackupService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.

	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3950)	This is SetServiceStartOptions, a member of class TIBBackupService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3950)	Creates an instance of TIBBackupService (see page 3948)
V	Destroy (see page 3950)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BackupFile (see page 3951)	Sets or returns the backup file name.
	BlockingFactor (see page 3951)	Sets or returns the blocking factor for the tape device.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	DatabaseName (see page 3951)	Sets or returns the database name.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Options (see page 3951)	Sets the options for the database backup.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.
	Verbose (see page 3948)	Sets or returns the backup or restore in verbose mode.

27.1.5.2 TIBBackupService.Create Constructor

Creates an instance of TIBBackupService (see page 3948)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBBackupService (see page 3948).

27.1.5.3 TIBBackupService.Destroy Destructor

Destroys the service.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBBackupService();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

27.1.5.4 TIBBackupService Methods

The methods of the TIBBackupService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 3950)	This is SetServiceStartOptions, a member of class TIBBackupService.

27.1.5.4.1 TIBBackupService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBBackupService.

27.1.5.5 TIBBackupService Properties

The properties of the TIBBackupService class are listed here.

Published Properties

	Name	Description
	BackupFile (see page 3951)	Sets or returns the backup file name.
	BlockingFactor (see page 3951)	Sets or returns the blocking factor for the tape device.
	DatabaseName (see page 3951)	Sets or returns the database name.
	Options (see page 3951)	Sets the options for the database backup.

27.1.5.5.1 TIBBackupService.BackupFile Property

Sets or returns the backup file name.

Pascal

```
property BackupFile: TStrings;
```

C++

```
__property TStrings BackupFile;
```

Description

Use BackupFile to set or return the backup file name.

27.1.5.5.2 TIBBackupService.BlockingFactor Property

Sets or returns the blocking factor for the tape device.

Pascal

```
property BlockingFactor: Integer;
```

C++

```
__property int BlockingFactor;
```

Description

Use BlockingFactor to set the blocking factor for the tape device as an integer.

27.1.5.5.3 TIBBackupService.DatabaseName Property

Sets or returns the database name.

Pascal

```
property DatabaseName: string;
```

C++

```
__property UnicodeString DatabaseName;
```

Description

Use DatabaseName to set or return the database name.

27.1.5.5.4 TIBBackupService.Options Property

Sets the options for the database backup.

Pascal

```
property Options: TBackupOptions;
```

C++

```
__property TBackupOptions Options;
```

Description

Use Options to select the options for the database backup.

27.1.6 TIBConfigService Class

TIBConfigService configures the database parameters

Class Hierarchy

Pascal

```
TIBConfigService = class(TIBControlService);
```

C++

```
class TIBConfigService : public TIBControlService;
```

File

IBServices

Description

Use a TIBConfigService object to configure database parameters.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.6.1 TIBConfigService Members

The following tables list the members exposed by TIBConfigService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3964)	This is SetServiceStartOptions, a member of class TIBControlService.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.

	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	ActivateShadow (see page 3954)	Activates the shadow file for database use.
	Attach (see page 3967)	Attaches to the database.
	BringDatabaseOnline (see page 3954)	Brings the database online.
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
V	ServiceStart (see page 3954)	Not used for TIBConfigService (see page 3952)
	SetAsyncMode (see page 3955)	Sets database writes to asynchronous mode.
	SetDBSqlDialect (see page 3955)	Sets the SQL dialect for the database.
	SetPageBuffers (see page 3955)	Sets the number of database page buffers.
	SetReadOnly (see page 3956)	Sets the database to read only.
	SetReserveSpace (see page 3956)	Sets reserve space on the data page for versioning.
	SetSweepInterval (see page 3956)	Sets the sweep interval.
	ShutdownDatabase (see page 3957)	Shuts down the database.

Public Properties

	Name	Description
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	DatabaseName (see page 3957)	Sets or returns the database name.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.6.2 TIBConfigService Methods

The methods of the TIBConfigService class are listed here.

Public Methods

	Name	Description
	ActivateShadow (see page 3954)	Activates the shadow file for database use.
	BringDatabaseOnline (see page 3954)	Brings the database online.
V	ServiceStart (see page 3954)	Not used for TIBConfigService (see page 3952)
	SetAsyncMode (see page 3955)	Sets database writes to asynchronous mode.
	SetDBSqlDialect (see page 3955)	Sets the SQL dialect for the database.
	SetPageBuffers (see page 3955)	Sets the number of database page buffers.

	SetReadOnly (see page 3956)	Sets the database to read only.
	SetReserveSpace (see page 3956)	Sets reserve space on the data page for versioning.
	SetSweepInterval (see page 3956)	Sets the sweep interval.
	ShutdownDatabase (see page 3957)	Shuts down the database.

27.1.6.2.1 TIBConfigService.ActivateShadow Method

Activates the shadow file for database use.

Pascal

```
procedure ActivateShadow;
```

C++

```
__fastcall ActivateShadow();
```

Description

Call ActivateShadow to activate the database shadow.

InterBase 6 lets you recover a database in case of disk failure, network failure, or accidental deletion of the database. This recovery method is known as disk shadowing, or simply shadowing. Before you can activate shadowing, you must first create a database shadow, as discussed in 'Shadowing' and 'Creating a shadow' in the InterBase 6 Operations Guide.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.2 TIBConfigService.BringDatabaseOnline Method

Brings the database online.

Pascal

```
procedure BringDatabaseOnline;
```

C++

```
__fastcall BringDatabaseOnline();
```

Description

Call BringDatabaseOnline to bring a database online.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.3 TIBConfigService.ServiceStart Method

Not used for TIBConfigService (see page 3952)

Pascal

```
procedure ServiceStart; override;
```

C++

```
virtual __fastcall ServiceStart();
```

Description

Do not use ServiceStart. TIBConfigService (see page 3952) does not currently use this method.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.4 TIBConfigService.SetAsyncMode Method

Sets database writes to asynchronous mode.

Pascal

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

C++

```
__fastcall SetAsyncMode(Boolean Value);
```

Description

Set SetAsyncMode to true to change the database write mode to asynchronous (buffered writes).

InterBase 6 allows you to write to databases in both synchronous and asynchronous modes. In synchronous mode, the database writes are forced. In asynchronous mode, the database writes are buffered.

For more information, refer to 'Forced writes vs. buffered writes' in the InterBase 6 Operations Guide.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.5 TIBConfigService.SetDBSqlDialect Method

Sets the SQL dialect for the database.

Pascal

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

C++

```
__fastcall SetDBSqlDialect(int Value);
```

Description

Call SetDBSqlDialect to set the SQL dialect for the database. Valid integer values are 1, 2, and 3.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.6 TIBConfigService.SetPageBuffers Method

Sets the number of database page buffers.

Pascal

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

C++

```
__fastcall SetPageBuffers(int Value);
```

Description

Call SetPageBuffers to set the number of database page buffers.

When a program establishes a connection to a database, InterBase allocates system memory to use as a private buffer. The buffers are used to store accessed database pages to speed performance. The number of buffers assigned determines how many simultaneous database pages it can have access to in the memory pool. Buffers remain assigned until a program finishes with a database.

For more information on page buffers, refer to 'Setting database cache buffers' in the InterBase 6 Programmer's Guide.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.7 TIBConfigService.SetReadOnly Method

Sets the database to read only.

Pascal

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

C++

```
__fastcall SetReadOnly(Boolean Value);
```

Description

Call SetReadOnly to set the database to read only.

Access mode specifies the type of access a transaction has for the table it uses. There are two possible modes: read-only and read-write.

- Read-only specifies that a transaction can read data from a table, but cannot insert, update, or delete table data.
- Read-write specifies that a transaction can select, insert, update, and delete table data. This is the default setting if none is specified.

Tip: You should specify the transaction's access mode even if is read-write. It makes the application's source code easier to read and debug, because the program's intentions are clearly spelled out.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.8 TIBConfigService.SetReserveSpace Method

Sets reserve space on the data page for versioning.

Pascal

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

C++

```
__fastcall SetReserveSpace(Boolean Value);
```

Description

Set SetReserveSpace to true to reserve space on the data page for versioning.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.9 TIBConfigService.SetSweepInterval Method

Sets the sweep interval.

Pascal

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

C++

```
__fastcall SetSweepInterval(int Value);
```

Description

Call SetSweepInterval set the sweep interval for the database. The sweep interval refers to the number of transactions between database sweeps.

To turn off database sweeps, set the sweep interval to 0.

Sweeping a database is a systematic way of removing outdated records from the database. Periodic sweeping keeps the database from getting too large. However, sweeping can also slow performance. For more information, refer to 'Setting the sweep interval' in the InterBase 6 Operations Guide.

Note: You must install InterBase 6 to use this feature.

27.1.6.2.10 TIBConfigService.ShutdownDatabase Method

Shuts down the database.

Pascal

```
procedure ShutdownDatabase(Options: TShutdownMode; Wait: Integer);
```

C++

```
__fastcall ShutdownDatabase(TShutdownMode Options, int Wait);
```

Description

Call ShutdownDatabase to shut down the database after a specified number of seconds.

Note: You must install InterBase 6 to use this feature.

27.1.6.3 TIBConfigService Properties

The properties of the TIBConfigService class are listed here.

Published Properties

	Name	Description
	DatabaseName (see page 3957)	Sets or returns the database name.

27.1.6.3.1 TIBConfigService.DatabaseName Property

Sets or returns the database name.

Pascal

```
property DatabaseName: string;
```

C++

```
__property UnicodeString DatabaseName;
```

Description

Use DatabaseName to set or return the database name to set properties on.

Note: You must install InterBase 6 to use this feature.

27.1.7 TIBControlAndQueryService Class

TIBControlAndQueryService is the ancestor object from which the log, statistical, validation, security, and backup and restore IBService components descend.

Class Hierarchy

Pascal

```
TIBControlAndQueryService = class(TIBControlService);
```

C++

```
class TIBControlAndQueryService : public TIBControlService;
```

File

IBServices

Description

TIBControlAndQueryService is the ancestor object from which the log, statistical, validation, security, and backup and restore IBService components descend. The service is unstructured because the output is in an unformatted text file.

Note: You must install InterBase 6 to use this feature.

27.1.7.1 TIBControlAndQueryService Members

The following tables list the members exposed by TIBControlAndQueryService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3964)	This is SetServiceStartOptions, a member of class TIBControlService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3959)	Creates an instance of a TIBControlAndQueryService (see page 3957).
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
	Eof (see page 3961)	Indicates that the end of file has been reached.
	Handle (see page 3970)	Returns the database handle.
	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.7.2 TIBControlAndQueryService.Create Constructor

Creates an instance of a TIBControlAndQueryService (see page 3957).

Pascal

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

C++

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

Description

Call Create to create an instance of a TIBControlAndQueryService (see page 3957) component.

Note: You must install InterBase 6 to use this feature.

27.1.7.3 TIBControlAndQueryService Methods

The methods of the TIBControlAndQueryService class are listed here.

Public Methods

	Name	Description
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.

27.1.7.3.1 TIBControlAndQueryService.GetNextChunk Method

Gets the next chunk of data.

Pascal

```
function GetNextChunk: String;
```

C++

```
__fastcall UnicodeString GetNextChunk();
```

Description

Call GetNextChunk to get the next chunk of data.

Note: You must install InterBase 6 to use this feature.

27.1.7.3.2 TIBControlAndQueryService.GetNextLine Method

Starts the service.

Pascal

```
function GetNextLine: String;
```

C++

```
__fastcall UnicodeString GetNextLine();
```

Description

Call GetNextLine to get the next line of data.

Note: You must install InterBase 6 to use this feature.

27.1.7.4 TIBControlAndQueryService Properties

The properties of the TIBControlAndQueryService class are listed here.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.

Published Properties

	Name	Description
	BufferSize (see page 3961)	Sets or returns the buffer size.

27.1.7.4.1 TIBControlAndQueryService.Action Property

Pascal

```
property Action: Integer;
```

C++

```
__property int Action;
```

Description

This is Action, a member of class TIBControlAndQueryService.

27.1.7.4.2 TIBControlAndQueryService.BufferSize Property

Sets or returns the buffer size.

Pascal

```
property BufferSize: Integer;
```

C++

```
__property int BufferSize;
```

Description

Use the BufferSize property to set or return the buffer size.

Note: You must install InterBase 6 to use this feature.

27.1.7.4.3 TIBControlAndQueryService.Eof Property

Indicates that the end of file has been reached.

Pascal

```
property Eof: boolean;
```

C++

```
__property boolean Eof;
```

Description

Use the Eof property to determine whether the end of the file has been reached.

Note: You must install InterBase 6 to use this feature.

27.1.8 TIBControlService Class

TIBControlService is the ancestor object from which the structured and unstructured IBService components descend.

Class Hierarchy

Pascal

```
TIBControlService = class(TIBCustomService);
```

C++

```
class TIBControlService : public TIBCustomService;
```

File

IBServices

Description

TIBControlService is the ancestor object from which the structured and unstructured IBService components descend.

Note: You must install InterBase 6 to use this feature.

27.1.8.1 TIBControlService Members

The following tables list the members exposed by TIBControlService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3964)	This is SetServiceStartOptions, a member of class TIBControlService.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.8.2 TIBControlService.Create Constructor

Creates an instance of TIBControlService (see page 3961)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBControlService (see page 3961).

Note: You must install InterBase 6 to use this feature.

27.1.8.3 TIBControlService Methods

The methods of the TIBControlService class are listed here.

Protected Methods

	Name	Description
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3964)	This is SetServiceStartOptions, a member of class TIBControlService.

Public Methods

	Name	Description
V	ServiceStart (see page 3963)	Starts the service.

27.1.8.3.1 TIBControlService.InternalServiceStart Method

Pascal

```
procedure InternalServiceStart;
```

C++

```
__fastcall InternalServiceStart();
```

Description

This is InternalServiceStart, a member of class TIBControlService.

27.1.8.3.2 TIBControlService.ServiceStart Method

Starts the service.

Pascal

```
procedure ServiceStart; virtual;
```

C++

```
virtual __fastcall ServiceStart();
```

Description

Call ServiceStart to start the service.

Note: You must install InterBase 6 to use this feature.

27.1.8.3.3 ServiceStartAddParam Method

This is the overview for the ServiceStartAddParam method overload.

Overload List

	Name	Description
	TIBControlService.ServiceStartAddParam (Integer, Integer) (see page 3964)	This is ServiceStartAddParam, a member of class TIBControlService.
	TIBControlService.ServiceStartAddParam (string, Integer) (see page 3964)	This is ServiceStartAddParam, a member of class TIBControlService.

27.1.8.3.3.1 TIBControlService.ServiceStartAddParam Method (Integer, Integer)

Pascal

```
procedure ServiceStartAddParam(Value: Integer; param: Integer); overload;
```

C++

```
__fastcall ServiceStartAddParam(int Value, int param);
```

Description

This is ServiceStartAddParam, a member of class TIBControlService.

27.1.8.3.3.2 TIBControlService.ServiceStartAddParam Method (string, Integer)

Pascal

```
procedure ServiceStartAddParam(Value: string; param: Integer); overload;
```

C++

```
__fastcall ServiceStartAddParam(UnicodeString Value, int param);
```

Description

This is ServiceStartAddParam, a member of class TIBControlService.

27.1.8.3.4 TIBControlService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; virtual;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBControlService.

27.1.8.4 TIBControlService Properties

The properties of the TIBControlService class are listed here.

Protected Properties

	Name	Description
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Properties

	Name	Description
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.

27.1.8.4.1 TIBControlService.IsServiceRunning Property

Indicates whether the service is running.

Pascal

```
property IsServiceRunning: Boolean;
```

C++

```
__property Boolean IsServiceRunning;
```

Description

Use the IsServiceRunning property to check whether the service is running.

Note: You must install InterBase 6 to use this feature.

27.1.8.4.2 TIBControlService.ServiceStartParams Property

Pascal

```
property ServiceStartParams: String;
```

C++

```
__property UnicodeString ServiceStartParams;
```

Description

This is ServiceStartParams, a member of class TIBControlService.

27.1.9 TIBCustomService Class

TIBCustomService is the ancestor object from which all IBService components descend.

Class Hierarchy

Pascal

```
TIBCustomService = class(TComponent);
```

C++

```
class TIBCustomService : public TComponent;
```

File

IBServices

Description

TIBCustomService is the ancestor object from which all IBService components descend.

27.1.9.1 TIBCustomService Members

The following tables list the members exposed by TIBCustomService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.

Public Properties

	Name	Description
R	Handle (see page 3970)	Returns the database handle.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.9.2 TIBCustomService.Create Constructor

Creates an instance of TIBCustomService (see page 3965)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBCustomService (see page 3965).

27.1.9.3 TIBCustomService.Destroy Destructor

Destroys the service.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBCustomService();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

27.1.9.4 TIBCustomService Methods

The methods of the TIBCustomService class are listed here.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
	Detach (see page 3968)	Detaches from the database.

27.1.9.4.1 TIBCustomService.Attach Method

Attaches to the database.

Pascal

```
procedure Attach;
```

C++

```
__fastcall Attach();
```

Description

Call Attach to attach to the database.

27.1.9.4.2 TIBCustomService.CheckActive Method

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

This is CheckActive, a member of class TIBCustomService.

27.1.9.4.3 TIBCustomService.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TIBCustomService.

27.1.9.4.4 TIBCustomService.Detach Method

Detaches from the database.

Pascal

```
procedure Detach;
```

C++

```
__fastcall Detach();
```

Description

Call Detach to detach from the database.

27.1.9.4.5 TIBCustomService.InternalServiceQuery Method

Pascal

```
procedure InternalServiceQuery;
```

C++

```
__fastcall InternalServiceQuery();
```

Description

This is InternalServiceQuery, a member of class TIBCustomService.

27.1.9.4.6 TIBCustomService.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

27.1.9.4.7 TIBCustomService.Login Method

Pascal

```
function Login: Boolean;
```

C++

```
__fastcall Boolean Login();
```

Description

This is Login, a member of class TIBCustomService.

27.1.9.5 TIBCustomService Properties

The properties of the TIBCustomService class are listed here.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.

Public Properties

	Name	Description
	Handle (see page 3970)	Returns the database handle.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.9.5.1 TIBCustomService.Active Property

Sets the active state of the service.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use the Active property to set the service to active (true) or the default, inactive (false).

27.1.9.5.2 TIBCustomService.BufferSize Property

Pascal

```
property BufferSize: Integer;
```

C++

```
__property int BufferSize;
```

Description

This is BufferSize, a member of class TIBCustomService.

27.1.9.5.3 TIBCustomService.Handle Property

Returns the database handle.

Pascal

```
property Handle: TISC_SVC_HANDLE;
```

C++

```
__property TISC_SVC_HANDLE Handle;
```

Description

Use the Handle property to return the database handle.

27.1.9.5.4 TIBCustomService.LoginPrompt Property

Turns on and off login prompt display.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Use the LoginPrompt**property to display (or not display) a login prompt before attaching to a database.**

27.1.9.5.5 TIBCustomService.OutputBuffer Property

Pascal

```
property OutputBuffer: PChar;
```

C++

```
__property const char * OutputBuffer;
```

Description

This is OutputBuffer, a member of class TIBCustomService.

27.1.9.5.6 TIBCustomService.OutputBufferOption Property

Pascal

```
property OutputBufferOption: TOutputBufferOption;
```

C++

```
__property TOutputBufferOption OutputBufferOption;
```

Description

This is OutputBufferOption, a member of class TIBCustomService.

27.1.9.5.7 TIBCustomService.Params Property

Sets or returns the database parameters

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Use the Params property to set or return database parameters.

27.1.9.5.8 TIBCustomService.Protocol Property

Sets or returns the network protocol.

Pascal

```
property Protocol: TProtocol;
```

C++

```
__property TProtocol Protocol;
```

Description

Use Protocol to select the network protocol.

27.1.9.5.9 TIBCustomService.ServerName Property

Sets the name of the server on which the services are to be run.

Pascal

```
property ServerName: string;
```

C++

```
__property UnicodeString ServerName;
```

Description

Use ServerName to set the name of the server on which the services are to be run.

27.1.9.5.10 TIBCustomService.ServiceParamBySPB Property

Returns and sets SPB parameters.

Pascal

```
property ServiceParamBySPB [const Idx: Integer]: String;
```

C++

```
__property UnicodeString ServiceParamBySPB[int const Idx];
```

Description

Use ParamBySPB to inspect and set SPB parameters.

For example DBParamBySPB[isc_SPB_user_name] sets and inspects the user name.

27.1.9.5.11 TIBCustomService.ServiceQueryParams Property

Pascal

```
property ServiceQueryParams: String;
```

C++

```
__property UnicodeString ServiceQueryParams;
```

Description

This is ServiceQueryParams, a member of class TIBCustomService.

27.1.9.5.12 TIBCustomService.TraceFlags Property

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

This is TraceFlags, a member of class TIBCustomService.

27.1.9.6 TIBCustomService Events

The events of the TIBCustomService class are listed here.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

27.1.9.6.1 TIBCustomService.OnAttach Event

Occurs when the database is attached.

Pascal

```
property OnAttach: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAttach;
```

Description

Write an OnAttach event handler to take specific actions when a database is attached. If an exception is raised in this event, the database is not attached.

27.1.9.6.2 TIBCustomService.OnLogin Event

Occurs during database login.

Pascal

```
property OnLogin: TLoginEvent;
```

C++

```
__property TLoginEvent OnLogin;
```

Description

Write an OnLogin event handler to take specific actions on database login.

27.1.10 TIBLicensingService Class

TIBLicensingService configures the licensing parameters

Class Hierarchy

Pascal

```
TIBLicensingService = class(TIBControlService);
```

C++

```
class TIBLicensingService : public TIBControlService;
```

File

IBServices

Description

Use a TIBLicensingService component to add or remove InterBase software activation certificates.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.10.1 TIBLicensingService Members

The following tables list the members exposed by TIBLicensingService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3976)	This is SetServiceStartOptions, a member of class TIBLicensingService.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	AddLicense (see page 3975)	Adds an InterBase software activation certificate.
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	RemoveLicense (see page 3976)	Removes an InterBase software activation certificate.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
	Handle (see page 3970)	Returns the database handle.
	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Action (see page 3976)	Indictes the licensing action.
	Active (see page 3970)	Sets the active state of the service.
	ID (see page 3977)	Sets or returns the license identification.
	Key (see page 3977)	Sets or returns the license key.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.10.2 TIBLicensingService Methods

The methods of the TIBLicensingService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 3976)	This is SetServiceStartOptions, a member of class TIBLicensingService.

Public Methods

	Name	Description
	AddLicense (see page 3975)	Adds an InterBase software activation certificate.
	RemoveLicense (see page 3976)	Removes an InterBase software activation certificate.

27.1.10.2.1 TIBLicensingService.AddLicense Method

Adds an InterBase software activation certificate.

Pascal

```
procedure AddLicense;
```

C++

```
__fastcall AddLicense();
```

Description

Call AddLicense along to add an InterBase software activation certificate. The Key (see page 3977) and ID (see page 3977) properties must be supplied.

Note: You must install InterBase 6 to use this feature.

27.1.10.2.2 TIBLicensingService.RemoveLicense Method

Removes an InterBase software activation certificate.

Pascal

```
procedure RemoveLicense;
```

C++

```
__fastcall RemoveLicense();
```

Description

Call RemoveLicense to remove an InterBase software activation certificate.

Note: You must install InterBase 6 to use this feature.

27.1.10.2.3 TIBLicensingService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBLicensingService.

27.1.10.3 TIBLicensingService Properties

The properties of the TIBLicensingService class are listed here.

Published Properties

	Name	Description
	Action (see page 3976)	Indictes the licensing action.
	ID (see page 3977)	Sets or returns the license identification.
	Key (see page 3977)	Sets or returns the license key.

27.1.10.3.1 TIBLicensingService.Action Property

Indictes the licensing action.

Pascal

```
property Action: TLicensingAction;
```

C++

```
__property TLicensingAction Action;
```

Description

Use Action to indicate what action the licensing service should take. This causes the service to call the AddLicense (see page 3975) or RemoveLicense (see page 3976) method, depending on whether the software activation certificate is added or removed.

Note: You must install InterBase 6 to use this feature.

27.1.10.3.2 TIBLicensingService.ID Property

Sets or returns the license identification.

Pascal

```
property ID: String;
```

C++

```
__property UnicodeString ID;
```

Description

Use ID to set or return the license identification.

Note: You must install InterBase 6 to use this feature.

27.1.10.3.3 TIBLicensingService.Key Property

Sets or returns the license key.

Pascal

```
property Key: String;
```

C++

```
__property UnicodeString Key;
```

Description

Use Key to set or return the license key.

Note: You must install InterBase 6 to use this feature.

27.1.11 TIBLogService Class

TIBLogService returns the contents of the interbase.log file from a server.

Class Hierarchy

Pascal

```
TIBLogService = class(TIBControlAndQueryService);
```

C++

```
class TIBLogService : public TIBControlAndQueryService;
```

File

IBServices

Description

Use a TIBLogService object to return the contents of the interbase.log file from a server.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.11.1 TIBLogService Members

The following tables list the members exposed by TIBLogService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3979)	This is SetServiceStartOptions, a member of class TIBLogService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3959)	Creates an instance of a TIBControlAndQueryService (see page 3957).
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.11.2 TIBLogService Methods

The methods of the TIBLogService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 3979)	This is SetServiceStartOptions, a member of class TIBLogService.

27.1.11.2.1 TIBLogService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBLogService.

27.1.12 TIBRestoreService Class

TIBRestoreService allows you to restore your database.

Class Hierarchy

Pascal

```
TIBRestoreService = class(TIBBackupRestoreService);
```

C++

```
class TIBRestoreService : public TIBBackupRestoreService;
```

File

IBServices

Description

Use a TIBRestoreService object to restore your database.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.12.1 TIBRestoreService Members

The following tables list the members exposed by TIBRestoreService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3982)	This is SetServiceStartOptions, a member of class TIBRestoreService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3981)	Creates an instance of TIBRestoreService (see page 3979)
V	Destroy (see page 3981)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BackupFile (see page 3982)	Sets or returns the backup file name.

	BufferSize (see page 3961)	Sets or returns the buffer size.
	DatabaseName (see page 3982)	Sets or returns the database name.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Options (see page 3983)	Sets the options for the database restore.
	PageBuffers (see page 3983)	Sets or returns page buffer size.
	PageSize (see page 3983)	Sets or returns the page size.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.
	Verbose (see page 3948)	Sets or returns the backup or restore in verbose mode.

27.1.12.2 TIBRestoreService.Create Constructor

Creates an instance of TIBRestoreService (see page 3979)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBRestoreService (see page 3979).

Note: You must install InterBase 6 to use this feature.

27.1.12.3 TIBRestoreService.Destroy Destructor

Destroys the service.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBRestoreService();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Note: You must install InterBase 6 to use this feature.

27.1.12.4 TIBRestoreService Methods

The methods of the TIBRestoreService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 3982)	This is SetServiceStartOptions, a member of class TIBRestoreService.

27.1.12.4.1 TIBRestoreService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBRestoreService.

27.1.12.5 TIBRestoreService Properties

The properties of the TIBRestoreService class are listed here.

Published Properties

	Name	Description
	BackupFile (see page 3982)	Sets or returns the backup file name.
	DatabaseName (see page 3982)	Sets or returns the database name.
	Options (see page 3983)	Sets the options for the database restore.
	PageBuffers (see page 3983)	Sets or returns page buffer size.
	PageSize (see page 3983)	Sets or returns the page size.

27.1.12.5.1 TIBRestoreService.BackupFile Property

Sets or returns the backup file name.

Pascal

```
property BackupFile: TStrings;
```

C++

```
__property TStrings BackupFile;
```

Description

Use BackupFile to set or return the backup file name.

Note: You must install InterBase 6 to use this feature.

27.1.12.5.2 TIBRestoreService.DatabaseName Property

Sets or returns the database name.

Pascal

```
property DatabaseName: TStrings;
```

C++

```
__property TStrings DatabaseName;
```

Description

Use DatabaseName to set or return the database name.

Note: You must install InterBase 6 to use this feature.

27.1.12.5.3 TIBRestoreService.Options Property

Sets the options for the database restore.

Pascal

```
property Options: TRestoreOptions;
```

C++

```
__property TRestoreOptions Options;
```

Description

Use Options to select the options for the database backup.

Note: You must install InterBase 6 to use this feature.

27.1.12.5.4 TIBRestoreService.PageBuffers Property

Sets or returns page buffer size.

Pascal

```
property PageBuffers: Integer;
```

C++

```
__property int PageBuffers;
```

Description

Use PageBuffers to set or return the page buffer size in kilobytes.

Note: You must install InterBase 6 to use this feature.

27.1.12.5.5 TIBRestoreService.PageSize Property

Sets or returns the page size.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

Use PageSize to set or return the page size in kilobytes.

Note: You must install InterBase 6 to use this feature.

27.1.13 TIBSecurityService Class

TIBSecurityService enables you to manage user access to the InterBase server.

Class Hierarchy

Pascal

```
TIBSecurityService = class(TIBControlAndQueryService);
```

C++

```
class TIBSecurityService : public TIBControlAndQueryService;
```

File

IBServices

Description

Use a TIBSecurityService object to manage user access to the InterBase server.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.13.1 TIBSecurityService Members

The following tables list the members exposed by TIBSecurityService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3987)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 3988)	This is SetServiceStartOptions, a member of class TIBSecurityService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	AddUser (see page 3986)	Adds a user to the user table.
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3985)	Creates an instance of TIBSecurityService (see page 3983).
	DeleteUser (see page 3986)	Deletes a user from the user table.
V	Destroy (see page 3986)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	DisplayUser (see page 3987)	Fetches user information.
	DisplayUsers (see page 3987)	Fetches information for all users.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.

	ModifyUser (see page 3988)	Modifies user information.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.
R	UserInfo (see page 3991)	Provides information about a user.
R	UserInfoCount (see page 3991)	Returns the number of elements in the TUserInfo (see page 4013) array.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	FirstName (see page 3989)	Sets or returns the user's first name.
	GroupID (see page 3989)	Sets or returns the user's group ID number.
	LastName (see page 3989)	Sets or returns the user's last name.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	MiddleName (see page 3990)	Sets or returns the user's middle name.
	Params (see page 3971)	Sets or returns the database parameters
	Password (see page 3990)	Sets the user's password.
	Protocol (see page 3971)	Sets or returns the network protocol.
	SecurityAction (see page 3990)	Specifies the security action.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	SQlRole (see page 3991)	This is SQlRole, a member of class TIBSecurityService.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.
	UserID (see page 3991)	Sets or returns the user's ID number.
	UserName (see page 3992)	Sets or returns the user's login name.

27.1.13.2 TIBSecurityService.Create Constructor

Creates an instance of TIBSecurityService (see page 3983).

Pascal

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

C++

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

Description

Call Create to create an instance of TIBSecurityService (see page 3983).

Note: You must install InterBase 6 to use this feature.

27.1.13.3 TIBSecurityService.Destroy Destructor

Destroys the service.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBSecurityService();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Note: You must install InterBase 6 to use this feature.

27.1.13.4 TIBSecurityService Methods

The methods of the TIBSecurityService class are listed here.

Protected Methods

	Name	Description
	Loaded (see page 3987)	Initializes the component after the form file has been read into memory.
	SetServiceStartOptions (see page 3988)	This is SetServiceStartOptions, a member of class TIBSecurityService.

Public Methods

	Name	Description
	AddUser (see page 3986)	Adds a user to the user table.
	DeleteUser (see page 3986)	Deletes a user from the user table.
	DisplayUser (see page 3987)	Fetches user information.
	DisplayUsers (see page 3987)	Fetches information for all users.
	ModifyUser (see page 3988)	Modifies user information.

27.1.13.4.1 TIBSecurityService.AddUser Method

Adds a user to the user table.

Pascal

```
procedure AddUser;
```

C++

```
__fastcall AddUser();
```

Description

Call AddUser to add a user to the user table.

Note: You must install InterBase 6 to use this feature.

27.1.13.4.2 TIBSecurityService.DeleteUser Method

Deletes a user from the user table.

Pascal

```
procedure DeleteUser;
```

C++

```
__fastcall DeleteUser();
```

Description

Call DeleteUser to delete a user from the user table.

Note: You must install InterBase 6 to use this feature.

27.1.13.4.3 TIBSecurityService.DisplayUser Method

Fetches user information.

Pascal

```
procedure DisplayUser(UserName: string);
```

C++

```
__fastcall DisplayUser(UnicodeString UserName);
```

Description

Call DisplayUser to fetch user information and dump it into the TUserInfo (see page 4013) record, which can then be returned by the UserInfo (see page 3991) property.

Note: You must install InterBase 6 to use this feature.

27.1.13.4.4 TIBSecurityService.DisplayUsers Method

Fetches information for all users.

Pascal

```
procedure DisplayUsers;
```

C++

```
__fastcall DisplayUsers();
```

Description

Call DisplayUsers to fetch all valid user information, which can then be returned by the UserInfo (see page 3991) property.

Note: You must install InterBase 6 to use this feature.

27.1.13.4.5 TIBSecurityService.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

27.1.13.4.6 TIBSecurityService.ModifyUser Method

Modifies user information.

Pascal

```
procedure ModifyUser;
```

C++

```
__fastcall ModifyUser();
```

Description

Call ModifyUser to modify a user's information.

Note: You must install InterBase 6 to use this feature.

27.1.13.4.7 TIBSecurityService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBSecurityService.

27.1.13.5 TIBSecurityService Properties

The properties of the TIBSecurityService class are listed here.

Public Properties

	Name	Description
R	UserInfo (see page 3991)	Provides information about a user.
R	UserInfoCount (see page 3991)	Returns the number of elements in the TUserInfo (see page 4013) array.

Published Properties

	Name	Description
	FirstName (see page 3989)	Sets or returns the user's first name.
	GroupID (see page 3989)	Sets or returns the user's group ID number.
	LastName (see page 3989)	Sets or returns the user's last name.
	MiddleName (see page 3990)	Sets or returns the user's middle name.
	Password (see page 3990)	Sets the user's password.
	SecurityAction (see page 3990)	Specifies the security action.
	SQlRole (see page 3991)	This is SQlRole, a member of class TIBSecurityService.
	UserID (see page 3991)	Sets or returns the user's ID number.
	UserName (see page 3992)	Sets or returns the user's login name.

27.1.13.5.1 TIBSecurityService.FirstName Property

Sets or returns the user's first name.

Pascal

```
property FirstName: string;
```

C++

```
__property UnicodeString FirstName;
```

Description

Use FirstName to set or return the user's first name.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.2 TIBSecurityService.GroupID Property

Sets or returns the user's group ID number.

Pascal

```
property GroupID: Integer;
```

C++

```
__property int GroupID;
```

Description

Use GroupID to set or return the user's group identification number (GID).

Note: You must install InterBase 6 to use this feature.

27.1.13.5.3 TIBSecurityService.LastName Property

Sets or returns the user's last name.

Pascal

```
property LastName: string;
```

C++

```
__property UnicodeString LastName;
```

Description

Use LastName to set or return the user's last name.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.4 TIBSecurityService.MiddleName Property

Sets or returns the user's middle name.

Pascal

```
property MiddleName: string;
```

C++

```
__property UnicodeString MiddleName;
```

Description

Use MiddleName to set or return user's middle name.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.5 TIBSecurityService.Password Property

Sets the user's password.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

Use Password to indicate the user's password when adding or modifying a user's information.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.6 TIBSecurityService.SecurityAction Property

Specifies the security action.

Pascal

```
property SecurityAction: TSecurityAction;
```

C++

```
__property TSecurityAction SecurityAction;
```

Description

Use SecurityAction to indicate what operation the security server should perform.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.7 TIBSecurityService.SQlRole Property

Pascal

```
property SQlRole: string;
```

C++

```
__property UnicodeString SQlRole;
```

Description

This is SQlRole, a member of class TIBSecurityService.

27.1.13.5.8 TIBSecurityService.UserID Property

Sets or returns the user's ID number.

Pascal

```
property UserID: Integer;
```

C++

```
__property int UserID;
```

Description

Use UserID to set or return the user's identification number (UID).

Note: You must install InterBase 6 to use this feature.

27.1.13.5.9 TIBSecurityService.UserInfo Property

Provides information about a user.

Pascal

```
property UserInfo [Index: Integer]: TUserInfo;
```

C++

```
__property TUserInfo UserInfo[int Index];
```

Description

Use UserInfo to return the user's information as a TUserInfo (see page 4013) record, which contains the user name, groupID, UserID (see page 3991), and the user's first, middle, and last names.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.10 TIBSecurityService.UserInfoCount Property

Returns the number of elements in the TUserInfo (see page 4013) array.

Pascal

```
property UserInfoCount: Integer;
```

C++

```
__property int UserInfoCount;
```

Description

Use UserInfoCount to return the number of elements in the TUserInfo (see page 4013) array.

Note: You must install InterBase 6 to use this feature.

27.1.13.5.11 TIBSecurityService.UserName Property

Sets or returns the user's login name.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

Use UserName to set or return the user's login (user) name.

Note: You must install InterBase 6 to use this feature.

27.1.14 TIBServerProperties Class

TIBServerProperties returns database server information.

Class Hierarchy

Pascal

```
TIBServerProperties = class(TIBCustomService);
```

C++

```
class TIBServerProperties : public TIBCustomService;
```

File

IBServices

Description

TIBServerProperties returns database server information, including configuration parameters, and version and license information.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.14.1 TIBServerProperties Members

The following tables list the members exposed by TIBServerProperties.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.

	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.

Protected Properties

	Name	Description
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3993)	Creates an instance of TIBServerProperties (see page 3992)
V	Destroy (see page 3994)	Destroys the instance of TIBServerProperties (see page 3992).
	Detach (see page 3968)	Detaches from the database.
	Fetch (see page 3994)	Fetches information from the server.
	FetchConfigParams (see page 3995)	Returns the database configuration parameters.
	FetchDatabaseInfo (see page 3995)	Returns database information.
	FetchLicenseInfo (see page 3995)	Returns license information.
	FetchLicenseMaskInfo (see page 3995)	Returns license mask information.
	FetchVersionInfo (see page 3996)	Returns the database version information.

Public Properties

	Name	Description
R	ConfigParams (see page 3996)	Returns the server configuration parameters.
R	DatabaseInfo (see page 3997)	Indicates database information.
R	Handle (see page 3970)	Returns the database handle.
R	LicenseInfo (see page 3997)	Indicates license information.
R	LicenseMaskInfo (see page 3997)	Returns license mask information.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.
R	VersionInfo (see page 3998)	Indicates version information.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Options (see page 3997)	Indicates the server properties.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.14.2 TIBServerProperties.Create Constructor

Creates an instance of TIBServerProperties (see page 3992)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBServerProperties (see page 3992).

Note: You must install InterBase 6 to use this feature.

27.1.14.3 TIBServerProperties.Destroy Destructor

Destroys the instance of TIBServerProperties (see page 3992).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBServerProperties();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Note: You must install InterBase 6 to use this feature.

27.1.14.4 TIBServerProperties Methods

The methods of the TIBServerProperties class are listed here.

Public Methods

	Name	Description
	Fetch (see page 3994)	Fetches information from the server.
	FetchConfigParams (see page 3995)	Returns the database configuration parameters.
	FetchDatabaseInfo (see page 3995)	Returns database information.
	FetchLicenseInfo (see page 3995)	Returns license information.
	FetchLicenseMaskInfo (see page 3995)	Returns license mask information.
	FetchVersionInfo (see page 3996)	Returns the database version information.

27.1.14.4.1 TIBServerProperties.Fetch Method

Fetches information from the server.

Pascal

```
procedure Fetch;
```

C++

```
__fastcall Fetch();
```

Description

Call Fetch to get information from the server.

Note: You must install InterBase 6 to use this feature.

27.1.14.4.2 TIBServerProperties.FetchConfigParams Method

Returns the database configuration parameters.

Pascal

```
procedure FetchConfigParams;
```

C++

```
__fastcall FetchConfigParams();
```

Description

Call FetchConfigParams to return database configuration parameters.

Note: You must install InterBase 6 to use this feature.

27.1.14.4.3 TIBServerProperties.FetchDatabaseInfo Method

Returns database information.

Pascal

```
procedure FetchDatabaseInfo;
```

C++

```
__fastcall FetchDatabaseInfo();
```

Description

Call FetchDatabaseInfo to return database information.

Note: You must install InterBase 6 to use this feature.

27.1.14.4.4 TIBServerProperties.FetchLicenseInfo Method

Returns license information.

Pascal

```
procedure FetchLicenseInfo;
```

C++

```
__fastcall FetchLicenseInfo();
```

Description

Call FetchLicenseInfo to return database license information.

Note: You must install InterBase 6 to use this feature.

27.1.14.4.5 TIBServerProperties.FetchLicenseMaskInfo Method

Returns license mask information.

Pascal

```
procedure FetchLicenseMaskInfo;
```

C++

```
__fastcall FetchLicenseMaskInfo();
```

Description

Call FetchLicenseMaskInfo to return database license mask information.

Note: You must install InterBase 6 to use this feature.

27.1.14.4.6 TIBServerProperties.FetchVersionInfo Method

Returns the database version information.

Pascal

```
procedure FetchVersionInfo;
```

C++

```
__fastcall FetchVersionInfo();
```

Description

Call FetchVersionInfo to return the database version information.

Note: You must install InterBase 6 to use this feature.

27.1.14.5 TIBServerProperties Properties

The properties of the TIBServerProperties class are listed here.

Public Properties

	Name	Description
	ConfigParams (see page 3996)	Returns the server configuration parameters.
	DatabaseInfo (see page 3997)	Indicates database information.
	LicenseInfo (see page 3997)	Indicates license information.
	LicenseMaskInfo (see page 3997)	Returns license mask information.
	VersionInfo (see page 3998)	Indicates version information.

Published Properties

	Name	Description
	Options (see page 3997)	Indicates the server properties.

27.1.14.5.1 TIBServerProperties.ConfigParams Property

Returns the server configuration parameters.

Pascal

```
property ConfigParams: TConfigParams;
```

C++

```
__property TConfigParams ConfigParams;
```

Description

Use the ConfigParams property to return the server configuration parameters as a TConfigParams (see page 3941) record, which includes the configuration file data, base location, lock file location, message file location, and the security database location.

Note: You must install InterBase 6 to use this feature.

27.1.14.5.2 TIBServerProperties.DatabaseInfo Property

Indicates database information.

Pascal

```
property DatabaseInfo: TDatabaseInfo;
```

C++

```
__property TDatabaseInfo DatabaseInfo;
```

Description

Use the DatabaseInfo property to obtain database information, which includes the database name, number of attachments, and number of databases.

Note: You must install InterBase 6 to use this feature.

27.1.14.5.3 TIBServerProperties.LicenseInfo Property

Indicates license information.

Pascal

```
property LicenseInfo: TLicenseInfo;
```

C++

```
__property TLicenseInfo LicenseInfo;
```

Description

Use the LicenseInfo property to return licensing information as a TLicenseInfo (see page 4006) record, which includes the key, ID, and number of licensed users.

Note: You must install InterBase 6 to use this feature.

27.1.14.5.4 TIBServerProperties.LicenseMaskInfo Property

Returns license mask information.

Pascal

```
property LicenseMaskInfo: TLicenseMaskInfo;
```

C++

```
__property TLicenseMaskInfo LicenseMaskInfo;
```

Description

Use the LicenseMaskInfo property to return licensing information as a TLicenseMaskInfo (see page 4008) record, which includes the license mask and capability mask. A license mask is a bitmask representing the software activation certificate options currently enabled on the server. A capability mask is a bitmask representing the capabilities currently enabled on the server.

Note: You must install InterBase 6 to use this feature.

27.1.14.5.5 TIBServerProperties.Options Property

Indicates the server properties.

Pascal

```
property Options: TPropertyOptions;
```

C++

```
__property TPropertyOptions Options;
```

Description

Use the Options property to set and return server properties.

Note: You must install InterBase 6 to use this feature.

27.1.14.5.6 TIBServerProperties.VersionInfo Property

Indicates version information.

Pascal

```
property VersionInfo: TVersionInfo;
```

C++

```
__property TVersionInfo VersionInfo;
```

Description

Use the VersionInfo property to obtain the server configuration parameters as a TVersionInfo (see page 4015) type, which includes the service version, and server version and implementation.

Note: You must install InterBase 6 to use this feature.

27.1.15 TIBStatisticalService Class

TIBStatisticalService allows you to view database statistics.

Class Hierarchy

Pascal

```
TIBStatisticalService = class(TIBControlAndQueryService);
```

C++

```
class TIBStatisticalService : public TIBControlAndQueryService;
```

File

IBServices

Description

Use a TIBStatisticalService object to view database statistics.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.15.1 TIBStatisticalService Members

The following tables list the members exposed by TIBStatisticalService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.
V	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
V	SetServiceStartOptions (see page 4000)	This is SetServiceStartOptions, a member of class TIBStatisticalService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
R	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
V	Create (see page 3959)	Creates an instance of a TIBControlAndQueryService (see page 3957).
V	Create (see page 3963)	Creates an instance of TIBControlService (see page 3961)
V	Create (see page 3966)	Creates an instance of TIBCustomService (see page 3965)
V	Destroy (see page 3967)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
V	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
R	Eof (see page 3961)	Indicates that the end of file has been reached.
R	Handle (see page 3970)	Returns the database handle.
R	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	DatabaseName (see page 4000)	Sets or returns the database name.

	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Options (see page 4001)	Indicates the desired statistical information.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.15.2 TIBStatisticalService Methods

The methods of the TIBStatisticalService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 4000)	This is SetServiceStartOptions, a member of class TIBStatisticalService.

27.1.15.2.1 TIBStatisticalService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBStatisticalService.

27.1.15.3 TIBStatisticalService Properties

The properties of the TIBStatisticalService class are listed here.

Published Properties

	Name	Description
	DatabaseName (see page 4000)	Sets or returns the database name.
	Options (see page 4001)	Indicates the desired statistical information.

27.1.15.3.1 TIBStatisticalService.DatabaseName Property

Sets or returns the database name.

Pascal

```
property DatabaseName: string;
```

C++

```
__property UnicodeString DatabaseName;
```

Description

Use DatabaseName to set or return the database name.

Note: You must install InterBase 6 to use this feature.

27.1.15.3.2 TIBStatisticalService.Options Property

Indicates the desired statistical information.

Pascal

```
property Options: TStatOptions;
```

C++

```
__property TStatOptions Options;
```

Description

Use the Options property to request database statistics. These options are incremental; that is, including DbLog also returns HeaderPages statistics, including IndexPages also returns DbLog and HeaderPages statistics, and so forth.

Note: You must install InterBase 6 to use this feature.

27.1.16 TIBValidationService Class

TIBValidationService enables you to validate your database and reconcile your database transactions.

Class Hierarchy

Pascal

```
TIBValidationService = class(TIBControlAndQueryService);
```

C++

```
class TIBValidationService : public TIBControlAndQueryService;
```

File

IBServices

Description

Use a TIBValidationService object to validate your database and reconcile your database transactions.

Note: You must install InterBase 6 to use this feature.

See Also

Getting Started with InterBase Express

27.1.16.1 TIBValidationService Members

The following tables list the members exposed by TIBValidationService.

Protected Methods

	Name	Description
	CheckActive (see page 3968)	This is CheckActive, a member of class TIBCustomService.
	CheckInactive (see page 3968)	This is CheckInactive, a member of class TIBCustomService.
	InternalServiceQuery (see page 3968)	This is InternalServiceQuery, a member of class TIBCustomService.
	InternalServiceStart (see page 3963)	This is InternalServiceStart, a member of class TIBControlService.

	Loaded (see page 3968)	Initializes the component after the form file has been read into memory.
	Login (see page 3969)	This is Login, a member of class TIBCustomService.
	ServiceStartAddParam (see page 3964)	This is the overview for the ServiceStartAddParam method overload.
	SetServiceStartOptions (see page 4004)	This is SetServiceStartOptions, a member of class TIBValidationService.

Protected Properties

	Name	Description
	Action (see page 3960)	This is Action, a member of class TIBControlAndQueryService.
	BufferSize (see page 3970)	This is BufferSize, a member of class TIBCustomService.
	OutputBuffer (see page 3971)	This is OutputBuffer, a member of class TIBCustomService.
	OutputBufferOption (see page 3971)	This is OutputBufferOption, a member of class TIBCustomService.
	ServiceQueryParams (see page 3972)	This is ServiceQueryParams, a member of class TIBCustomService.
	ServiceStartParams (see page 3965)	This is ServiceStartParams, a member of class TIBControlService.

Public Methods

	Name	Description
	Attach (see page 3967)	Attaches to the database.
	Create (see page 4003)	Creates an instance of TIBValidationService (see page 4001)
	Destroy (see page 4003)	Destroys the service.
	Detach (see page 3968)	Detaches from the database.
	FetchLimboTransactionInfo (see page 4003)	Gets limbo transaction information.
	FixLimboTransactionErrors (see page 4004)	Fixes limbo transaction errors.
	GetNextChunk (see page 3959)	Gets the next chunk of data.
	GetNextLine (see page 3960)	Starts the service.
	ServiceStart (see page 3963)	Starts the service.

Public Properties

	Name	Description
	Eof (see page 3961)	Indicates that the end of file has been reached.
	Handle (see page 3970)	Returns the database handle.
	IsServiceRunning (see page 3965)	Indicates whether the service is running.
	LimboTransactionInfo (see page 4005)	Returns the limbo transaction information.
	LimboTransactionInfoCount (see page 4005)	Returns the number of elements in the TLimboTransactionInfo (see page 4010) array.
	ServiceParamBySPB (see page 3972)	Returns and sets SPB parameters.

Published Events

	Name	Description
	OnAttach (see page 3973)	Occurs when the database is attached.
	OnLogin (see page 3973)	Occurs during database login.

Published Properties

	Name	Description
	Active (see page 3970)	Sets the active state of the service.
	BufferSize (see page 3961)	Sets or returns the buffer size.
	DatabaseName (see page 4005)	Sets or returns the database name.
	GlobalAction (see page 4005)	Indicates the global transaction action.
	LoginPrompt (see page 3971)	Turns on and off login prompt display.
	Options (see page 4006)	Indicates the options for the database restore.
	Params (see page 3971)	Sets or returns the database parameters
	Protocol (see page 3971)	Sets or returns the network protocol.
	ServerName (see page 3972)	Sets the name of the server on which the services are to be run.
	TraceFlags (see page 3972)	This is TraceFlags, a member of class TIBCustomService.

27.1.16.2 TIBValidationService.Create Constructor

Creates an instance of TIBValidationService (see page 4001)

Pascal

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

C++

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

Description

Call Create to create an instance of TIBValidationService (see page 4001).

Note: You must install InterBase 6 to use this feature.

27.1.16.3 TIBValidationService.Destroy Destructor

Destroys the service.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBValidationService();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Note: You must install InterBase 6 to use this feature.

27.1.16.4 TIBValidationService Methods

The methods of the TIBValidationService class are listed here.

Protected Methods

	Name	Description
	SetServiceStartOptions (see page 4004)	This is SetServiceStartOptions, a member of class TIBValidationService.

Public Methods

	Name	Description
	FetchLimboTransactionInfo (see page 4003)	Gets limbo transaction information.
	FixLimboTransactionErrors (see page 4004)	Fixes limbo transaction errors.

27.1.16.4.1 TIBValidationService.FetchLimboTransactionInfo Method

Gets limbo transaction information.

Pascal

```
procedure FetchLimboTransactionInfo;
```

C++

```
__fastcall FetchLimboTransactionInfo();
```

Description

Call FetchLimboTransactionInfo to get limbo transaction information from the LimboTransactionInfo (see page 4005) property.

Note: You must install InterBase 6 to use this feature.

27.1.16.4.2 TIBValidationService.FixLimboTransactionErrors Method

Fixes limbo transaction errors.

Pascal

```
procedure FixLimboTransactionErrors;
```

C++

```
__fastcall FixLimboTransactionErrors();
```

Description

Call FixLimboTransactionErrors to repair limbo transaction errors.

Note: You must install InterBase 6 to use this feature.

27.1.16.4.3 TIBValidationService.SetServiceStartOptions Method

Pascal

```
procedure SetServiceStartOptions; override;
```

C++

```
virtual __fastcall SetServiceStartOptions();
```

Description

This is SetServiceStartOptions, a member of class TIBValidationService.

27.1.16.5 TIBValidationService Properties

The properties of the TIBValidationService class are listed here.

Public Properties

	Name	Description
	LimboTransactionInfo (see page 4005)	Returns the limbo transaction information.
	LimboTransactionInfoCount (see page 4005)	Returns the number of elements in the TLimboTransactionInfo (see page 4010) array.

Published Properties

	Name	Description
	DatabaseName (see page 4005)	Sets or returns the database name.
	GlobalAction (see page 4005)	Indicates the global transaction action.
	Options (see page 4006)	Indicates the options for the database restore.

27.1.16.5.1 TIBValidationService.DatabaseName Property

Sets or returns the database name.

Pascal

```
property DatabaseName: string;
```

C++

```
__property UnicodeString DatabaseName;
```

Description

Use DatabaseName to set or return the name of the database to be repaired.

Note: You must install InterBase 6 to use this feature.

27.1.16.5.2 TIBValidationService.GlobalAction Property

Indicates the global transaction action.

Pascal

```
property GlobalAction: TTransactionGlobalAction;
```

C++

```
__property TTransactionGlobalAction GlobalAction;
```

Description

Use GlobalAction to set or return the global transaction action. This value indicates what action to take with the transactions that follow.

Note: You must install InterBase 6 to use this feature.

27.1.16.5.3 TIBValidationService.LimboTransactionInfo Property

Returns the limbo transaction information.

Pascal

```
property LimboTransactionInfo [Index: integer]: TLimboTransactionInfo;
```

C++

```
__property TLimboTransactionInfo LimboTransactionInfo[int Index];
```

Description

Use LimboTransactionInfo to return the limbo transaction information.

Note: You must install InterBase 6 to use this feature.

27.1.16.5.4 TIBValidationService.LimboTransactionInfoCount Property

Returns the number of elements in the TLimboTransactionInfo (see page 4010) array.

Pascal

```
property LimboTransactionInfoCount: Integer;
```

C++

```
__property int LimboTransactionInfoCount;
```

Description

Use LimboTransactionInfoCount to set or return the number of elements in the TLimboTransactionInfo (see page 4010) array.

Note: You must install InterBase 6 to use this feature.

27.1.16.5.5 TIBValidationService.Options Property

Indicates the options for the database restore.

Pascal

```
property Options: TValidateOptions;
```

C++

```
__property TValidateOptions Options;
```

Description

Use Options to select the options for the database restore.

Note: You must install InterBase 6 to use this feature.

27.1.17 TLicenseInfo Class

TLicenseInfo stores information about licensed users.

Class Hierarchy

Pascal

```
TLicenseInfo = class;
```

C++

```
class TLicenseInfo;
```

File

IBServices

Description

Key (see page 4008) lists the license key for each licensed user.

Id (see page 4008) lists the user Id (see page 4008) for each licensed user.

Desc (see page 4007) lists the description of each licensed user.

LicensedUsers (see page 4008) indicates the number of users (the number of entries in the lists).

27.1.17.1 TLicenseInfo Members

The following tables list the members exposed by TLicenseInfo.

Public Fields

	Name	Description
	Desc (see page 4007)	This is Desc, a member of class TLicenseInfo.
	Id (see page 4008)	This is Id, a member of class TLicenseInfo.
	Key (see page 4008)	This is Key, a member of class TLicenseInfo.
	LicensedUsers (see page 4008)	This is LicensedUsers, a member of class TLicenseInfo.

Public Methods

	Name	Description
	Create (see page 4007)	This is Create, a member of class TLicenseInfo.
	Destroy (see page 4007)	This is Destroy, a member of class TLicenseInfo.

27.1.17.2 TLicenseInfo.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TLicenseInfo();
```

Description

This is Create, a member of class TLicenseInfo.

27.1.17.3 TLicenseInfo.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLicenseInfo();
```

Description

This is Destroy, a member of class TLicenseInfo.

27.1.17.4 TLicenseInfo Fields

The fields of the TLicenseInfo class are listed here.

Public Fields

	Name	Description
	Desc (see page 4007)	This is Desc, a member of class TLicenseInfo.
	Id (see page 4008)	This is Id, a member of class TLicenseInfo.
	Key (see page 4008)	This is Key, a member of class TLicenseInfo.
	LicensedUsers (see page 4008)	This is LicensedUsers, a member of class TLicenseInfo.

27.1.17.4.1 TLicenseInfo.Desc Field

Pascal

```
Desc: array of string;
```

C++

```
array of string Desc;
```

Description

This is Desc, a member of class TLicenseInfo.

27.1.17.4.2 TLicenseInfo.Id Field

Pascal

```
Id: array of string;
```

C++

```
array of string Id;
```

Description

This is Id, a member of class TLicenseInfo.

27.1.17.4.3 TLicenseInfo.Key Field

Pascal

```
Key: array of string;
```

C++

```
array of string Key;
```

Description

This is Key, a member of class TLicenseInfo.

27.1.17.4.4 TLicenseInfo.LicensedUsers Field

Pascal

```
LicensedUsers: Integer;
```

C++

```
int LicensedUsers;
```

Description

This is LicensedUsers, a member of class TLicenseInfo.

27.1.18 TLicenseMaskInfo Class

TLicenseMaskInfo indicates the software activation certificate options enabled on the server.

Class Hierarchy

IBServices.TLicenseMaskInfo

Pascal

```
TLicenseMaskInfo = class;
```

C++

```
class TLicenseMaskInfo;
```

File

IBServices

Description

LicenseMask (see page 4009) is a bitmask representing the software activation certificate options currently enabled on the server.

CapabilityMask (see page 4009) is a bitmask representing the capabilities currently enabled on the server.

27.1.18.1 TLicenseMaskInfo Members

The following tables list the members exposed by TLicenseMaskInfo.

Public Fields

	Name	Description
	CapabilityMask (see page 4009)	This is CapabilityMask, a member of class TLicenseMaskInfo.
	LicenseMask (see page 4009)	This is LicenseMask, a member of class TLicenseMaskInfo.

27.1.18.2 TLicenseMaskInfo Fields

The fields of the TLicenseMaskInfo class are listed here.

Public Fields

	Name	Description
	CapabilityMask (see page 4009)	This is CapabilityMask, a member of class TLicenseMaskInfo.
	LicenseMask (see page 4009)	This is LicenseMask, a member of class TLicenseMaskInfo.

27.1.18.2.1 TLicenseMaskInfo.CapabilityMask Field

Pascal

```
CapabilityMask: Integer;
```

C++

```
int CapabilityMask;
```

Description

This is CapabilityMask, a member of class TLicenseMaskInfo.

27.1.18.2.2 TLicenseMaskInfo.LicenseMask Field

Pascal

```
LicenseMask: Integer;
```

C++

```
int LicenseMask;
```

Description

This is LicenseMask, a member of class TLicenseMaskInfo.

27.1.19 TLimboTransactionInfo Class

TLimboTransactionInfo stores information about a limbo transaction.

Class Hierarchy

Pascal

```
TLimboTransactionInfo = class;
```

C++

```
class TLimboTransactionInfo;
```

File

IBServices

Description

TLimboTransactionInfo describes limbo transaction. Limbo transactions are usually caused by the failure of a two-phased commit. They can also exist due to system failure or when a single-database transaction is prepared..

MultiDatabase (see page 4012) indicates whether the limbo transaction involves multiple databases.

Transaction ID (see page 4012) is the identifier for the limbo transaction.

Host site is the host name for the server where the transaction was started.

RemoteSite (see page 4012) is the host name for a remote server involved in the transaction.

RemoteDatabasePath (see page 4012) is the path to the remote server.

State (see page 4012) is the current state of the limbo transaction.

Advise (see page 4011) is the suggested action for ending the transaction.

Action (see page 4011) is the action to take for ending the transaction.

27.1.19.1 TLimboTransactionInfo Members

The following tables list the members exposed by TLimboTransactionInfo.

Public Fields

	Name	Description
	Action (see page 4011)	This is Action, a member of class TLimboTransactionInfo.
	Advise (see page 4011)	This is Advise, a member of class TLimboTransactionInfo.
	HostSite (see page 4011)	This is HostSite, a member of class TLimboTransactionInfo.
	ID (see page 4012)	This is ID, a member of class TLimboTransactionInfo.
	MultiDatabase (see page 4012)	This is MultiDatabase, a member of class TLimboTransactionInfo.
	RemoteDatabasePath (see page 4012)	This is RemoteDatabasePath, a member of class TLimboTransactionInfo.
	RemoteSite (see page 4012)	This is RemoteSite, a member of class TLimboTransactionInfo.
	State (see page 4012)	This is State, a member of class TLimboTransactionInfo.

27.1.19.2 TLimboTransactionInfo Fields

The fields of the TLimboTransactionInfo class are listed here.

Public Fields

	Name	Description
	Action (see page 4011)	This is Action, a member of class TLimboTransactionInfo.
	Advise (see page 4011)	This is Advise, a member of class TLimboTransactionInfo.
	HostSite (see page 4011)	This is HostSite, a member of class TLimboTransactionInfo.
	ID (see page 4012)	This is ID, a member of class TLimboTransactionInfo.
	MultiDatabase (see page 4012)	This is MultiDatabase, a member of class TLimboTransactionInfo.
	RemoteDatabasePath (see page 4012)	This is RemoteDatabasePath, a member of class TLimboTransactionInfo.
	RemoteSite (see page 4012)	This is RemoteSite, a member of class TLimboTransactionInfo.
	State (see page 4012)	This is State, a member of class TLimboTransactionInfo.

27.1.19.2.1 TLimboTransactionInfo.Action Field

Pascal

```
Action: TTransactionAction;
```

C++

```
TTransactionAction Action;
```

Description

This is Action, a member of class TLimboTransactionInfo.

27.1.19.2.2 TLimboTransactionInfo.Advise Field

Pascal

```
Advise: TTransactionAdvise;
```

C++

```
TTransactionAdvise Advise;
```

Description

This is Advise, a member of class TLimboTransactionInfo.

27.1.19.2.3 TLimboTransactionInfo.HostSite Field

Pascal

```
HostSite: String;
```

C++

```
UnicodeString HostSite;
```

Description

This is HostSite, a member of class TLimboTransactionInfo.

27.1.19.2.4 TLimboTransactionInfo.ID Field

Pascal

```
ID: Integer;
```

C++

```
int ID;
```

Description

This is ID, a member of class TLimboTransactionInfo.

27.1.19.2.5 TLimboTransactionInfo.MultiDatabase Field

Pascal

```
MultiDatabase: Boolean;
```

C++

```
Boolean MultiDatabase;
```

Description

This is MultiDatabase, a member of class TLimboTransactionInfo.

27.1.19.2.6 TLimboTransactionInfo.RemoteDatabasePath Field

Pascal

```
RemoteDatabasePath: String;
```

C++

```
UnicodeString RemoteDatabasePath;
```

Description

This is RemoteDatabasePath, a member of class TLimboTransactionInfo.

27.1.19.2.7 TLimboTransactionInfo.RemoteSite Field

Pascal

```
RemoteSite: String;
```

C++

```
UnicodeString RemoteSite;
```

Description

This is RemoteSite, a member of class TLimboTransactionInfo.

27.1.19.2.8 TLimboTransactionInfo.State Field

Pascal

```
State: TTransactionState;
```

C++

```
TTransactionState State;
```

Description

This is State, a member of class TLimboTransactionInfo.

27.1.20 TUserInfo Class

TUserInfo stores information about an InterBase user for the security service.

Class Hierarchy

Pascal

```
TUserInfo = class;
```

C++

```
class TUserInfo;
```

File

IBServices

Description

TUserInfo is the user descriptor that the InterBase security service uses to describe a single user.

UserName (see page 4015) is the user's login name.

FirstName (see page 4014) is the user's first name.

MiddleName (see page 4014) is the user's middle name.

LastName (see page 4014) is the user's Last name.

GroupID (see page 4014) is the user's group ID.

UserID (see page 4015) is the user's individual ID.

27.1.20.1 TUserInfo Members

The following tables list the members exposed by TUserInfo.

Public Fields

	Name	Description
	FirstName (see page 4014)	This is FirstName, a member of class TUserInfo.
	GroupID (see page 4014)	This is GroupID, a member of class TUserInfo.
	LastName (see page 4014)	This is LastName, a member of class TUserInfo.
	MiddleName (see page 4014)	This is MiddleName, a member of class TUserInfo.
	UserID (see page 4015)	This is UserID, a member of class TUserInfo.
	UserName (see page 4015)	This is UserName, a member of class TUserInfo.

27.1.20.2 TUserInfo Fields

The fields of the TUserInfo class are listed here.

Public Fields

	Name	Description
	FirstName (see page 4014)	This is FirstName, a member of class TUserInfo.
	GroupID (see page 4014)	This is GroupID, a member of class TUserInfo.
	LastName (see page 4014)	This is LastName, a member of class TUserInfo.
	MiddleName (see page 4014)	This is MiddleName, a member of class TUserInfo.
	UserID (see page 4015)	This is UserID, a member of class TUserInfo.
	UserName (see page 4015)	This is UserName, a member of class TUserInfo.

27.1.20.2.1 TUserInfo.FirstName Field

Pascal

```
FirstName: string;
```

C++

```
UnicodeString FirstName;
```

Description

This is FirstName, a member of class TUserInfo.

27.1.20.2.2 TUserInfo.GroupID Field

Pascal

```
GroupID: Integer;
```

C++

```
int GroupID;
```

Description

This is GroupID, a member of class TUserInfo.

27.1.20.2.3 TUserInfo.LastName Field

Pascal

```
LastName: string;
```

C++

```
UnicodeString LastName;
```

Description

This is LastName, a member of class TUserInfo.

27.1.20.2.4 TUserInfo.MiddleName Field

Pascal

```
MiddleName: string;
```

C++

```
UnicodeString MiddleName;
```

Description

This is MiddleName, a member of class TUserInfo.

27.1.20.2.5 TUserInfo.UserID Field

Pascal

```
UserID: Integer;
```

C++

```
int UserID;
```

Description

This is UserID, a member of class TUserInfo.

27.1.20.2.6 TUserInfo.UserName Field

Pascal

```
UserName: string;
```

C++

```
UnicodeString UserName;
```

Description

This is UserName, a member of class TUserInfo.

27.1.21 TVersionInfo Class

TVersionInfo represents version information about an InterBase server.

Class Hierarchy

Pascal

```
TVersionInfo = class;
```

C++

```
class TVersionInfo;
```

File

IBServices

Description

TVersionInfo supplies the following version information:

ServerVersion (see page 4016) gives the version of the InterBase server.

ServerImplementation (see page 4016) gives the implementation string.

ServiceVersion (see page 4016) give the version number for the service that fetches the version information.

27.1.21.1 TVersionInfo Members

The following tables list the members exposed by TVersionInfo.

Public Fields

	Name	Description
	ServerImplementation (see page 4016)	This is ServerImplementation, a member of class TVersionInfo.
	ServerVersion (see page 4016)	This is ServerVersion, a member of class TVersionInfo.
	ServiceVersion (see page 4016)	This is ServiceVersion, a member of class TVersionInfo.

27.1.21.2 TVersionInfo Fields

The fields of the TVersionInfo class are listed here.

Public Fields

	Name	Description
	ServerImplementation (see page 4016)	This is ServerImplementation, a member of class TVersionInfo.
	ServerVersion (see page 4016)	This is ServerVersion, a member of class TVersionInfo.
	ServiceVersion (see page 4016)	This is ServiceVersion, a member of class TVersionInfo.

27.1.21.2.1 TVersionInfo.ServerImplementation Field

Pascal

```
ServerImplementation: string;
```

C++

```
UnicodeString ServerImplementation;
```

Description

This is ServerImplementation, a member of class TVersionInfo.

27.1.21.2.2 TVersionInfo.ServerVersion Field

Pascal

```
ServerVersion: String;
```

C++

```
UnicodeString ServerVersion;
```

Description

This is ServerVersion, a member of class TVersionInfo.

27.1.21.2.3 TVersionInfo.ServiceVersion Field

Pascal

```
ServiceVersion: Integer;
```

C++

```
int ServiceVersion;
```

Description

This is ServiceVersion, a member of class TVersionInfo.

27.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBackupOption (see page 4018)	This is record IBServices.TBackupOption.
	TLicensingAction (see page 4019)	TLicensingAction indicates whether to add or remove InterBase software activation certificates.
	TOutputBufferOption (see page 4019)	This is record IBServices.TOutputBufferOption.
	TPropertyOption (see page 4020)	This is record IBServices.TPropertyOption.
	TProtocol (see page 4020)	TProtocol identifies a network protocol.
	TRestoreOption (see page 4021)	This is record IBServices.TRestoreOption.
	TSecurityAction (see page 4021)	TSecurityAction indicates what operation the InterBase security service should perform.
	TSecurityModifyParam (see page 4022)	This is record IBServices.TSecurityModifyParam.
	TShutdownMode (see page 4022)	TShutdownMode indicates what steps to take when shutting down the InterBase database.
	TStatOption (see page 4023)	This is record IBServices.TStatOption.
	TTransactionAction (see page 4023)	TTransactionAction indicates the action a transaction can take when it times out.
	TTransactionAdvise (see page 4024)	This is record IBServices.TTransactionAdvise.
	TTransactionGlobalAction (see page 4024)	TTransactionGlobalAction indicates the action to take on a set of transactions.
	TTransactionState (see page 4025)	This is record IBServices.TTransactionState.
	TValidateOption (see page 4026)	This is record IBServices.TValidateOption.

27.2.1 IBServices.TBackupOption Enumeration

Pascal

```
TBackupOption = (
  IgnoreChecksums,
  IgnoreLimbo,
  MetadataOnly,
  NoGarbageCollection,
  OldMetadataDesc,
  NonTransportable,
  ConvertExtTables
);
```

C++

```
enum TBackupOption {
  IgnoreChecksums,
  IgnoreLimbo,
  MetadataOnly,
  NoGarbageCollection,
  OldMetadataDesc,
  NonTransportable,
  ConvertExtTables
};
```

File

IBServices

Description

This is record IBServices.TBackupOption.

27.2.2 IBServices.TLicensingAction Enumeration

TLicensingAction indicates whether to add or remove InterBase software activation certificates.

Pascal

```
TLicensingAction = (
  LicenseAdd,
  LicenseRemove
);
```

C++

```
enum TLicensingAction {
  LicenseAdd,
  LicenseRemove
};
```

File

IBServices

Description

TLicensingAction values are:

Value Meaning

LicenseAdd Add an InterBase software activation certificate.

LicenseRemove Remove an InterBase software activation certificate.

27.2.3 IBServices.TOutputBufferOption Enumeration

Pascal

```
TOutputBufferOption = (
  ByLine,
  ByChunk
);
```

C++

```
enum TOutputBufferOption {
  ByLine,
  ByChunk
};
```

File

IBServices

Description

This is record IBServices.TOutputBufferOption.

27.2.4 IBServices.TPropertyOption Enumeration

Pascal

```
TPropertyOption = (
  Database,
  License,
  LicenseMask,
  ConfigParameters,
  Version
);
```

C++

```
enum TPropertyOption {
  Database,
  License,
  LicenseMask,
  ConfigParameters,
  Version
};
```

File

IBServices

Description

This is record IBServices.TPropertyOption.

27.2.5 IBServices.TProtocol Enumeration

TProtocol identifies a network protocol.

Pascal

```
TProtocol = (
  TCP,
  SPX,
  NamedPipe,
  Local
);
```

C++

```
enum TProtocol {
  TCP,
  SPX,
  NamedPipe,
  Local
};
```

File

IBServices

Description

TProtocol values are:

Local	
NamedPipe	Named pipe (FIFO)
SPX	IPX/SPX
TCP	Transport control protocol

27.2.6 IBServices.TRestoreOption Enumeration

Pascal

```
TRestoreOption = (
  DeactivateIndexes,
  NoShadow,
  NoValidityCheck,
  OneRelationAtATime,
  Replace,
  CreateNewDB,
  UseAllSpace
);
```

C++

```
enum TRestoreOption {
  DeactivateIndexes,
  NoShadow,
  NoValidityCheck,
  OneRelationAtATime,
  Replace,
  CreateNewDB,
  UseAllSpace
};
```

File

IBServices

Description

This is record IBServices.TRestoreOption.

27.2.7 IBServices.TSecurityAction Enumeration

TSecurityAction indicates what operation the InterBase security service should perform.

Pascal

```
TSecurityAction = (
  ActionAddUser,
  ActionDeleteUser,
  ActionModifyUser,
  ActionDisplayUser
);
```

C++

```
enum TSecurityAction {
  ActionAddUser,
  ActionDeleteUser,
```

```
    ActionModifyUser,
    ActionDisplayUser
  };
```

File

IBServices

Description

TSecurityAction values are:

ActionAddUser	Add the user
ActionDeleteUser	Delete the user
ActionDisplayUser	Display the user's information
ActionModifyUser	Modify the user account

27.2.8 IBServices.TSecurityModifyParam Enumeration

Pascal

```
  TSecurityModifyParam = (
    ModifyFirstName,
    ModifyMiddleName,
    ModifyLastName,
    ModifyUserId,
    ModifyGroupId,
    ModifyPassword
  );
```

C++

```
  enum TSecurityModifyParam {
    ModifyFirstName,
    ModifyMiddleName,
    ModifyLastName,
    ModifyUserId,
    ModifyGroupId,
    ModifyPassword
  };
```

File

IBServices

Description

This is record IBServices.TSecurityModifyParam.

27.2.9 IBServices.TShutdownMode Enumeration

TShutdownMode indicates what steps to take when shutting down the InterBase database.

Pascal

```
  TShutdownMode = (
    Forced,
    DenyTransaction,
```

```
  DenyAttachment
);
```

C++

```
enum TShutdownMode {
  Forced,
  DenyTransaction,
  DenyAttachment
};
```

File

IBServices

Description

TShutdownMode values are:

Forced	Shut down the database after the specified number of seconds. To shut down the database immediately, set the number of seconds to 0.
DenyTransaction	Deny new transactions for the specified number of seconds and then shut down. The shutdown will fail if there are still active transactions to the database.
DenyAttachment	Deny new attachments for the specified number of seconds and then shut down. The shutdown will fail if there are still active attachments to the database.

27.2.10 IBServices.TStatOption Enumeration

Pascal

```
TStatOption = (
  DataPages,
  DbLog,
  HeaderPages,
  IndexPages,
  SystemRelations
);
```

C++

```
enum TStatOption {
  DataPages,
  DbLog,
  HeaderPages,
  IndexPages,
  SystemRelations
};
```

File

IBServices

Description

This is record IBServices.TStatOption.

27.2.11 IBServices.TTransactionAction Enumeration

TTransactionAction indicates the action a transaction can take when it times out.

Pascal

```
TTransactionAction = (
  CommitAction,
  RollbackAction
);
```

C++

```
enum TTransactionAction {
  CommitAction,
  RollbackAction
};
```

File

IBServices

Description

TTransactionAction values are:

Value Meaning

TARollback Roll back the transaction.

TACommit Commit the transaction.

TARollbackRetaining Roll back the transaction, but retain the current transaction context. (Requires InterBase 6.0 or later)

TACommitRetaining Commit the transaction, but retain the current transaction context.

27.2.12 IBServices.TTransactionAdvise Enumeration

Pascal

```
TTransactionAdvise = (
  CommitAdvise,
  RollbackAdvise,
  UnknownAdvise
);
```

C++

```
enum TTransactionAdvise {
  CommitAdvise,
  RollbackAdvise,
  UnknownAdvise
};
```

File

IBServices

Description

This is record IBServices.TTransactionAdvise.

27.2.13 IBServices.TTransactionGlobalAction Enumeration

TTransactionGlobalAction indicates the action to take on a set of transactions.

Pascal

```
TTransactionGlobalAction = (
  CommitGlobal,
  RollbackGlobal,
  RecoverTwoPhaseGlobal,
  NoGlobalAction
);
```

C++

```
enum TTransactionGlobalAction {
  CommitGlobal,
  RollbackGlobal,
  RecoverTwoPhaseGlobal,
  NoGlobalAction
};
```

File

IBServices

Description

TTransactionGlobalAction values are:

Value Meaning

CommitGlobal Commit the transactions.

RollbackGlobal Roll back the transactions.

RecoverTwoPhaseGlobal Perform automated two-phase commit recovery on the transactions.

NoGlobalAction Take no action.

27.2.14 IBServices.TTransactionState Enumeration

Pascal

```
TTransactionState = (
  LimboState,
  CommitState,
  RollbackState,
  UnknownState
);
```

C++

```
enum TTransactionState {
  LimboState,
  CommitState,
  RollbackState,
  UnknownState
};
```

File

IBServices

Description

This is record IBServices.TTransactionState.

27.2.15 IBServices.TValidateOption Enumeration

Pascal

```
TValidateOption = (
  LimboTransactions,
  CheckDB,
  IgnoreChecksum,
  KillShadows,
  MendDB,
  SweepDB,
  ValidateDB,
  ValidateFull
);
```

C++

```
enum TValidateOption {
  LimboTransactions,
  CheckDB,
  IgnoreChecksum,
  KillShadows,
  MendDB,
  SweepDB,
  ValidateDB,
  ValidateFull
};
```

File

IBServices

Description

This is record IBServices.TValidateOption.

27.3 Types

The following table lists types in this documentation.

Types

Name	Description
TBackupOptions (see page 4027)	TBackupOption (see page 4018) and TBackupOptions indicate how TIBBackupService (see page 3948) performs database backup.
TLoginEvent (see page 4028)	This is type IBServices.TLoginEvent.
TPropertyOptions (see page 4028)	TPropertyOption (see page 4020) and TPropertyOptions indicate groups of InterBase server properties.
TRestoreOptions (see page 4029)	TRestoreOption (see page 4021) and TRestoreOptions indicate what operations take place when restoring an InterBase database.
TSecurityModifyParams (see page 4030)	This is type IBServices.TSecurityModifyParams.
TStatOptions (see page 4030)	TStatOption (see page 4023) and TStatOptions indicate the different statistical information available about a database.
TValidateOptions (see page 4031)	TValidateOption (see page 4026) and TValidateOptions indicate what operations the validation service should perform when restoring a database.

27.3.1 IBServices.TBackupOptions Type

TBackupOption (see page 4018) and TBackupOptions indicate how TIBBackupService (see page 3948) performs database backup.

Pascal

```
TBackupOptions = set of TBackupOption;
```

C++

```
TBackupOption TBackupOptions;
```

File

IBServices

Description

TBackupOptions is a set of TBackupOption (see page 4018) values. The following table lists the possible values:

IgnoreChecksums	A checksum is a page-by-page analysis of data to verify its integrity. A bad checksum means that a data page has been randomly overwritten, for example, due to a system crash. Checksum errors indicate data corruption, and InterBase normally prevents you from backing up a database if bad checksums are detected. Examine the data the next time you restore the database. To ignore checksums during backup, set IgnoreChecksums to true.
IgnoreLimbo	Limbo transactions are usually caused by the failure of a two-phased commit. They can also exist due to system failure or when a single-database transaction is prepared. To ignore limbo transactions during backup, set IgnoreLimbo to true. When your application ignores limbo transactions during backup, it ignores all records versions created by any limbo transaction, finds the most recently committed version of a record, and backs up that version.

MetadataOnly	When backing up a database, you can exclude its data, saving only the metadata. You might want to do this to: Retain a record of the metadata before it is modified. Create an empty copy of the database. The copy has the same metadata but can be populated with different data. Set MetaDataOnly to true to save only the metadata.
NoGarbageCollection	Garbage collections marks space used by old versions of data records as free for reuse. Generally, you want to perform garbage collection during backup. To disable garbage collection, set NoGarbageCollection to true.
OldMetadataDesc	Set OldMetadataDesc to true if you want to back up metadata in the old-style format.
NonTransportable	Set to NonTransportable to true if you do not want data to be moved to a machine with a different operating system than the one on which the backup was performed. If NonTransportable is set to false (the default), then data is saved in a generic format, enabling you to restore to any machine that supports InterBase.
ConvertExtTables	Set ConvertExtTables to true if you want to convert external files into internal tables during the backup

27.3.2 IBServices.TLoginEvent Type

Pascal

```
TLoginEvent = procedure (Database: TIBCustomService; LoginParams: TStrings) of object;
```

C++

```
(Database: TIBCustomService; LoginParams: TStrings) ( TLoginEvent)();
```

File

IBServices

Description

This is type IBServices.TLoginEvent.

27.3.3 IBServices.TPropertyOptions Type

TPropertyOption (see page 4020) and TPropertyOptions indicate groups of InterBase server properties.

Pascal

```
TPropertyOptions = set of TPropertyOption;
```

C++

```
TPropertyOption TPropertyOptions;
```

File

IBServices

Description

TPropertyOptions is a set of TPropertyOption (see page 4020) values. Each TPropertyOption (see page 4020) value indicates a type of server property you wish to see. Each type of sever property has a corresponding data structure that holds

the information, as indicated in the following table:

Database	TDatabaseInfo (see page 3944)
License	TLicenseInfo (see page 4006)
ConfigParameters	TConfigParams (see page 3941)
Version	TVersionInfo (see page 4015)

27.3.4 IBServices.TRestoreOptions Type

TRestoreOption (see page 4021) and TRestoreOptions indicate what operations take place when restoring an InterBase database.

Pascal

```
TRestoreOptions = set of TRestoreOption;
```

C++

```
TRestoreOption TRestoreOptions;
```

File

IBServices

Description

TRestoreOptions is a set of TRestoreOption (see page 4021) values. The following table lists the possible values:

DeactivateIndexes	Normally, InterBase rebuilds indexes when it restores a database. If the database contained duplicate entries in a unique index when it was backed up, restoration fails. Duplicate values can be introduced into a database if indexes were temporarily made inactive (for example, to allow insertion of many records or to re-balance an index). To enable restoration to succeed in this case, set DeactivateIndexes to true. This makes indexes inactive and prevents them from rebuilding.
NoShadow	Normally, InterBase rebuilds previously defined database shadows when it restores a database. To make sure the shadows are not built, set the NoShadow option to true.
NoValidityCheck	If you redefine validity constraints in a database where data is already entered, your data might no longer satisfy the validity constraints. You might not discover this until you try to restore the database, at which time an error about invalid data appears. To restore a database that contains invalid data, set NoValidity to true. This deletes validity constraints from the metadata. After the database is restored, change the data to make it valid according to the new integrity constraints. Then add back the constraints that were deleted. **Important:** Always make a copy of metadata before redefining it; for example with the MetadataOnly option of the Options property of TIBBackupService (see page 3948).
OneRelationAtATime	Set OneRelationATime to true to restore the metadata and data for each table together, one table at a time. This option is useful when you are having trouble restoring a backup file; for example, if the data is corrupt or invalid according to integrity constraints.
Replace	Your application cannot overwrite an existing database file unless you set Replace to true. If you attempt to restore to an existing database name and this option is not set, then the restore will not proceed. **Important:** Do not restore an existing database file when clients are operating on it. A safer approach is to rename the existing database file, restore the database, then drop or archive the old database as needed.

CreateNew	To restore a database to a new file, set the Create option to true.
UseAllSpace	When InterBase restores a database, it fills each data page to 80% of capacity by default. To set your application to restore a database with a 100% fill ratio on each data page, set the UseAllSpace option to true.

27.3.5 IBServices.TSecurityModifyParams Type

Pascal

```
TSecurityModifyParams = set of TSecurityModifyParam;
```

C++

```
TSecurityModifyParam TSecurityModifyParams;
```

File

IBServices

Description

This is type IBServices.TSecurityModifyParams.

27.3.6 IBServices.TStatOptions Type

TStatOption (see page 4023) and TStatOptions indicate the different statistical information available about a database.

Pascal

```
TStatOptions = set of TStatOption;
```

C++

```
TStatOption TStatOptions;
```

File

IBServices

Description

TStatOptions is a set of TStatOption (see page 4023) values. The following table lists the possible values:

Value Meaning

DataPages Request statistics for data tables in the database.

DbLog Stop reporting statistics after reporting information on the log pages.

HeaderPages Stop reporting statistics after reporting the information on the header page.

IndexPages Request statistics for the user indexes in the database.

SystemRelations Request statistics for system tables and indexes in addition to user tables and indexes.

27.3.7 IBServices.TValidateOptions Type

TValidateOption (see page 4026) and TValidateOptions indicate what operations the validation service should perform when restoring a database.

Pascal

```
TValidateOptions = set of TValidateOption;
```

C++

```
TValidateOption TValidateOptions;
```

File

IBServices

Description

TValidateOptions is a set of TValidateOption (see page 4026) values. The following table lists the possible values:

LimboTransactions	Return a list of limbo transactions
CheckDB	Validate the database, but don't repair it. Used in conjunction with ValidateDB.
IgnoreChecksum	Ignore all checksum errors
KillShadows	Kill all unavailable shadow files
MendDB	Prepare a corrupt database for backup
SweepDB	Perform a database sweep
ValidateDB	Validate the database structure
ValidateFull	Validate record fragments

27.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DefaultBufferSize (see page 4032)	This is constant IBServices.DefaultBufferSize.
SPBConstantNames (see page 4032)	This is constant IBServices.SPBConstantNames.
SPBConstantValues (see page 4033)	This is constant IBServices.SPBConstantValues.
SPBPrefix (see page 4033)	This is constant IBServices.SPBPrefix.

27.4.1 IBServices.DefaultBufferSize Constant

Pascal

```
DefaultBufferSize = 32000;
```

C++

```
DefaultBufferSize = 32000;
```

File

IBServices

Description

This is constant IBServices.DefaultBufferSize.

27.4.2 IBServices.SPBConstantNames Constant

Pascal

```
SPBConstantNames: array[1..isc_spb_last_spb_constant] of String = ( 'user_name',
'sys_user_name', 'sys_user_name_enc', 'password', 'password_enc', 'command_line', 'db_name',
'verbose', 'options', 'connect_timeout', 'dummy_packet_interval', 'sql_role_name' );
```

C++

```
array[1..isc_spb_last_spb_constant] of String SPBConstantNames = ( 'user_name',
'sys_user_name', 'sys_user_name_enc', 'password', 'password_enc', 'command_line', 'db_name',
'verbose', 'options', 'connect_timeout', 'dummy_packet_interval', 'sql_role_name' );
```

File

IBServices

Description

This is constant IBServices.SPBConstantNames.

27.4.3 IBServices.SPBConstantValues Constant

Pascal

```
SPBConstantValues: array[1..isc_spb_last_spb_constant] of Integer = (
isc_spb_user_name_mapped_to_server, isc_spb_sys_user_name_mapped_to_server,
isc_spb_sys_user_name_enc_mapped_to_server, isc_spb_password_mapped_to_server,
isc_spb_password_enc_mapped_to_server, isc_spb_command_line_mapped_to_server,
isc_spb_dbname_mapped_to_server, isc_spb_verbose_mapped_to_server,
isc_spb_options_mapped_to_server, isc_spb_connect_timeout_mapped_to_server,
isc_spb_dummy_packet_interval_mapped_to_server, isc_spb_sql_role_name_mapped_to_server );
```

C++

```
array[1..isc_spb_last_spb_constant] of Integer SPBConstantValues = (
isc_spb_user_name_mapped_to_server, isc_spb_sys_user_name_mapped_to_server,
isc_spb_sys_user_name_enc_mapped_to_server, isc_spb_password_mapped_to_server,
isc_spb_password_enc_mapped_to_server, isc_spb_command_line_mapped_to_server,
isc_spb_dbname_mapped_to_server, isc_spb_verbose_mapped_to_server,
isc_spb_options_mapped_to_server, isc_spb_connect_timeout_mapped_to_server,
isc_spb_dummy_packet_interval_mapped_to_server, isc_spb_sql_role_name_mapped_to_server );
```

File

IBServices

Description

This is constant IBServices.SPBConstantValues.

27.4.4 IBServices.SPBPrefix Constant

Pascal

```
SPBPrefix = 'isc_spb_';
```

C++

```
SPBPrefix = 'isc_spb_';
```

File

IBServices

Description

This is constant IBServices.SPBPrefix.

28 IBSQL Namespace

This is namespace IBSQL.

Topics

Name	Description
Classes (see page 4036)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4098)	The following table lists structs, records, enums in this documentation.
Types (see page 4100)	The following table lists types in this documentation.

Classes

	Name	Description
	TIBBatch (see page 4036)	TIBBatch is the abstract base class for the TIBBatchInput (see page 4039) and TIBBatchOutput (see page 4040) objects.
	TIBBatchInput (see page 4039)	TIBBatchInput is the abstract class for performing all batch input.
	TIBBatchOutput (see page 4040)	TIBBatchOutput is the abstract class for performing all batch output.
	TIBInputDelimitedFile (see page 4042)	TIBInputDelimitedFile performs batch input of data from delimited files.
	TIBInputRawFile (see page 4047)	TIBInputRawFile inputs data from a raw file.
	TIBOutputDelimitedFile (see page 4049)	TIBOutputDelimitedFile performs batch output of data to delimited files.
	TIBOutputRawFile (see page 4053)	TIBOutputRawFile outputs data to a raw file format.
	TIBSQL (see page 4055)	TIBSQL provides an object for executing an InterBase SQL (see page 4076) statement with minimal overhead.
	TIBXSQLDA (see page 4078)	TIBXSQLDA provides properties and methods for use with the IBSQL component.
	TIBXSQLVAR (see page 4086)	TIBXSQLVAR provides properties and methods for use with the IBSQL component.

Structs, Records, Enums

	Name	Description
	TIBSQLTypes (see page 4098)	TIBSQLTypes indicates the type of an SQL query against an InterBase database.

Types

Name	Description
TIBXSQLVARArray (see page 4100)	This is type IBSQL.TIBXSQLVARArray.

28.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBBatch (see page 4036)	TIBBatch is the abstract base class for the TIBBatchInput (see page 4039) and TIBBatchOutput (see page 4040) objects.
	TIBBatchInput (see page 4039)	TIBBatchInput is the abstract class for performing all batch input.
	TIBBatchOutput (see page 4040)	TIBBatchOutput is the abstract class for performing all batch output.
	TIBInputDelimitedFile (see page 4042)	TIBInputDelimitedFile performs batch input of data from delimited files.
	TIBInputRawFile (see page 4047)	TIBInputRawFile inputs data from a raw file.
	TIBOutputDelimitedFile (see page 4049)	TIBOutputDelimitedFile performs batch output of data to delimited files.
	TIBOutputRawFile (see page 4053)	TIBOutputRawFile outputs data to a raw file format.
	TIBSQL (see page 4055)	TIBSQL provides an object for executing an InterBase SQL (see page 4076) statement with minimal overhead.
	TIBXSQLDA (see page 4078)	TIBXSQLDA provides properties and methods for use with the IBSQL (see page 4035) component.
	TIBXSQLVAR (see page 4086)	TIBXSQLVAR provides properties and methods for use with the IBSQL (see page 4035) component.

28.1.1 TIBBatch Class

TIBBatch is the abstract base class for the TIBBatchInput (see page 4039) and TIBBatchOutput (see page 4040) objects.

Class Hierarchy

Pascal

```
TIBBatch = class(TObject);
```

C++

```
class TIBBatch : public TObject;
```

File

IBSQL

Description

Use a TIBBatch object to provide properties and methods for use with the TIBBatchInput (see page 4039) and TIBBatchOutput (see page 4040) objects, which make it possible to input and output data in virtually any format.

Descendents of this class can specify a file name (for input or output), and a TIBXSQLDA (see page 4078) object representing a record or parameters. The ReadyFile (see page 4038) method is called right before performing the batch input or output.

28.1.1.1 TIBBatch Members

The following tables list the members exposed by TIBBatch.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

Public Methods

	Name	Description
	ReadyFile (see page 4038)	Prepares the file.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.1.2 TIBBatch Fields

The fields of the TIBBatch class are listed here.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

28.1.1.2.1 TIBBatch.FColumns Field

Pascal

```
FColumns: TIBXSQLDA;
```

C++

```
TIBXSQLDA FColumns;
```

Description

This is FColumns, a member of class TIBBatch.

28.1.1.2.2 TIBBatch.FFilename Field

Pascal

```
FFilename: String;
```

C++

```
UnicodeString FFilename;
```

Description

This is FFilename, a member of class TIBBatch.

28.1.1.2.3 TIBBatch.FParams Field

Pascal

```
FParams: TIBXSQLDA;
```

C++

```
TIBXSQLDA FParams;
```

Description

This is FParams, a member of class TIBBatch.

28.1.1.3 TIBBatch Methods

The methods of the TIBBatch class are listed here.

Public Methods

	Name	Description
	ReadyFile (see page 4038)	Prepares the file.

28.1.1.3.1 TIBBatch.ReadyFile Method

Prepares the file.

Pascal

```
procedure ReadyFile; virtual; abstract;
```

C++

```
virtual __fastcall ReadyFile() = 0;
```

Description

Call ReadyFile to prepare the file for input or output.

28.1.1.4 TIBBatch Properties

The properties of the TIBBatch class are listed here.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.1.4.1 TIBBatch.Columns Property

Returns the XSQLDA columns.

Pascal

```
property Columns: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA Columns;
```

Description

Use the Columns property to retrieve or set the XSQLDA columns.

28.1.1.4.2 TIBBatch.Filename Property

Sets or displays the name of the external file.

Pascal

```
property Filename: String;
```

C++

```
__property UnicodeString Filename;
```

Description

Use the Filename property to set or display the external file name.

28.1.1.4.3 TIBBatch.Params Property

Returns the XSQLDA parameters.

Pascal

```
property Params: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA Params;
```

Description

Use the Params property to retrieve or set the XSQLDA parameters.

28.1.2 TIBBatchInput Class

TIBBatchInput is the abstract class for performing all batch input.

Class Hierarchy

Pascal

```
TIBBatchInput = class(TIBBatch);
```

C++

```
class TIBBatchInput : public TIBBatch;
```

File

IBSQL

Description

Use a TIBBatchInput object to provide properties and methods for performing all batch input.

28.1.2.1 TIBBatchInput Members

The following tables list the members exposed by TIBBatchInput.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

Public Methods

	Name	Description
	ReadParameters (see page 4040)	Reads the XSQLDA input parameters.
	ReadyFile (see page 4038)	Prepares the file.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.2.2 TIBBatchInput Methods

The methods of the TIBBatchInput class are listed here.

Public Methods

	Name	Description
	ReadParameters (see page 4040)	Reads the XSQLDA input parameters.

28.1.2.2.1 TIBBatchInput.ReadParameters Method

Reads the XSQLDA input parameters.

Pascal

```
function ReadParameters: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean ReadParameters() = 0;
```

Description

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA) from the specified file.

28.1.3 TIBBatchOutput Class

TIBBatchOutput is the abstract class for performing all batch output.

Class Hierarchy

Pascal

```
TIBBatchOutput = class(TIBBatch);
```

C++

```
class TIBBatchOutput : public TIBBatch;
```

File

IBSQL

Description

Use a TIBBatchOutput object to provide properties and methods for performing all batch output.

28.1.3.1 TIBBatchOutput Members

The following tables list the members exposed by TIBBatchOutput.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

Public Methods

	Name	Description
	ReadyFile (see page 4038)	Prepares the file.
	WriteColumns (see page 4041)	Outputs the data in columns in the XSQLDA to the specified file.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.3.2 TIBBatchOutput Methods

The methods of the TIBBatchOutput class are listed here.

Public Methods

	Name	Description
	WriteColumns (see page 4041)	Outputs the data in columns in the XSQLDA to the specified file.

28.1.3.2.1 TIBBatchOutput.WriteColumns Method

Outputs the data in columns in the XSQLDA to the specified file.

Pascal

```
function WriteColumns: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean WriteColumns() = 0;
```

Description

Call WriteColumns to output data in columns in the extended SQL descriptor area (XSQLDA) to the specified file.

28.1.4 TIBInputDelimitedFile Class

TIBInputDelimitedFile performs batch input of data from delimited files.

Class Hierarchy

Pascal

```
TIBInputDelimitedFile = class(TIBBatchInput);
```

C++

```
class TIBInputDelimitedFile : public TIBBatchInput;
```

File

IBSQL

Description

Use a TIBInputDelimitedFile object to perform batch input of data from delimited files.

28.1.4.1 TIBInputDelimitedFile Members

The following tables list the members exposed by TIBInputDelimitedFile.

Protected Fields

	Name	Description
	FColDelimiter (see page 4043)	This is FColDelimiter, a member of class TIBInputDelimitedFile.
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FEOF (see page 4043)	This is FEOF, a member of class TIBInputDelimitedFile.
	FFile (see page 4044)	This is FFile, a member of class TIBInputDelimitedFile.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FLookAhead (see page 4044)	This is FLookAhead, a member of class TIBInputDelimitedFile.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.
	FReadBlanksAsNull (see page 4044)	This is FReadBlanksAsNull, a member of class TIBInputDelimitedFile.
	FRowDelimiter (see page 4044)	This is FRowDelimiter, a member of class TIBInputDelimitedFile.
	FSkipTitles (see page 4045)	This is FSkipTitles, a member of class TIBInputDelimitedFile.

Public Methods

	Name	Description
	Destroy (see page 4043)	Destroys the instance of TIBInputDelimitedFile (see page 4042).
	GetColumn (see page 4045)	Returns the contents of a column.
	ReadParameters (see page 4045)	Reads the input parameters of the XSQLDA.
	ReadyFile (see page 4045)	Prepares the output for the XSQLDA.

Public Properties

	Name	Description
	ColDelimiter (see page 4046)	Sets the column delimiter for the input file.
	Columns (see page 4038)	Returns the XSQLDA columns.

	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.
	ReadBlanksAsNull (see page 4046)	Reads blank spaces in the input file as null characters.
	RowDelimiter (see page 4046)	Sets the row delimiter for the input file.
	SkipTitles (see page 4047)	Skips the first record of a delimited file.

28.1.4.2 TIBInputDelimitedFile.Destroy Destructor

Destroys the instance of TIBInputDelimitedFile (see page 4042).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBInputDelimitedFile();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBInputDelimitedFile (see page 4042) object is not already freed and only then calls Destroy.

28.1.4.3 TIBInputDelimitedFile Fields

The fields of the TIBInputDelimitedFile class are listed here.

Protected Fields

	Name	Description
	FColDelimiter (see page 4043)	This is FColDelimiter, a member of class TIBInputDelimitedFile.
	FEOF (see page 4043)	This is FEOF, a member of class TIBInputDelimitedFile.
	FFile (see page 4044)	This is FFile, a member of class TIBInputDelimitedFile.
	FLookAhead (see page 4044)	This is FLookAhead, a member of class TIBInputDelimitedFile.
	FReadBlanksAsNull (see page 4044)	This is FReadBlanksAsNull, a member of class TIBInputDelimitedFile.
	FRowDelimiter (see page 4044)	This is FRowDelimiter, a member of class TIBInputDelimitedFile.
	FSkipTitles (see page 4045)	This is FSkipTitles, a member of class TIBInputDelimitedFile.

28.1.4.3.1 TIBInputDelimitedFile.FColDelimiter Field

Pascal

```
FColDelimiter: string;
```

C++

```
UnicodeString FColDelimiter;
```

Description

This is FColDelimiter, a member of class TIBInputDelimitedFile.

28.1.4.3.2 TIBInputDelimitedFile.FEOF Field

Pascal

```
FEOF: Boolean;
```

C++

```
Boolean FEOF;
```

Description

This is FEOF, a member of class TIBInputDelimitedFile.

28.1.4.3.3 TIBInputDelimitedFile.FFile Field

Pascal

```
FFile: TFileStream;
```

C++

```
TFileStream FFile;
```

Description

This is FFile, a member of class TIBInputDelimitedFile.

28.1.4.3.4 TIBInputDelimitedFile.FLookAhead Field

Pascal

```
FLookAhead: Char;
```

C++

```
Char FLookAhead;
```

Description

This is FLookAhead, a member of class TIBInputDelimitedFile.

28.1.4.3.5 TIBInputDelimitedFile.FReadBlanksAsNull Field

Pascal

```
FReadBlanksAsNull: Boolean;
```

C++

```
Boolean FReadBlanksAsNull;
```

Description

This is FReadBlanksAsNull, a member of class TIBInputDelimitedFile.

28.1.4.3.6 TIBInputDelimitedFile.FRowDelimiter Field

Pascal

```
FRowDelimiter: string;
```

C++

```
UnicodeString FRowDelimiter;
```

Description

This is FRowDelimiter, a member of class TIBInputDelimitedFile.

28.1.4.3.7 TIBInputDelimitedFile.FSkipTitles Field

Pascal

```
FSkipTitles: Boolean;
```

C++

```
Boolean FSkipTitles;
```

Description

This is FSkipTitles, a member of class TIBInputDelimitedFile.

28.1.4.4 TIBInputDelimitedFile Methods

The methods of the TIBInputDelimitedFile class are listed here.

Public Methods

	Name	Description
	GetColumn (see page 4045)	Returns the contents of a column.
V	ReadParameters (see page 4045)	Reads the input parameters of the XSQLDA.
V	ReadyFile (see page 4045)	Prepares the output for the XSQLDA.

28.1.4.4.1 TIBInputDelimitedFile.GetColumn Method

Returns the contents of a column.

Pascal

```
function GetColumn(var Col: string): Integer;
```

C++

```
__fastcall int GetColumn(UnicodeString Col);
```

Description

Call GetColumn to return the contents of a column in the extended SQL descriptor area (XSQLDA).

28.1.4.4.2 TIBInputDelimitedFile.ReadParameters Method

Reads the input parameters of the XSQLDA.

Pascal

```
function ReadParameters: Boolean; override;
```

C++

```
virtual __fastcall Boolean ReadParameters();
```

Description

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA).

28.1.4.4.3 TIBInputDelimitedFile.ReadyFile Method

Prepares the output for the XSQLDA.

Pascal

```
procedure ReadyFile; override;
```

C++

```
virtual __fastcall ReadyFile();
```

Description

Call ReadyFile to prepare the output for the extended SQL descriptor area (XSQLDA).

28.1.4.5 TIBInputDelimitedFile Properties

The properties of the TIBInputDelimitedFile class are listed here.

Public Properties

	Name	Description
	ColDelimiter (see page 4046)	Sets the column delimiter for the input file.
	ReadBlanksAsNull (see page 4046)	Reads blank spaces in the input file as null characters.
	RowDelimiter (see page 4046)	Sets the row delimiter for the input file.
	SkipTitles (see page 4047)	Skips the first record of a delimited file.

28.1.4.5.1 TIBInputDelimitedFile.ColDelimiter Property

Sets the column delimiter for the input file.

Pascal

```
property ColDelimiter: string;
```

C++

```
__property UnicodeString ColDelimiter;
```

Description

Use ColDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the input file.

28.1.4.5.2 TIBInputDelimitedFile.ReadBlanksAsNull Property

Reads blank spaces in the input file as null characters.

Pascal

```
property ReadBlanksAsNull: Boolean;
```

C++

```
__property Boolean ReadBlanksAsNull;
```

Description

Set ReadBlanksAsNull to true read blank spaces as null characters in the input file.

28.1.4.5.3 TIBInputDelimitedFile.RowDelimiter Property

Sets the row delimiter for the input file.

Pascal

```
property RowDelimiter: string;
```

C++

```
__property UnicodeString RowDelimiter;
```

Description

Use RowDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the input file.

28.1.4.5.4 TIBInputDelimitedFile.SkipTitles Property

Skips the first record of a delimited file.

Pascal

```
property SkipTitles: Boolean;
```

C++

```
__property Boolean SkipTitles;
```

Description

Set SkipTitles to true to treat the first record of a delimited file as titles and skip it. Field titles are not useful in batch inputs, and this property allows you to skip them.

28.1.5 TIBInputRawFile Class

TIBInputRawFile inputs data from a raw file.

Class Hierarchy

Pascal

```
TIBInputRawFile = class(TIBBatchInput);
```

C++

```
class TIBInputRawFile : public TIBBatchInput;
```

File

IBSQL

Description

Use a TIBInputRawFile object input data from a raw file. A raw file is the equivalent to InterBase external file output. Raw files are not limited to a straight character format.

28.1.5.1 TIBInputRawFile Members

The following tables list the members exposed by TIBInputRawFile.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFile (see page 4048)	This is FFile, a member of class TIBInputRawFile.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.

	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

Public Methods

	Name	Description
	Destroy (see page 4048)	Destroys the instance of TIBInputRawFile (see page 4047).
	ReadParameters (see page 4049)	Reads the XSQLDA input parameters.
	ReadyFile (see page 4049)	Prepares the output for the XSQLDA.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.5.2 TIBInputRawFile.Destroy Destructor

Destroys the instance of TIBInputRawFile (see page 4047).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBInputRawFile();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBInputRawFile (see page 4047) object is not already freed and only then calls Destroy.

28.1.5.3 TIBInputRawFile Fields

The fields of the TIBInputRawFile class are listed here.

Protected Fields

	Name	Description
	FFile (see page 4048)	This is FFile, a member of class TIBInputRawFile.

28.1.5.3.1 TIBInputRawFile.FFile Field

Pascal

```
FFile: TFileStream;
```

C++

```
TFileStream FFile;
```

Description

This is FFile, a member of class TIBInputRawFile.

28.1.5.4 TIBInputRawFile Methods

The methods of the TIBInputRawFile class are listed here.

Public Methods

	Name	Description
	ReadParameters (see page 4049)	Reads the XSQLDA input parameters.
	ReadyFile (see page 4049)	Prepares the output for the XSQLDA.

28.1.5.4.1 TIBInputRawFile.ReadParameters Method

Reads the XSQLDA input parameters.

Pascal

```
function ReadParameters: Boolean; override;
```

C++

```
virtual __fastcall Boolean ReadParameters();
```

Description

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA).

28.1.5.4.2 TIBInputRawFile.ReadyFile Method

Prepares the output for the XSQLDA.

Pascal

```
procedure ReadyFile; override;
```

C++

```
virtual __fastcall ReadyFile();
```

Description

Call ReadyFile to prepare the output for the extended SQL descriptor area (XSQLDA).

28.1.6 TIBOutputDelimitedFile Class

TIBOutputDelimitedFile performs batch output of data to delimited files.

Class Hierarchy

Pascal

```
TIBOutputDelimitedFile = class(TIBBatchOutput);
```

C++

```
class TIBOutputDelimitedFile : public TIBBatchOutput;
```

File

IBSQL

Description

Use a TIBOutputDelimitedFile object to perform batch output of data to delimited files.

28.1.6.1 TIBOutputDelimitedFile Members

The following tables list the members exposed by TIBOutputDelimitedFile.

Protected Fields

	Name	Description
	FColDelimiter (see page 4051)	This is FColDelimiter, a member of class TIBOutputDelimitedFile.
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFile (see page 4051)	This is FFile, a member of class TIBOutputDelimitedFile.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FOutputTitles (see page 4051)	This is FOutputTitles, a member of class TIBOutputDelimitedFile.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.
	FRowDelimiter (see page 4051)	This is FRowDelimiter, a member of class TIBOutputDelimitedFile.

Public Methods

	Name	Description
	Destroy (see page 4050)	Destroys the instance of TIBOutputDelimitedFile (see page 4049).
	ReadyFile (see page 4052)	Prepares the output for the XSQLDA.
	WriteColumns (see page 4052)	Outputs the data in columns in the XSQLDA.

Public Properties

	Name	Description
	ColDelimiter (see page 4052)	Sets the column delimiter for the output file.
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	OutputTitles (see page 4053)	Outputs the titles at the top of the file.
	Params (see page 4039)	Returns the XSQLDA parameters.
	RowDelimiter (see page 4053)	Sets the row delimiter for the output file.

28.1.6.2 TIBOutputDelimitedFile.Destroy Destructor

Destroys the instance of TIBOutputDelimitedFile (see page 4049).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBOutputDelimitedFile();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBOutputDelimitedFile (see page 4049) object is not already freed and only then calls Destroy.

28.1.6.3 TIBOutputDelimitedFile Fields

The fields of the TIBOutputDelimitedFile class are listed here.

Protected Fields

	Name	Description
	FColDelimiter (see page 4051)	This is FColDelimiter, a member of class TIBOutputDelimitedFile.
	FFile (see page 4051)	This is FFile, a member of class TIBOutputDelimitedFile.

	FOutputTitles (see page 4051)	This is FOutputTitles, a member of class TIBOutputDelimitedFile.
	FRowDelimiter (see page 4051)	This is FRowDelimiter, a member of class TIBOutputDelimitedFile.

28.1.6.3.1 TIBOutputDelimitedFile.FColDelimiter Field

Pascal

```
FColDelimiter: string;
```

C++

```
UnicodeString FColDelimiter;
```

Description

This is FColDelimiter, a member of class TIBOutputDelimitedFile.

28.1.6.3.2 TIBOutputDelimitedFile.FFile Field

Pascal

```
FFile: TFileStream;
```

C++

```
TFileStream FFile;
```

Description

This is FFile, a member of class TIBOutputDelimitedFile.

28.1.6.3.3 TIBOutputDelimitedFile.FOutputTitles Field

Pascal

```
FOutputTitles: Boolean;
```

C++

```
Boolean FOutputTitles;
```

Description

This is FOutputTitles, a member of class TIBOutputDelimitedFile.

28.1.6.3.4 TIBOutputDelimitedFile.FRowDelimiter Field

Pascal

```
FRowDelimiter: string;
```

C++

```
UnicodeString FRowDelimiter;
```

Description

This is FRowDelimiter, a member of class TIBOutputDelimitedFile.

28.1.6.4 TIBOutputDelimitedFile Methods

The methods of the TIBOutputDelimitedFile class are listed here.

Public Methods

	Name	Description
	ReadyFile (see page 4052)	Prepares the output for the XSQLDA.
	WriteColumns (see page 4052)	Outputs the data in columns in the XSQLDA.

28.1.6.4.1 TIBOutputDelimitedFile.ReadyFile Method

Prepares the output for the XSQLDA.

Pascal

```
procedure ReadyFile; override;
```

C++

```
virtual __fastcall ReadyFile();
```

Description

Call Ready file to prepare the output for the extended SQL descriptor area (XSQLDA).

28.1.6.4.2 TIBOutputDelimitedFile.WriteColumns Method

Outputs the data in columns in the XSQLDA.

Pascal

```
function WriteColumns: Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteColumns();
```

Description

Set WriteColumns to true to output data in columns in the extended SQL descriptor area (XSQLDA).

28.1.6.5 TIBOutputDelimitedFile Properties

The properties of the TIBOutputDelimitedFile class are listed here.

Public Properties

	Name	Description
	ColDelimiter (see page 4052)	Sets the column delimiter for the output file.
	OutputTitles (see page 4053)	Outputs the titles at the top of the file.
	RowDelimiter (see page 4053)	Sets the row delimiter for the output file.

28.1.6.5.1 TIBOutputDelimitedFile.ColDelimiter Property

Sets the column delimiter for the output file.

Pascal

```
property ColDelimiter: string;
```

C++

```
__property UnicodeString ColDelimiter;
```

Description

Use ColDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the output file.

28.1.6.5.2 TIBOutputDelimitedFile.OutputTitles Property

Outputs the titles at the top of the file.

Pascal

```
property OutputTitles: Boolean;
```

C++

```
__property Boolean OutputTitles;
```

Description

Set OutputTitles to true to output the titles at the top of the file.

28.1.6.5.3 TIBOutputDelimitedFile.RowDelimiter Property

Sets the row delimiter for the output file.

Pascal

```
property RowDelimiter: string;
```

C++

```
__property UnicodeString RowDelimiter;
```

Description

Use RowDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the output file.

28.1.7 TIBOutputRawFile Class

TIBOutputRawFile outputs data to a raw file format.

Class Hierarchy

Pascal

```
TIBOutputRawFile = class(TIBBatchOutput);
```

C++

```
class TIBOutputRawFile : public TIBBatchOutput;
```

File

IBSQL

Description

Use a TIBOutputRawFile object to output data from a raw file. A raw file is the equivalent to InterBase external file output. Raw files are not limited to a straight character format.

28.1.7.1 TIBOutputRawFile Members

The following tables list the members exposed by TIBOutputRawFile.

Protected Fields

	Name	Description
	FColumns (see page 4037)	This is FColumns, a member of class TIBBatch.
	FFile (see page 4054)	This is FFile, a member of class TIBOutputRawFile.
	FFilename (see page 4037)	This is FFilename, a member of class TIBBatch.
	FParams (see page 4038)	This is FParams, a member of class TIBBatch.

Public Methods

	Name	Description
	Destroy (see page 4054)	Destroys the instance of TIBOutputRawFile (see page 4053).
	ReadyFile (see page 4055)	Prepares the output for the XSQLDA.
	WriteColumns (see page 4055)	Outputs the data in columns in the XSQLDA.

Public Properties

	Name	Description
	Columns (see page 4038)	Returns the XSQLDA columns.
	Filename (see page 4039)	Sets or displays the name of the external file.
	Params (see page 4039)	Returns the XSQLDA parameters.

28.1.7.2 TIBOutputRawFile.Destroy Destructor

Destroys the instance of TIBOutputRawFile (see page 4053).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBOutputRawFile();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBOutputRawFile (see page 4053) object is not already freed and only then calls Destroy.

28.1.7.3 TIBOutputRawFile Fields

The fields of the TIBOutputRawFile class are listed here.

Protected Fields

	Name	Description
	FFile (see page 4054)	This is FFile, a member of class TIBOutputRawFile.

28.1.7.3.1 TIBOutputRawFile.FFile Field

Pascal

```
FFile: TFileStream;
```

C++

```
TFileStream FFile;
```

Description

This is FFile, a member of class TIBOutputRawFile.

28.1.7.4 TIBOutputRawFile Methods

The methods of the TIBOutputRawFile class are listed here.

Public Methods

	Name	Description
	ReadyFile (see page 4055)	Prepares the output for the XSQLDA.
	WriteColumns (see page 4055)	Outputs the data in columns in the XSQLDA.

28.1.7.4.1 TIBOutputRawFile.ReadyFile Method

Prepares the output for the XSQLDA.

Pascal

```
procedure ReadyFile; override;
```

C++

```
virtual __fastcall ReadyFile();
```

Description

Call Ready file to prepare the output for the extended SQL descriptor area (XSQLDA).

28.1.7.4.2 TIBOutputRawFile.WriteColumns Method

Outputs the data in columns in the XSQLDA.

Pascal

```
function WriteColumns: Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteColumns();
```

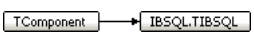
Description

Call WriteColumns to output data in columns in the extended SQL descriptor area (XSQLDA).

28.1.8 TIBSQL Class

TIBSQL provides an object for executing an InterBase SQL (see page 4076) statement with minimal overhead.

Class Hierarchy



Pascal

```
TIBSQL = class(TComponent);
```

C++

```
class TIBSQL : public TComponent;
```

File

IBSQL

Description

Use a TIBSQL object to execute an InterBase SQL (see page 4076) statement with minimal overhead. TIBSQL has no standard interface to data-aware controls and is unidirectional.

See Also

Getting Started with InterBase Express

28.1.8.1 TIBSQL Members

The following tables list the members exposed by TIBSQL.

Protected Fields

	Name	Description
	FBase (see page 4059)	This is FBase, a member of class TIBSQL.
	FBOF (see page 4059)	This is FBOF, a member of class TIBSQL.
	FCursor (see page 4059)	This is FCursor, a member of class TIBSQL.
	FEOF (see page 4059)	This is FEOF, a member of class TIBSQL.
	FGenerateParamNames (see page 4060)	This is FGenerateParamNames, a member of class TIBSQL.
	FGoToFirstRecordOnExecute (see page 4060)	This is FGoToFirstRecordOnExecute, a member of class TIBSQL.
	FHandle (see page 4060)	This is FHandle, a member of class TIBSQL.
	FOnSQLChanging (see page 4060)	This is FOnSQLChanging, a member of class TIBSQL.
	FOpen (see page 4061)	This is FOpen, a member of class TIBSQL.
	FParamCheck (see page 4061)	This is FParamCheck, a member of class TIBSQL.
	FPrepared (see page 4061)	This is FPrepared, a member of class TIBSQL.
	FProcessedSQL (see page 4061)	This is FProcessedSQL, a member of class TIBSQL.
	FRecordCount (see page 4061)	This is FRecordCount, a member of class TIBSQL.
	FSQL (see page 4062)	This is FSQL, a member of class TIBSQL.
	FSQLParams (see page 4062)	This is FSQLParams, a member of class TIBSQL.
	FSQLRecord (see page 4062)	This is FSQLRecord, a member of class TIBSQL.
	FSQLType (see page 4062)	This is FSQLType, a member of class TIBSQL.

Protected Methods

	Name	Description
	BeforeTransactionEnd (see page 4064)	This is BeforeTransactionEnd, a member of class TIBSQL.
	DoBeforeDatabaseDisconnect (see page 4066)	This is DoBeforeDatabaseDisconnect, a member of class TIBSQL.
	GetDatabase (see page 4067)	This is GetDatabase, a member of class TIBSQL.
	GetDBHandle (see page 4067)	This is GetDBHandle, a member of class TIBSQL.
	GetEOF (see page 4067)	This is GetEOF, a member of class TIBSQL.
	GetFieldIndex (see page 4067)	This is GetFieldIndex, a member of class TIBSQL.
	GetFields (see page 4067)	This is GetFields, a member of class TIBSQL.
	GetPlan (see page 4068)	This is GetPlan, a member of class TIBSQL.
	GetRecordCount (see page 4068)	This is GetRecordCount, a member of class TIBSQL.
	GetRowsAffected (see page 4068)	This is GetRowsAffected, a member of class TIBSQL.
	GetSQLParams (see page 4068)	This is GetSQLParams, a member of class TIBSQL.

	GetTransaction (see page 4069)	This is GetTransaction, a member of class TIBSQL.
	GetTRHandle (see page 4069)	This is GetTRHandle, a member of class TIBSQL.
	PreprocessSQL (see page 4070)	This is PreprocessSQL, a member of class TIBSQL.
	SetDatabase (see page 4070)	This is SetDatabase, a member of class TIBSQL.
	SetSQL (see page 4070)	This is SetSQL, a member of class TIBSQL.
	SetTransaction (see page 4071)	This is SetTransaction, a member of class TIBSQL.
	SQLChanging (see page 4071)	This is SQLChanging, a member of class TIBSQL.

Public Methods

	Name	Description
	BatchInput (see page 4063)	Executes the parameterized query in SQL (see page 4076) for input in the referenced input object.
	BatchOutput (see page 4064)	Outputs the selected query in SQL (see page 4076) to the referenced OutputObject.
	Call (see page 4064)	Returns an error message based on the error code.
	CheckClosed (see page 4064)	raises an exception if the query is not closed.
	CheckOpen (see page 4065)	raises an exception if the query is not open.
	CheckValidStatement (see page 4065)	raises an exception if the query does not have a valid statement.
	Close (see page 4065)	Closes the query.
V	Create (see page 4058)	Creates an instance of a TIBSQL (see page 4055) component.
	Current (see page 4065)	Returns an extended SQL (see page 4076) descriptor for the current record.
V	Destroy (see page 4058)	Frees all resources associated with this instance.
	ExecQuery (see page 4066)	Executes an SQL (see page 4076) query.
	FieldByName (see page 4066)	Returns the XSQLVAR fields by name.
	FreeHandle (see page 4066)	Frees InterBase resources associated with the query.
	GetUniqueRelationName (see page 4069)	Gets the unique relation name.
	Next (see page 4069)	Returns an extended SQL (see page 4076) descriptor for the next record.
	ParamByName (see page 4069)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 4070)	Prepares a query for execution.

Public Properties

	Name	Description
R	Bof (see page 4072)	Indicates whether or not a query is at the beginning of the dataset.
R	DBHandle (see page 4072)	Specifies the database handle for the query.
R	Eof (see page 4072)	Indicates whether or not a query is at the end of the dataset.
R	FieldIndex (see page 4073)	Returns the index of the named field.
R	Fields (see page 4073)	Returns the XSQLVAR fields.
	GenerateParamNames (see page 4073)	Generates a list of parameter names for the query.
R	Handle (see page 4074)	Specifies the handle for the query.
R	Open (see page 4074)	Determines if the dataset is open.
R	Params (see page 4075)	Returns the XSQLDA parameters.
R	Plan (see page 4075)	Returns the plan for the query.
R	Prepared (see page 4075)	Indicates whether or not the query has been prepared.
R	RecordCount (see page 4075)	Returns the current count of records from the query.
R	RowsAffected (see page 4076)	Returns the number of rows affected.
R	SQLType (see page 4076)	Returns the type of query to be executed.
R	TRHandle (see page 4077)	Specifies the transaction handle for the query.
R	UniqueRelationName (see page 4077)	Indicates the unique relation name.

Published Events

	Name	Description
	OnSQLChanging (see page 4077)	Occurs when the SQL (see page 4076) query is being modified.

Published Properties

	Name	Description
	Database (see page 4072)	Sets or returns the database associated with the query.
	GoToFirstRecordOnExecute (see page 4074)	Goes to the first record in the result set upon opening it.
	ParamCheck (see page 4074)	Specifies whether the parameter list for an SQL (see page 4076) query is regenerated if the SQL (see page 4076) property changes at runtime.
	SQL (see page 4076)	Sets the SQL query to be executed.
	Transaction (see page 4077)	Sets or returns the transaction to be used by the query.

28.1.8.2 TIBSQL.Create Constructor

Creates an instance of a TIBSQL (see page 4055) component.

Pascal

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

C++

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

Description

Call (see page 4064) Create to create an instance of a TIBSQL (see page 4055) component.

28.1.8.3 TIBSQL.Destroy Destructor

Frees all resources associated with this instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBSQL();
```

Description

Do not call Destroy directly in an application. Usually destruction of objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not already freed before calling Destroy.

28.1.8.4 TIBSQL Fields

The fields of the TIBSQL class are listed here.

Protected Fields

	Name	Description
	FBase (see page 4059)	This is FBase, a member of class TIBSQL.
	FBOF (see page 4059)	This is FBOF, a member of class TIBSQL.
	FCursor (see page 4059)	This is FCursor, a member of class TIBSQL.
	FEOF (see page 4059)	This is FEOF, a member of class TIBSQL.
	FGenerateParamNames (see page 4060)	This is FGenerateParamNames, a member of class TIBSQL.
	FGoToFirstRecordOnExecute (see page 4060)	This is FGoToFirstRecordOnExecute, a member of class TIBSQL.
	FHandle (see page 4060)	This is FHandle, a member of class TIBSQL.
	FOnSQLChanging (see page 4060)	This is FOnSQLChanging, a member of class TIBSQL.

	FOpen (see page 4061)	This is FOpen, a member of class TIBSQL.
	FParamCheck (see page 4061)	This is FParamCheck, a member of class TIBSQL.
	FPrepared (see page 4061)	This is FPrepared, a member of class TIBSQL.
	FProcessedSQL (see page 4061)	This is FProcessedSQL, a member of class TIBSQL.
	FRecordCount (see page 4061)	This is FRecordCount, a member of class TIBSQL.
	FSQL (see page 4062)	This is FSQL, a member of class TIBSQL.
	FSQLParams (see page 4062)	This is FSQLParams, a member of class TIBSQL.
	FSQLRecord (see page 4062)	This is FSQLRecord, a member of class TIBSQL.
	FSQLType (see page 4062)	This is FSQLType, a member of class TIBSQL.

28.1.8.4.1 TIBSQL.FBase Field

Pascal

```
FBase: TIBBase;
```

C++

```
TIBBase FBase;
```

Description

This is FBase, a member of class TIBSQL.

28.1.8.4.2 TIBSQL.FBOF Field

Pascal

```
FBOF: Boolean;
```

C++

```
Boolean FBOF;
```

Description

This is FBOF, a member of class TIBSQL.

28.1.8.4.3 TIBSQL.FCursor Field

Pascal

```
FCursor: String;
```

C++

```
UnicodeString FCursor;
```

Description

This is FCursor, a member of class TIBSQL.

28.1.8.4.4 TIBSQL.FEOF Field

Pascal

```
FEOF: Boolean;
```

C++

```
Boolean FEOF;
```

Description

This is FEOF, a member of class TIBSQL.

28.1.8.4.5 TIBSQL.FGenerateParamNames Field

Pascal

```
FGenerateParamNames: Boolean;
```

C++

```
Boolean FGenerateParamNames;
```

Description

This is FGenerateParamNames, a member of class TIBSQL.

28.1.8.4.6 TIBSQL.FGoToFirstRecordOnExecute Field

Pascal

```
FGoToFirstRecordOnExecute: Boolean;
```

C++

```
Boolean FGoToFirstRecordOnExecute;
```

Description

This is FGoToFirstRecordOnExecute, a member of class TIBSQL.

28.1.8.4.7 TIBSQL.FHandle Field

Pascal

```
FHandle: TISC_STMT_HANDLE;
```

C++

```
TISC_STMT_HANDLE FHandle;
```

Description

This is FHandle, a member of class TIBSQL.

28.1.8.4.8 TIBSQL.FOnSQLChanging Field

Pascal

```
FOnSQLChanging: TNotifyEvent;
```

C++

```
TNotifyEvent FOnSQLChanging;
```

Description

This is FOnSQLChanging, a member of class TIBSQL.

28.1.8.4.9 TIBSQL.FOpen Field

Pascal

```
FOpen: Boolean;
```

C++

```
Boolean FOpen;
```

Description

This is FOpen, a member of class TIBSQL.

28.1.8.4.10 TIBSQL.FParamCheck Field

Pascal

```
FParamCheck: Boolean;
```

C++

```
Boolean FParamCheck;
```

Description

This is FParamCheck, a member of class TIBSQL.

28.1.8.4.11 TIBSQL.FPrepared Field

Pascal

```
FPrepared: Boolean;
```

C++

```
Boolean FPrepared;
```

Description

This is FPrepared, a member of class TIBSQL.

28.1.8.4.12 TIBSQL.FProcessedSQL Field

Pascal

```
FProcessedSQL: TStrings;
```

C++

```
TStrings FProcessedSQL;
```

Description

This is FProcessedSQL, a member of class TIBSQL.

28.1.8.4.13 TIBSQL.FRecordCount Field

Pascal

```
FRecordCount: Integer;
```

C++

```
int FRecordCount;
```

Description

This is FRecordCount, a member of class TIBSQL.

28.1.8.4.14 TIBSQL.FSQL Field

Pascal

```
FSQL: TStrings;
```

C++

```
TStrings FSQL;
```

Description

This is FSQL, a member of class TIBSQL.

28.1.8.4.15 TIBSQL.FSQLParams Field

Pascal

```
FSQLParams: TIBXSQLDA;
```

C++

```
TIBXSQLDA FSQLParams;
```

Description

This is FSQLParams, a member of class TIBSQL.

28.1.8.4.16 TIBSQL.FSQLRecord Field

Pascal

```
FSQLRecord: TIBXSQLDA;
```

C++

```
TIBXSQLDA FSQLRecord;
```

Description

This is FSQLRecord, a member of class TIBSQL.

28.1.8.4.17 TIBSQL.FSQLType Field

Pascal

```
FSQLType: TIBSQLTypes;
```

C++

```
TIBSQLTypes FSQLType;
```

Description

This is FSQLType, a member of class TIBSQL.

28.1.8.5 TIBSQL Methods

The methods of the TIBSQL class are listed here.

Protected Methods

	Name	Description
	BeforeTransactionEnd (see page 4064)	This is BeforeTransactionEnd, a member of class TIBSQL.
	DoBeforeDatabaseDisconnect (see page 4066)	This is DoBeforeDatabaseDisconnect, a member of class TIBSQL.
	GetDatabase (see page 4067)	This is GetDatabase, a member of class TIBSQL.
	GetDBHandle (see page 4067)	This is GetDBHandle, a member of class TIBSQL.
	GetEOF (see page 4067)	This is GetEOF, a member of class TIBSQL.
	GetFieldIndex (see page 4067)	This is GetFieldIndex, a member of class TIBSQL.
	GetFields (see page 4067)	This is GetFields, a member of class TIBSQL.
	GetPlan (see page 4068)	This is GetPlan, a member of class TIBSQL.
	GetRecordCount (see page 4068)	This is GetRecordCount, a member of class TIBSQL.
	GetRowsAffected (see page 4068)	This is GetRowsAffected, a member of class TIBSQL.
	GetSQLParams (see page 4068)	This is GetSQLParams, a member of class TIBSQL.
	GetTransaction (see page 4069)	This is GetTransaction, a member of class TIBSQL.
	GetTRHandle (see page 4069)	This is GetTRHandle, a member of class TIBSQL.
	PreprocessSQL (see page 4070)	This is PreprocessSQL, a member of class TIBSQL.
	SetDatabase (see page 4070)	This is SetDatabase, a member of class TIBSQL.
	SetSQL (see page 4070)	This is SetSQL, a member of class TIBSQL.
	SetTransaction (see page 4071)	This is SetTransaction, a member of class TIBSQL.
	SQLChanging (see page 4071)	This is SQLChanging, a member of class TIBSQL.

Public Methods

	Name	Description
	BatchInput (see page 4063)	Executes the parameterized query in SQL (see page 4076) for input in the referenced input object.
	BatchOutput (see page 4064)	Outputs the selected query in SQL (see page 4076) to the referenced OutputObject.
	Call (see page 4064)	Returns an error message based on the error code.
	CheckClosed (see page 4064)	raises an exception if the query is not closed.
	CheckOpen (see page 4065)	raises an exception if the query is not open.
	CheckValidStatement (see page 4065)	raises an exception if the query does not have a valid statement.
	Close (see page 4065)	Closes the query.
	Current (see page 4065)	Returns an extended SQL (see page 4076) descriptor for the current record.
	ExecQuery (see page 4066)	Executes an SQL (see page 4076) query.
	FieldByName (see page 4066)	Returns the XSQLVAR fields by name.
	FreeHandle (see page 4066)	Frees InterBase resources associated with the query.
	GetUniqueRelationName (see page 4069)	Gets the unique relation name.
	Next (see page 4069)	Returns an extended SQL (see page 4076) descriptor for the next record.
	ParamByName (see page 4069)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 4070)	Prepares a query for execution.

28.1.8.5.1 TIBSQL.BatchInput Method

Executes the parameterized query in SQL (see page 4076) for input in the referenced input object.

Pascal

```
procedure BatchInput(InputObject: TIBBatchInput);
```

C++

```
__fastcall BatchInput(TIBBatchInput InputObject);
```

Description

Call (see page 4064) BatchInput to execute the parameterized query in SQL (see page 4076) for input in the referenced input object.

28.1.8.5.2 TIBSQL.BatchOutput Method

Outputs the selected query in SQL (see page 4076) to the referenced OutputObject.

Pascal

```
procedure BatchOutput(OutputObject: TIBBatchOutput);
```

C++

```
__fastcall BatchOutput(TIBBatchOutput OutputObject);
```

Description

Call (see page 4064) BatchOutput to output the selected query in SQL (see page 4076) to the referenced OutputObject.

28.1.8.5.3 TIBSQL.BeforeTransactionEnd Method

Pascal

```
procedure BeforeTransactionEnd(Sender: TObject);
```

C++

```
__fastcall BeforeTransactionEnd(TObject * Sender);
```

Description

This is BeforeTransactionEnd, a member of class TIBSQL.

28.1.8.5.4 TIBSQL.Call Method

Returns an error message based on the error code.

Pascal

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

C++

```
__fastcall ISC_STATUS Call(ISC_STATUS ErrCode, Boolean RaiseError);
```

Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

28.1.8.5.5 TIBSQL.CheckClosed Method

raises an exception if the query is not closed.

Pascal

```
procedure CheckClosed;
```

C++

```
__fastcall CheckClosed();
```

Description

Call (see page 4064) CheckClosed to raise an exception if the query is not closed.

28.1.8.5.6 TIBSQL.CheckOpen Method

raises an exception if the query is not open.

Pascal

```
procedure CheckOpen;
```

C++

```
__fastcall CheckOpen();
```

Description

Call (see page 4064) CheckOpen to raise an exception if the query is closed.

28.1.8.5.7 TIBSQL.CheckValidStatement Method

raises an exception if the query does not have a valid statement.

Pascal

```
procedure CheckValidStatement;
```

C++

```
__fastcall CheckValidStatement();
```

Description

Call (see page 4064) CheckValidStatement to raise an exception if the query does not have a valid statement.

28.1.8.5.8 TIBSQL.Close Method

Closes the query.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call (see page 4064) Close to close the query.

28.1.8.5.9 TIBSQL.Current Method

Returns an extended SQL (see page 4076) descriptor for the current record.

Pascal

```
function Current: TIBXSQLDA;
```

C++

```
__fastcall TIBXSQLDA Current();
```

Description

Call (see page 4064) Current to get an extended SQL (see page 4076) descriptor for the current record.

28.1.8.5.10 TIBSQL.DoBeforeDatabaseDisconnect Method

Pascal

```
procedure DoBeforeDatabaseDisconnect(Sender: TObject);
```

C++

```
__fastcall DoBeforeDatabaseDisconnect(TObject * Sender);
```

Description

This is DoBeforeDatabaseDisconnect, a member of class TIBSQL.

28.1.8.5.11 TIBSQL.ExecQuery Method

Executes an SQL (see page 4076) query.

Pascal

```
procedure ExecQuery;
```

C++

```
__fastcall ExecQuery();
```

Description

Note: Call (see page 4064) ExecQuery to execute the SQL (see page 4076) query.

28.1.8.5.12 TIBSQL.FieldByName Method

Returns the XSQLVAR fields by name.

Pascal

```
function FieldByName(FieldName: String): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR FieldByName(UnicodeString FieldName);
```

Description

Use the FieldByName method to return the XSQLVAR fields by name.

28.1.8.5.13 TIBSQL.FreeHandle Method

Frees InterBase resources associated with the query.

Pascal

```
procedure FreeHandle;
```

C++

```
__fastcall FreeHandle();
```

Description

Call (see page 4064) FreeHandle to free the InterBase resources associated with the query.

28.1.8.5.14 TIBSQL.GetDatabase Method

Pascal

```
function GetDatabase: TIBDatabase;
```

C++

```
__fastcall TIBDatabase GetDatabase();
```

Description

This is GetDatabase, a member of class TIBSQL.

28.1.8.5.15 TIBSQL.GetDBHandle Method

Pascal

```
function GetDBHandle: PISC_DB_HANDLE;
```

C++

```
__fastcall PISC_DB_HANDLE GetDBHandle();
```

Description

This is GetDBHandle, a member of class TIBSQL.

28.1.8.5.16 TIBSQL.GetEOF Method

Pascal

```
function GetEOF: Boolean;
```

C++

```
__fastcall Boolean GetEOF();
```

Description

This is GetEOF, a member of class TIBSQL.

28.1.8.5.17 TIBSQL.GetFieldIndex Method

Pascal

```
function GetFieldIndex(FieldName: String): Integer;
```

C++

```
__fastcall int GetFieldIndex(UnicodeString FieldName);
```

Description

This is GetFieldIndex, a member of class TIBSQL.

28.1.8.5.18 TIBSQL.GetFields Method

Pascal

```
function GetFields(const Idx: Integer): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR GetFields(const int Idx);
```

Description

This is GetFields, a member of class TIBSQL.

28.1.8.5.19 TIBSQL.GetPlan Method

Pascal

```
function GetPlan: String;
```

C++

```
__fastcall UnicodeString GetPlan();
```

Description

This is GetPlan, a member of class TIBSQL.

28.1.8.5.20 TIBSQL.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer;
```

C++

```
__fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TIBSQL.

28.1.8.5.21 TIBSQL.GetRowsAffected Method

Pascal

```
function GetRowsAffected: Integer;
```

C++

```
__fastcall int GetRowsAffected();
```

Description

This is GetRowsAffected, a member of class TIBSQL.

28.1.8.5.22 TIBSQL.GetSQLParams Method

Pascal

```
function GetSQLParams: TIBXSQLDA;
```

C++

```
__fastcall TIBXSQLDA GetSQLParams();
```

Description

This is GetSQLParams, a member of class TIBSQL.

28.1.8.5.23 TIBSQL.GetTransaction Method

Pascal

```
function GetTransaction: TIBTransaction;
```

C++

```
__fastcall TIBTransaction GetTransaction();
```

Description

This is GetTransaction, a member of class TIBSQL.

28.1.8.5.24 TIBSQL.GetTRHandle Method

Pascal

```
function GetTRHandle: PISC_TR_HANDLE;
```

C++

```
__fastcall PISC_TR_HANDLE GetTRHandle();
```

Description

This is GetTRHandle, a member of class TIBSQL.

28.1.8.5.25 TIBSQL.GetUniqueRelationName Method

Gets the unique relation name.

Pascal

```
function GetUniqueRelationName: String;
```

C++

```
__fastcall UnicodeString GetUniqueRelationName();
```

Description

Note: Call (see page 4064) GetUniqueRelationName to get the unique relation name of the query.

28.1.8.5.26 TIBSQL.Next Method

Returns an extended SQL (see page 4076) descriptor for the next record.

Pascal

```
function Next: TIBXSQLDA;
```

C++

```
__fastcall TIBXSQLDA Next();
```

Description

Call (see page 4064) Next to get an extended SQL (see page 4076) descriptor for the next record.

28.1.8.5.27 TIBSQL.ParamByName Method

Accesses parameter information based on a specified parameter name.

Pascal

```
function ParamByName(Idx: String): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR ParamByName(UnicodeString Idx);
```

Description

Call (see page 4064) ParamByName to set or use parameter information for a specific parameter based on its name.

Idx is the name of the parameter for which to retrieve information.

28.1.8.5.28 TIBSQL.Prepare Method

Prepares a query for execution.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call (see page 4064) Prepare to prepare a query for execution.

28.1.8.5.29 TIBSQL.PreprocessSQL Method

Pascal

```
procedure PreprocessSQL;
```

C++

```
__fastcall PreprocessSQL();
```

Description

This is PreprocessSQL, a member of class TIBSQL.

28.1.8.5.30 TIBSQL.SetDatabase Method

Pascal

```
procedure SetDatabase(Value: TIBDatabase);
```

C++

```
__fastcall SetDatabase(TIBDatabase Value);
```

Description

This is SetDatabase, a member of class TIBSQL.

28.1.8.5.31 TIBSQL.SetSQL Method

Pascal

```
procedure SetSQL(Value: TStrings);
```

C++

```
__fastcall SetSQL(TStrings Value);
```

Description

This is SetSQL, a member of class TIBSQL.

28.1.8.5.32 TIBSQL.SetTransaction Method

Pascal

```
procedure SetTransaction(Value: TIBTransaction);
```

C++

```
__fastcall SetTransaction(TIBTransaction Value);
```

Description

This is SetTransaction, a member of class TIBSQL.

28.1.8.5.33 TIBSQL.SQLChanging Method

Pascal

```
procedure SQLChanging(Sender: TObject);
```

C++

```
__fastcall SQLChanging(TObject * Sender);
```

Description

This is SQLChanging, a member of class TIBSQL.

28.1.8.6 TIBSQL Properties

The properties of the TIBSQL class are listed here.

Public Properties

	Name	Description
R	Bof (see page 4072)	Indicates whether or not a query is at the beginning of the dataset.
R	DBHandle (see page 4072)	Specifies the database handle for the query.
R	Eof (see page 4072)	Indicates whether or not a query is at the end of the dataset.
R	FieldIndex (see page 4073)	Returns the index of the named field.
R	Fields (see page 4073)	Returns the XSQLVAR fields.
	GenerateParamNames (see page 4073)	Generates a list of parameter names for the query.
R	Handle (see page 4074)	Specifies the handle for the query.
R	Open (see page 4074)	Determines if the dataset is open.
R	Params (see page 4075)	Returns the XSQLDA parameters.
R	Plan (see page 4075)	Returns the plan for the query.
R	Prepared (see page 4075)	Indicates whether or not the query has been prepared.
R	RecordCount (see page 4075)	Returns the current count of records from the query.
R	RowsAffected (see page 4076)	Returns the number of rows affected.
R	SQLType (see page 4076)	Returns the type of query to be executed.
R	TRHandle (see page 4077)	Specifies the transaction handle for the query.
R	UniqueRelationName (see page 4077)	Indicates the unique relation name.

Published Properties

	Name	Description
	Database (see page 4072)	Sets or returns the database associated with the query.
	GoToFirstRecordOnExecute (see page 4074)	Goes to the first record in the result set upon opening it.
	ParamCheck (see page 4074)	Specifies whether the parameter list for an SQL (see page 4076) query is regenerated if the SQL (see page 4076) property changes at runtime.
	SQL (see page 4076)	Sets the SQL query to be executed.
	Transaction (see page 4077)	Sets or returns the transaction to be used by the query.

28.1.8.6.1 TIBSQL.Bof Property

Indicates whether or not a query is at the beginning of the dataset.

Pascal

```
property Bof: Boolean;
```

C++

```
__property Boolean Bof;
```

Description

Use the BOF property to determine whether or not a query is at the beginning of the dataset.

28.1.8.6.2 TIBSQL.Database Property

Sets or returns the database associated with the query.

Pascal

```
property Database: TIBDatabase;
```

C++

```
__property TIBDatabase Database;
```

Description

Use the Database property to set or return the database associated with the query.

28.1.8.6.3 TIBSQL.DBHandle Property

Specifies the database handle for the query.

Pascal

```
property DBHandle: PISC_DB_HANDLE;
```

C++

```
__property PISC_DB_HANDLE DBHandle;
```

Description

Use the DBHandle property to return the database handle for the query.

28.1.8.6.4 TIBSQL.Eof Property

Indicates whether or not a query is at the end of the dataset.

Pascal

```
property Eof: Boolean;
```

C++

```
__property Boolean Eof;
```

Description

Use the EOF property to determine whether or not a query is at the end of the dataset and whether or not the query returned any result rows.

In addition, if this property is true immediately after the query is opened, then it means that the dataset is empty.

28.1.8.6.5 TIBSQL.FieldIndex Property

Returns the index of the named field.

Pascal

```
property FieldIndex [FieldName: String]: Integer;
```

C++

```
__property int FieldIndex[UnicodeString FieldName];
```

Description

Use the FieldIndex property to return the index of the name field.

28.1.8.6.6 TIBSQL.Fields Property

Returns the XSQLVAR fields.

Pascal

```
property Fields [const Idx: Integer]: TIBXSQLVAR;
```

C++

```
__property TIBXSQLVAR Fields[int const Idx];
```

Description

Use the Fields property to return the XSQLVAR fields.

28.1.8.6.7 TIBSQL.GenerateParamNames Property

Generates a list of parameter names for the query.

Pascal

```
property GenerateParamNames: Boolean;
```

C++

```
__property Boolean GenerateParamNames;
```

Description

Set GenerateParamNames to true to have the query generate a list of parameter names.

28.1.8.6.8 TIBSQL.GoToFirstRecordOnExecute Property

Goes to the first record in the result set upon opening it.

Pascal

```
property GoToFirstRecordOnExecute: Boolean;
```

C++

```
__property Boolean GoToFirstRecordOnExecute;
```

Description

Use the GoToFirstRecordOnExecute property to go to the first record in a result set upon opening it. By default, this property is set to true. GoToFirstRecordOnExecute exists primarily for use in TIBDataSet, which sets this value to false for its internal TIBSQLs.

28.1.8.6.9 TIBSQL.Handle Property

Specifies the handle for the query.

Pascal

```
property Handle: TISC_STMT_HANDLE;
```

C++

```
__property TISC_STMT_HANDLE Handle;
```

Description

Use the Handle property to get the query handle.

28.1.8.6.10 TIBSQL.Open Property

Determines if the dataset is open.

Pascal

```
property Open: Boolean;
```

C++

```
__property Boolean Open;
```

Description

Use the Open property to determine if the dataset is open.

28.1.8.6.11 TIBSQL.ParamCheck Property

Specifies whether the parameter list for an SQL (see page 4076) query is regenerated if the SQL (see page 4076) property changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the TIBSQL (see page 4055) query. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TIBSQL (see page 4055) query executing the DDL statement.

An application that does not use parameterized queries may choose to set ParamCheck to false, but otherwise ParamCheck should be true.

28.1.8.6.12 TIBSQL.Params Property

Returns the XSQLDA parameters.

Pascal

```
property Params: TIBXSQLDA;
```

C++

```
__property TIBXSQLDA Params;
```

Description

Use the Params property to return the XSQLDA parameters.

28.1.8.6.13 TIBSQL.Plan Property

Returns the plan for the query.

Pascal

```
property Plan: String;
```

C++

```
__property UnicodeString Plan;
```

Description

Use the Plan property to view the query plan once the query has been prepared.

28.1.8.6.14 TIBSQL.Prepared Property

Indicates whether or not the query has been prepared.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Use the Prepared property to determine whether or not a query has yet been prepared.

28.1.8.6.15 TIBSQL.RecordCount Property

Returns the current count of records from the query.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Use the RecordCount property to see how many records are returned by a query. If the result set is to return 100 rows, RecordCount will only be 100 after all the records have been visited. That is, after looking at the first record, RecordCount is 1, and so forth.

28.1.8.6.16 TIBSQL.RowsAffected Property

Returns the number of rows affected.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

Use the RowsAffected property to return the number of rows affected by the query. This property is useful for INSERT, DELETE, and UPDATE statements.

28.1.8.6.17 TIBSQL.SQL Property

Sets the SQL query to be executed.

Pascal

```
property SQL: TStrings;
```

C++

```
__property TStrings SQL;
```

Description

Use the SQL property to write or view the SQL query to be executed.

28.1.8.6.18 TIBSQL.SQLType Property

Returns the type of query to be executed.

Pascal

```
property SQLType: TIBSQLTypes;
```

C++

```
__property TIBSQLTypes SQLType;
```

Description

Use the SQLType to determine the type of query to be executed.

28.1.8.6.19 TIBSQL.Transaction Property

Sets or returns the transaction to be used by the query.

Pascal

```
property Transaction: TIBTransaction;
```

C++

```
__property TIBTransaction Transaction;
```

Description

Use the Transaction property to set or return the transaction to be used by the query.

28.1.8.6.20 TIBSQL.TRHandle Property

Specifies the transaction handle for the query.

Pascal

```
property TRHandle: PISC_TR_HANDLE;
```

C++

```
__property PISC_TR_HANDLE TRHandle;
```

Description

Use the TRHandle property to return the transaction handle for the query.

28.1.8.6.21 TIBSQL.UniqueRelationName Property

Indicates the unique relation name.

Pascal

```
property UniqueRelationName: String;
```

C++

```
__property UnicodeString UniqueRelationName;
```

Description

Use the UniqueRelationName property to indicate the unique relation name for a query that involves only one base table.

28.1.8.7 TIBSQL Events

The events of the TIBSQL class are listed here.

Published Events

	Name	Description
	OnSQLChanging (see page 4077)	Occurs when the SQL (see page 4076) query is being modified.

28.1.8.7.1 TIBSQL.OnSQLChanging Event

Occurs when the SQL (see page 4076) query is being modified.

Pascal

```
property OnSQLChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSQLChanging;
```

Description

Write an OnSQLChanging event handler to take specific actions when a query is being modified. If an exception is raised in this event, the query is not changed.

28.1.9 TIBXSQLDA Class

TIBXSQLDA provides properties and methods for use with the IBSQL (see page 4035) component.

Class Hierarchy

Pascal

```
TIBXSQLDA = class(TObject);
```

C++

```
class TIBXSQLDA : public TObject;
```

File

IBSQL

Description

Use a TIBXSQLDA object to provide properties and methods for use with the IBSQL (see page 4035) component.

All Dynamic SQL (DSQL) applications must declare one or more extended descriptor areas (XSQLDAs). The XSQLDA is a host language data structure that DSQL uses to transport data to or from a database when processing an SQL statement string. There are two types of XSQLDAs: input descriptors and output descriptors. Both input and output descriptors are implemented using the XSQLDA structure.

28.1.9.1 TIBXSQLDA Members

The following tables list the members exposed by TIBXSQLDA.

Protected Fields

	Name	Description
	FCount (see page 4080)	This is FCount, a member of class TIBXSQLDA.
	FNames (see page 4080)	This is FNames, a member of class TIBXSQLDA.
	FSize (see page 4080)	This is FSize, a member of class TIBXSQLDA.
	FSQL (see page 4081)	This is FSQL, a member of class TIBXSQLDA.
	FUniqueRelationName (see page 4081)	This is FUniqueRelationName, a member of class TIBXSQLDA.
	FXSQLDA (see page 4081)	This is FXSQLDA, a member of class TIBXSQLDA.
	FXSQLVARs (see page 4081)	This is FXSQLVARs, a member of class TIBXSQLDA.

Protected Methods

	Name	Description
	GetModified (see page 4082)	This is GetModified, a member of class TIBXSQLDA.
	GetNames (see page 4083)	This is GetNames, a member of class TIBXSQLDA.
	GetRecordSize (see page 4083)	This is GetRecordSize, a member of class TIBXSQLDA.
	GetXSQLDA (see page 4083)	This is GetXSQLDA, a member of class TIBXSQLDA.
	GetXSQLVAR (see page 4083)	This is GetXSQLVAR, a member of class TIBXSQLDA.
	GetXSQLVARByName (see page 4083)	This is GetXSQLVARByName, a member of class TIBXSQLDA.
	Initialize (see page 4084)	This is Initialize, a member of class TIBXSQLDA.
	SetCount (see page 4084)	This is SetCount, a member of class TIBXSQLDA.

Public Methods

	Name	Description
	AddName (see page 4082)	Adds a name to the XSQLDA structure.
	ByName (see page 4082)	Returns the XSQLVAR fields by name.
	Create (see page 4079)	Creates an instance of an XSQLDA structure.
V	Destroy (see page 4079)	Destroys the XSQLDA structure.

Public Properties

	Name	Description
R	AsXSQLDA (see page 4084)	Represents the XSQLDA field's value as an XSQLDA value.
	Count (see page 4085)	Returns the number of XSQLDA fields.
R	Modified (see page 4085)	Indicates whether a field has been modified.
R	Names (see page 4085)	Returns the XSQLDA field names.
R	RecordSize (see page 4085)	Returns the size of the XSQLDA record.
R	UniqueRelationName (see page 4086)	Returns the name of the unique relation.
R	Vars (see page 4086)	Returns the XSQLVAR defined for the XSQLDA parameter.

28.1.9.2 TIBXSQLDA.Create Constructor

Creates an instance of an XSQLDA structure.

Pascal

```
constructor Create(Query: TIBSQL);
```

C++

```
__fastcall TIBXSQLDA(TIBSQL Query);
```

Description

Call Create to create an instance of an XSQLDA structure. Create queries the DynamicSQL component for the structure members.

28.1.9.3 TIBXSQLDA.Destroy Destructor

Destroys the XSQLDA structure.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBXSQLDA();
```

Description

Do not call Destroy directly. Call Free instead. Free checks to ensure that the object instance is not nil before calling Destroy.

28.1.9.4 TIBXSQLDA Fields

The fields of the TIBXSQLDA class are listed here.

Protected Fields

	Name	Description
	FCount (see page 4080)	This is FCount, a member of class TIBXSQLDA.
	FNames (see page 4080)	This is FNames, a member of class TIBXSQLDA.
	FSize (see page 4080)	This is FSize, a member of class TIBXSQLDA.
	FSQL (see page 4081)	This is FSQL, a member of class TIBXSQLDA.
	FUniqueRelationName (see page 4081)	This is FUniqueRelationName, a member of class TIBXSQLDA.
	FXSQLDA (see page 4081)	This is FXSQLDA, a member of class TIBXSQLDA.
	FXSQLVARs (see page 4081)	This is FXSQLVARs, a member of class TIBXSQLDA.

28.1.9.4.1 TIBXSQLDA.FCount Field

Pascal

```
FCount: Integer;
```

C++

```
int FCount;
```

Description

This is FCount, a member of class TIBXSQLDA.

28.1.9.4.2 TIBXSQLDA.FNames Field

Pascal

```
FNames: TStrings;
```

C++

```
TStrings FNames;
```

Description

This is FNames, a member of class TIBXSQLDA.

28.1.9.4.3 TIBXSQLDA.FSize Field

Pascal

```
FSize: Integer;
```

C++

```
int FSize;
```

Description

This is FSize, a member of class TIBXSQLDA.

28.1.9.4.4 TIBXSQLDA.FSQL Field

Pascal

```
FSQL: TIBSQL;
```

C++

```
TIBSQL FSQL;
```

Description

This is FSQL, a member of class TIBXSQLDA.

28.1.9.4.5 TIBXSQLDA.FUniqueRelationName Field

Pascal

```
FUniqueRelationName: String;
```

C++

```
UnicodeString FUniqueRelationName;
```

Description

This is FUniqueRelationName, a member of class TIBXSQLDA.

28.1.9.4.6 TIBXSQLDA.FXSQLDA Field

Pascal

```
FXSQLDA: PXSQLDA;
```

C++

```
PXSQLDA FXSQLDA;
```

Description

This is FXSQLDA, a member of class TIBXSQLDA.

28.1.9.4.7 TIBXSQLDA.FXSQLVARs Field

Pascal

```
FXSQLVARs: TIBXSQLVARArray;
```

C++

```
TIBXSQLVARArray FXSQLVARs;
```

Description

This is FXSQLVARs, a member of class TIBXSQLDA.

28.1.9.5 TIBXSQLDA Methods

The methods of the TIBXSQLDA class are listed here.

Protected Methods

	Name	Description
	GetModified (see page 4082)	This is GetModified, a member of class TIBXSQLDA.
	GetNames (see page 4083)	This is GetNames, a member of class TIBXSQLDA.
	GetRecordSize (see page 4083)	This is GetRecordSize, a member of class TIBXSQLDA.
	GetXSQLDA (see page 4083)	This is GetXSQLDA, a member of class TIBXSQLDA.
	GetXSQLVAR (see page 4083)	This is GetXSQLVAR, a member of class TIBXSQLDA.
	GetXSQLVARByName (see page 4083)	This is GetXSQLVARByName, a member of class TIBXSQLDA.
	Initialize (see page 4084)	This is Initialize, a member of class TIBXSQLDA.
	SetCount (see page 4084)	This is SetCount, a member of class TIBXSQLDA.

Public Methods

	Name	Description
	AddName (see page 4082)	Adds a name to the XSQLDA structure.
	ByName (see page 4082)	Returns the XSQLVAR fields by name.

28.1.9.5.1 TIBXSQLDA.AddName Method

Adds a name to the XSQLDA structure.

Pascal

```
procedure AddName(FieldName: String; Idx: Integer);
```

C++

```
__fastcall AddName(UnicodeString FieldName, int Idx);
```

Description

Call AddName to add a name to the XSQLDA structure.

28.1.9.5.2 TIBXSQLDA.ByName Method

Returns the XSQLVAR fields by name.

Pascal

```
function ByName(Idx: String): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR ByName(UnicodeString Idx);
```

Description

Use the ByName method to return the XSQLVAR fields by name.

28.1.9.5.3 TIBXSQLDA.GetModified Method

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

This is GetModified, a member of class TIBXSQLDA.

28.1.9.5.4 TIBXSQLDA.GetNames Method

Pascal

```
function GetNames: String;
```

C++

```
__fastcall UnicodeString GetNames();
```

Description

This is GetNames, a member of class TIBXSQLDA.

28.1.9.5.5 TIBXSQLDA.GetRecordSize Method

Pascal

```
function GetRecordSize: Integer;
```

C++

```
__fastcall int GetRecordSize();
```

Description

This is GetRecordSize, a member of class TIBXSQLDA.

28.1.9.5.6 TIBXSQLDA.GetXSQLDA Method

Pascal

```
function GetXSQLDA: PXSQLDA;
```

C++

```
__fastcall PXSQLDA GetXSQLDA();
```

Description

This is GetXSQLDA, a member of class TIBXSQLDA.

28.1.9.5.7 TIBXSQLDA.GetXSQLVAR Method

Pascal

```
function GetXSQLVAR(Idx: Integer): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR GetXSQLVAR(int Idx);
```

Description

This is GetXSQLVAR, a member of class TIBXSQLDA.

28.1.9.5.8 TIBXSQLDA.GetXSQLVARByName Method

Pascal

```
function GetXSQLVARByName(Idx: String): TIBXSQLVAR;
```

C++

```
__fastcall TIBXSQLVAR GetXSQLVARByName(UnicodeString Idx);
```

Description

This is GetXSQLVARByName, a member of class TIBXSQLDA.

28.1.9.5.9 TIBXSQLDA.Initialize Method

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

This is Initialize, a member of class TIBXSQLDA.

28.1.9.5.10 TIBXSQLDA.SetCount Method

Pascal

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

C++

```
__fastcall SetCount(int Value);
```

Description

This is SetCount, a member of class TIBXSQLDA.

28.1.9.6 TIBXSQLDA Properties

The properties of the TIBXSQLDA class are listed here.

Public Properties

	Name	Description
R	AsXSQLDA (see page 4084)	Represents the XSQLDA field's value as an XSQLDA value.
	Count (see page 4085)	Returns the number of XSQLDA fields.
R	Modified (see page 4085)	Indicates whether a field has been modified.
R	Names (see page 4085)	Returns the XSQLDA field names.
R	RecordSize (see page 4085)	Returns the size of the XSQLDA record.
R	UniqueRelationName (see page 4086)	Returns the name of the unique relation.
R	Vars (see page 4086)	Returns the XSQLVAR defined for the XSQLDA parameter.

28.1.9.6.1 TIBXSQLDA.AsXSQLDA Property

Represents the XSQLDA field's value as an XSQLDA value.

Pascal

```
property AsXSQLDA: PXSQLDA;
```

C++

```
__property PXSQLDA AsXSQLDA;
```

Description

Use the AsXSQLDA property to read the value of the field's data into an object or variable of type XSQLDA, or to assign an

XSQLDA value to the contents of the field.

28.1.9.6.2 TIBXSQLDA.Count Property

Returns the number of XSQLDA fields.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use the Count property to return the number of XSQLDA fields.

28.1.9.6.3 TIBXSQLDA.Modified Property

Indicates whether a field has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use the Modified property to determine whether a field has been modified.

28.1.9.6.4 TIBXSQLDA.Names Property

Returns the XSQLDA field names.

Pascal

```
property Names: String;
```

C++

```
__property UnicodeString Names;
```

Description

Use the Names property to return the XSQLDA field names.

28.1.9.6.5 TIBXSQLDA.RecordSize Property

Returns the size of the XSQLDA record.

Pascal

```
property RecordSize: Integer;
```

C++

```
__property int RecordSize;
```

Description

Use the RecordSize property to return the XSQLDA record size.

28.1.9.6.6 TIBXSQLDA.UniqueRelationName Property

Returns the name of the unique relation.

Pascal

```
property UniqueRelationName: String;
```

C++

```
__property UnicodeString UniqueRelationName;
```

Description

Use the UniqueRelationName property to return the name of the relation if only one relation is involved in the query. Otherwise, it returns nil (Delphi) or NULL (C++). This property is primarily used for internal purposes.

28.1.9.6.7 TIBXSQLDA.Vars Property

Returns the XSQLVAR defined for the XSQLDA parameter.

Pascal

```
property Vars [Idx: Integer]: TIBXSQLVAR;
```

C++

```
__property TIBXSQLVAR Vars[int Idx];
```

Description

Use Vars to return the XSQLVAR defined for the XSQLDA parameter.

Vars is the default property for TIBXSQLDSA, which means that the name of the Vars property can be omitted when the TIBXSQLDA (see page 4078) object implements a property in another object.

28.1.10 TIBXSQLVAR Class

TIBXSQLVAR provides properties and methods for use with the IBSQL (see page 4035) component.

Class Hierarchy

Pascal

```
TIBXSQLVAR = class(TObject);
```

C++

```
class TIBXSQLVAR : public TObject;
```

File

IBSQL

Description

Use a TIBXSQLVAR object to provide properties and methods for use with the IBSQL (see page 4035) component.

The XSQLVAR structure is a field, sqlvar, in the XSQLDA. The sqlvar is especially important, because one XSQLVAR must be defined for each input parameter or column returned.

Applications do not declare instances of the XSQLVAR ahead of time, but must, instead, dynamically allocate storage for the proper number of XSQLVAR structures required for each DSQL statement before it is executed, then deallocate it, as appropriate, after statement execution.

28.1.10.1 TIBXSQLVAR Members

The following tables list the members exposed by TIBXSQLVAR.

Public Methods

	Name	Description
	Assign (see page 4088)	Assigns another XSQLVAR to the XSQLVAR component.
	Clear (see page 4088)	Clears the value of this TIBXSQLVAR (see page 4086) object.
	Create (see page 4087)	Creates an instance of an XSQLVAR structure.
	LoadFromFile (see page 4089)	Loads the contents of a file to a Blob field.
	LoadFromStream (see page 4089)	Loads a stream into a Blob field.
	SaveToFile (see page 4089)	Saves the contents of a Blob field to a file.
	SaveToStream (see page 4089)	Saves the contents of a Blob field to a stream.

Public Properties

	Name	Description
	AsCurrency (see page 4090)	Represents the XSQLVAR field's value as a Currency value.
	AsDate (see page 4090)	Represents the XSQLVAR field's value as a TDateTime value.
	AsDateTime (see page 4091)	Represents the XSQLVAR field's value as a TDateTime value.
	AsDouble (see page 4091)	Represents the XSQLVAR field's value as a Double value.
	AsFloat (see page 4091)	Represents the XSQLVAR field's value as a Float value.
	AsInt64 (see page 4092)	Represents the XSQLVAR field's value as a 64-bit integer.
	AsInteger (see page 4092)	Represents the XSQLVAR field's value as a 32-bit integer value.
	AsLong (see page 4092)	Represents the XSQLVAR field's value as a Long integer value.
	AsPointer (see page 4092)	Represents the XSQLVAR field's value as a Pointer value.
	AsQuad (see page 4093)	Represents the XSQLVAR field's value as a Quad value.
	AsShort (see page 4093)	Represents the XSQLVAR field's value as a Short integer value.
	AsString (see page 4093)	Represents the XSQLVAR field's value as a string.
	AsTime (see page 4093)	Represents the XSQLVAR field's value as a TDateTime value.
	AsTrimString (see page 4094)	Represents the XSQLVAR field's value as a string with any trailing spaces removed.
	AsVariant (see page 4094)	Represents the XSQLVAR field's value as type Variant.
	AsXSQLVAR (see page 4094)	Represents the XSQLVAR field's value as an XSQLVAR value.
	Data (see page 4095)	Accesses the underlying InterBase XSQLVAR structure.
R	Index (see page 4095)	Indicates the position of the XSQLVAR in the XSQLDA.
	IsNull (see page 4095)	Indicates whether the field has a value assigned to it.
	IsNullable (see page 4095)	Indicates whether the field can have a value assigned to it.
	Modified (see page 4096)	Indicates whether a field has been modified.
R	Name (see page 4096)	Returns the name of the XSQLVAR.
R	Size (see page 4096)	Indicates the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.
R	SQLType (see page 4096)	Indicates the SQL datatype of parameters or select list items.
	Value (see page 4097)	Returns the value of the XSQLVAR field component.

28.1.10.2 TIBXSQLVAR.Create Constructor

Creates an instance of an XSQLVAR structure.

Pascal

```
constructor Create(Parent: TIBXSQLDA; Query: TIBSQL);
```

C++

```
__fastcall TIBXSQLVAR(TIBXSQLDA Parent, TIBSQL Query);
```

Description

Call Create to create an instance of TIBXSQLVAR (see page 4086).

Parent is an extended descriptor area that defines the namespace to use for processing the variable name.

Query identifies the query that includes this variable.

28.1.10.3 TIBXSQLVAR Methods

The methods of the TIBXSQLVAR class are listed here.

Public Methods

	Name	Description
	Assign (see page 4088)	Assigns another XSQLVAR to the XSQLVAR component.
	Clear (see page 4088)	Clears the value of this TIBXSQLVAR (see page 4086) object.
	LoadFromFile (see page 4089)	Loads the contents of a file to a Blob field.
	LoadFromStream (see page 4089)	Loads a stream into a Blob field.
	SaveToFile (see page 4089)	Saves the contents of a Blob field to a file.
	SaveToStream (see page 4089)	Saves the contents of a Blob field to a stream.

28.1.10.3.1 TIBXSQLVAR.Assign Method

Assigns another XSQLVAR to the XSQLVAR component.

Pascal

```
procedure Assign(Source: TIBXSQLVAR);
```

C++

```
__fastcall Assign(TIBXSQLVAR Source);
```

Description

Use Assign to assign another XSQLVAR to the XSQLVAR component. The fields must have compatible data types.

28.1.10.3.2 TIBXSQLVAR.Clear Method

Clears the value of this TIBXSQLVAR (see page 4086) object.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to assign a Null value to this TIBXSQLVAR (see page 4086) object.

28.1.10.3.3 TIBXSQLVAR.LoadFromFile Method

Loads the contents of a file to a Blob field.

Pascal

```
procedure LoadFromFile(const FileName: String);
```

C++

```
__fastcall LoadFromFile(const UnicodeString FileName);
```

Description

Call LoadFromFile to load the contents of a file to a Blob field.

28.1.10.3.4 TIBXSQLVAR.LoadFromStream Method

Loads a stream into a Blob field.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to load a stream into a Blob field.

28.1.10.3.5 TIBXSQLVAR.SaveToFile Method

Saves the contents of a Blob field to a file.

Pascal

```
procedure SaveToFile(const FileName: String);
```

C++

```
__fastcall SaveToFile(const UnicodeString FileName);
```

Description

Call SaveToFile to save the contents of a Blob field to a file.

28.1.10.3.6 TIBXSQLVAR.SaveToStream Method

Saves the contents of a Blob field to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the contents of a Blob field to a stream.

28.1.10.4 TIBXSQLVAR Properties

The properties of the TIBXSQLVAR class are listed here.

Public Properties

	Name	Description
	AsCurrency (see page 4090)	Represents the XSQLVAR field's value as a Currency value.
	AsDate (see page 4090)	Represents the XSQLVAR field's value as a TDateTime value.
	AsDateTime (see page 4091)	Represents the XSQLVAR field's value as a TDateTime value.
	AsDouble (see page 4091)	Represents the XSQLVAR field's value as a Double value.
	AsFloat (see page 4091)	Represents the XSQLVAR field's value as a Float value.
	AsInt64 (see page 4092)	Represents the XSQLVAR field's value as a 64-bit integer.
	AsInteger (see page 4092)	Represents the XSQLVAR field's value as a 32-bit integer value.
	AsLong (see page 4092)	Represents the XSQLVAR field's value as a Long integer value.
	AsPointer (see page 4092)	Represents the XSQLVAR field's value as a Pointer value.
	AsQuad (see page 4093)	Represents the XSQLVAR field's value as a Quad value.
	AsShort (see page 4093)	Represents the XSQLVAR field's value as a Short integer value.
	AsString (see page 4093)	Represents the XSQLVAR field's value as a string.
	AsTime (see page 4093)	Represents the XSQLVAR field's value as a TDateTime value.
	AsTrimString (see page 4094)	Represents the XSQLVAR field's value as a string with any trailing spaces removed.
	AsVariant (see page 4094)	Represents the XSQLVAR field's value as type Variant.
	AsXSQLVAR (see page 4094)	Represents the XSQLVAR field's value as an XSQLVAR value.
	Data (see page 4095)	Accesses the underlying InterBase XSQLVAR structure.
R	Index (see page 4095)	Indicates the position of the XSQLVAR in the XSQLDA.
	IsNull (see page 4095)	Indicates whether the field has a value assigned to it.
	IsNullable (see page 4095)	Indicates whether the field can have a value assigned to it.
	Modified (see page 4096)	Indicates whether a field has been modified.
R	Name (see page 4096)	Returns the name of the XSQLVAR.
R	Size (see page 4096)	Indicates the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.
R	SQLType (see page 4096)	Indicates the SQL datatype of parameters or select list items.
	Value (see page 4097)	Returns the value of the XSQLVAR field component.

28.1.10.4.1 TIBXSQLVAR.AsCurrency Property

Represents the XSQLVAR field's value as a Currency value.

Pascal

```
property AsCurrency: Currency;
```

C++

```
__property Currency AsCurrency;
```

Description

Use the AsCurrency property to read the value of the XSQLVAR field's data into an object or variable of type Currency, or to assign a Currency value to the contents of the field.

28.1.10.4.2 TIBXSQLVAR.AsDate Property

Represents the XSQLVAR field's value as a TDateTime value.

Pascal

```
property AsDate: TDateTime;
```

C++

```
__property TDateTime AsDate;
```

Description

Use the AsDate property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field. The Time portion of the TDateTime value is set to zero.

28.1.10.4.3 TIBXSQLVAR.AsDateTime Property

Represents the XSQLVAR field's value as a TDateTime value.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Use the AsDateTime property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field.

28.1.10.4.4 TIBXSQLVAR.AsDouble Property

Represents the XSQLVAR field's value as a Double value.

Pascal

```
property AsDouble: Double;
```

C++

```
__property Double AsDouble;
```

Description

Use the AsDouble property to read the value of the field's data into an object or variable of type Double, or to assign a Double value to the contents of the field.

28.1.10.4.5 TIBXSQLVAR.AsFloat Property

Represents the XSQLVAR field's value as a Float value.

Pascal

```
property AsFloat: Float;
```

C++

```
__property Float AsFloat;
```

Description

Use the AsDouble (see page 4091) property to read the value of the field's data into an object or variable of type Float, or to assign a Float value to the contents of the field.

28.1.10.4.6 TIBXSQLVAR.AsInt64 Property

Represents the XSQLVAR field's value as a 64-bit integer.

Pascal

```
property AsInt64: Int64;
```

C++

```
__property Int64 AsInt64;
```

Description

Use the AsInt64 property to read the value of the field's data into a 64-bit integer, or to assign an 64-bit integer value to the contents of the field.

28.1.10.4.7 TIBXSQLVAR.AsInteger Property

Represents the XSQLVAR field's value as a 32-bit integer value.

Pascal

```
property AsInteger: Integer;
```

C++

```
__property int AsInteger;
```

Description

Use the AsInteger property to read the value of the field's data into an integer, or to assign an integer value to the contents of the field.

28.1.10.4.8 TIBXSQLVAR.AsLong Property

Represents the XSQLVAR field's value as a Long integer value.

Pascal

```
property AsLong: Long;
```

C++

```
__property Long AsLong;
```

Description

Use the AsLong property to read the value of the field's data into a Long integer, or to assign a Long integer value to the contents of the field.

28.1.10.4.9 TIBXSQLVAR.AsPointer Property

Represents the XSQLVAR field's value as a Pointer value.

Pascal

```
property AsPointer: Pointer;
```

C++

```
__property void * AsPointer;
```

Description

Use the AsPointer property to read the value of the field's data into an object or variable of type Pointer, or to assign a Pointer value to the contents of the field.

28.1.10.4.10 TIBXSQLVAR.AsQuad Property

Represents the XSQLVAR field's value as a Quad value.

Pascal

```
property AsQuad: TISC_QUAD;
```

C++

```
__property TISC_QUAD AsQuad;
```

Description

Use the AsQuad property to read the value of the field's data into an object or variable of type Quad, or to assign a Quad value to the contents of the field.

28.1.10.4.11 TIBXSQLVAR.AsShort Property

Represents the XSQLVAR field's value as a Short integer value.

Pascal

```
property AsShort: Short;
```

C++

```
__property Short AsShort;
```

Description

Use the AsShort property to to read the value of the field's data into a Short integer, or to assign a Short integer value to the contents of the field.

28.1.10.4.12 TIBXSQLVAR.AsString Property

Represents the XSQLVAR field's value as a string.

Pascal

```
property AsString: String;
```

C++

```
__property UnicodeString AsString;
```

Description

Use the AsString property to read the value of the field's data into a string, or to assign a String value to the contents of the field.

28.1.10.4.13 TIBXSQLVAR.AsTime Property

Represents the XSQLVAR field's value as a TDateTime value.

Pascal

```
property AsTime: TDateTime;
```

C++

```
__property TDateTime AsTime;
```

Description

Use the AsTime property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field. The Date portion of the field is set to zero.

28.1.10.4.14 TIBXSQLVAR.AsTrimString Property

Represents the XSQLVAR field's value as a string with any trailing spaces removed.

Pascal

```
property AsTrimString: String;
```

C++

```
__property UnicodeString AsTrimString;
```

Description

Use the AsTrimString property to read the value of the field's data into a string, removing any trailing spaces from the value.

When assigning AsTrimString, any trailing spaces in the assigned string are removed before assigning the value. The spaces do not become part of the underlying value.

28.1.10.4.15 TIBXSQLVAR.AsVariant Property

Represents the XSQLVAR field's value as type Variant.

Pascal

```
property AsVariant: Variant;
```

C++

```
__property Variant AsVariant;
```

Description

Use the AsVariant property to read the value of the field's data into a Variant, or to assign a Variant value to the contents of the field.

28.1.10.4.16 TIBXSQLVAR.AsXSQLVAR Property

Represents the XSQLVAR field's value as an XSQLVAR value.

Pascal

```
property AsXSQLVAR: PXSQLVAR;
```

C++

```
__property PXSQLVAR AsXSQLVAR;
```

Description

Use the AsXSQLVAR property to read the value of the field's data into an object or variable of type XSQLVAR, or to assign an XSQLVAR value to the contents of the field.

28.1.10.4.17 TIBXSQLVAR.Data Property

Accesses the underlying InterBase XSQLVAR structure.

Pascal

```
property Data: PXSQLVAR;
```

C++

```
__property PXSQLVAR Data;
```

Description

Use the Data property to access the underlying InterBase XSQLVAR structure.

28.1.10.4.18 TIBXSQLVAR.Index Property

Indicates the position of the XSQLVAR in the XSQLDA.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Use the Index property to obtain the position of the XSQLVAR in the XSQLDA.

28.1.10.4.19 TIBXSQLVAR.IsNull Property

Indicates whether the field has a value assigned to it.

Pascal

```
property IsNull: Boolean;
```

C++

```
__property Boolean IsNull;
```

Description

Use the IsNull property to determine if the field contains a value. If IsNull is true, the field is blank. If IsNull is false, the field has a value.

28.1.10.4.20 TIBXSQLVAR.IsNullable Property

Indicates whether the field can have a value assigned to it.

Pascal

```
property IsNullable: Boolean;
```

C++

```
__property Boolean IsNullable;
```

Description

Use the IsNullable property to determine if the field can contain a value. If IsNullable is true, the field can contain a value. If IsNullable is false, the field cannot contain a value.

28.1.10.4.21 TIBXSQLVAR.Modified Property

Indicates whether a field has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use the Modified property to determine whether a field has been modified.

28.1.10.4.22 TIBXSQLVAR.Name Property

Returns the name of the XSQLVAR.

Pascal

```
property Name: String;
```

C++

```
__property UnicodeString Name;
```

Description

Use the Name property to return the name of the XSQLVAR.

28.1.10.4.23 TIBXSQLVAR.Size Property

Indicates the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use the Size property to return the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.

28.1.10.4.24 TIBXSQLVAR.SQLType Property

Indicates the SQL datatype of parameters or select list items.

Pascal

```
property SQLType: Integer;
```

C++

```
__property int SQLType;
```

Description

Read the SQLType property to indicate the SQL datatype of parameters or select list items.

28.1.10.4.25 TIBXSQLVAR.Value Property

Returns the value of the XSQLVAR field component.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

Use Value to return the value of the XSQLVAR field component as a Variant.

28.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TIBSQLTypes (see page 4098)	TIBSQLTypes indicates the type of an SQL query against an InterBase database.

28.2.1 IBSQL.TIBSQLTypes Enumeration

TIBSQLTypes indicates the type of an SQL query against an InterBase database.

Pascal

```
TIBSQLTypes = (
  SQLUnknown,
  SQLSelect,
  SQLInsert,
  SQLUpdate,
  SQLDelete,
  SQLDDL,
  SQLGetSegment,
  SQLPutSegment,
  SQLExecProcedure,
  SQLStartTransaction,
  SQLCommit,
  SQLRollback,
  SQLSelectForUpdate,
  SQLSetGenerator
);
```

C++

```
enum TIBSQLTypes {
  SQLUnknown,
  SQLSelect,
  SQLInsert,
  SQLUpdate,
  SQLDelete,
  SQLDDL,
  SQLGetSegment,
  SQLPutSegment,
  SQLExecProcedure,
  SQLStartTransaction,
  SQLCommit,
  SQLRollback,
  SQLSelectForUpdate,
  SQLSetGenerator
};
```

File

IBSQL

Description

TIBSQLTypes values are:

SQLCommit	Commits an active transaction
SQLDDL	Executes a DDL statement
SQLDelete	Removes rows in a table or in the active set of a cursor
SQLExecProcedure	Calls a stored procedure
SQLGetSegment	Reads a segment from an open Blob
SQLInsert	Adds one or more new rows to a specified table
SQLPutSegment	Writes a Blob segment
SQLRollback	Restores the database to its state prior to the start of the current transaction
SQLSetForUpdate	Stored procedure is set for updating
SQLSetGenerator	Sets a new value for an existing generator
SQLSelect	Retrieves data from one or more tables
SQLStartTransaction	Starts a new transaction against one or more databases
SQLUnknown	Unknown SQL type
SQLUpdate	Changes data in all or part of an existing row in a table, view, or active set of a cursor

28.3 Types

The following table lists types in this documentation.

Types

Name	Description
TIBXSQLVARArray (see page 4100)	This is type IBSQL.TIBXSQLVARArray.

28.3.1 IBSQL.TIBXSQLVARArray Type

Pascal

```
TIBXSQLVARArray = Array of TIBXSQLVAR;
```

C++

```
Array of TIBXSQLVAR TIBXSQLVARArray;
```

File

IBSQL

Description

This is type IBSQL.TIBXSQLVARArray.

29 IBSQLMonitor Namespace

This is namespace IBSQLMonitor.

Topics

Name	Description
Classes (see page 4103)	The following table lists classes in this documentation.
Interfaces (see page 4109)	The following table lists interfaces in this documentation.
Functions (see page 4117)	The following table lists functions in this documentation.
Types (see page 4119)	The following table lists types in this documentation.
Constants (see page 4120)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIBCustomSQLMonitor (see page 4103)	TIBCustomSQLMonitor is the ancestor for TIBSQLMonitor (see page 4106).
	TIBSQLMonitor (see page 4106)	TIBSQLMonitor monitors dynamic SQL passed to the InterBase server.

Constants

Name	Description
WM_IBSQL_SQL_EVENT (see page 4120)	This is constant IBSQLMonitor.WM_IBSQL_SQL_EVENT.
WM_MAX_IBSQL_MONITOR (see page 4120)	This is constant IBSQLMonitor.WM_MAX_IBSQL_MONITOR.
WM_MIN_IBSQL_MONITOR (see page 4120)	This is constant IBSQLMonitor.WM_MIN_IBSQL_MONITOR.

Functions

	Name	Description
	DisableMonitoring (see page 4117)	Disables SQL monitoring.
	EnableMonitoring (see page 4117)	This is function IBSQLMonitor.EnableMonitoring.
	MonitorHook (see page 4118)	This is function IBSQLMonitor.MonitorHook.
	MonitoringEnabled (see page 4118)	This is function IBSQLMonitor.MonitoringEnabled.

Interfaces

	Name	Description
	IIBSQLMonitorHook (see page 4109)	This is class IBSQLMonitor.IIBSQLMonitorHook.

Types

Name	Description
TSQLEvent (see page 4119)	TSQLEvent is the type of event handlers that respond to activity on an InterBase server.

29.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBCustomSQLMonitor (see page 4103)	TIBCustomSQLMonitor is the ancestor for TIBSQLMonitor (see page 4106).
	TIBSQLMonitor (see page 4106)	TIBSQLMonitor monitors dynamic SQL passed to the InterBase server.

29.1.1 TIBCustomSQLMonitor Class

TIBCustomSQLMonitor is the ancestor for TIBSQLMonitor (see page 4106).

Class Hierarchy

Pascal

```
TIBCustomSQLMonitor = class(TComponent);
```

C++

```
class TIBCustomSQLMonitor : public TComponent;
```

File

IBSQLMonitor

Description

TIBCustomSQLMonitor is the ancestor object from which TIBSQLMonitor (see page 4106) is derived.

29.1.1.1 TIBCustomSQLMonitor Members

The following tables list the members exposed by TIBCustomSQLMonitor.

Protected Events

	Name	Description
	OnSQL (see page 4106)	Reports dynamic SQL activity on InterBase applications.

Protected Properties

	Name	Description
	Enabled (see page 4105)	Indicates whether the IB (see page 3541) SQL monitor is actively monitoring communication with the server.
	TraceFlags (see page 4106)	Indicates which database operations are tracked.

Public Methods

	Name	Description
	Create (see page 4104)	Creates an instance of TIBCustomSQLMonitor (see page 4103).
	Destroy (see page 4104)	Destroys the instance of TIBCustomSQLMonitor (see page 4103).
	Release (see page 4104)	Frees the helper objects that intercept database commands.

Public Properties

	Name	Description
	Handle (see page 4105)	Specifies the window handle that the SQL monitor uses to receive asynchronous messages.

29.1.1.2 TIBCustomSQLMonitor.Create Constructor

Creates an instance of TIBCustomSQLMonitor (see page 4103).

Pascal

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

C++

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

Description

Call Create to create an instance of TIBCustomSQLMonitor (see page 4103). Create also:

Creates the window

Registers the event

Fires the Monitor thread

29.1.1.3 TIBCustomSQLMonitor.Destroy Destructor

Destroys the instance of TIBCustomSQLMonitor (see page 4103).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBCustomSQLMonitor();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Destroy unregisters the event, destroys the window, and kills the Monitor thread.

29.1.1.4 TIBCustomSQLMonitor Methods

The methods of the TIBCustomSQLMonitor class are listed here.

Public Methods

	Name	Description
	Release (see page 4104)	Frees the helper objects that intercept database commands.

29.1.1.4.1 TIBCustomSQLMonitor.Release Method

Frees the helper objects that intercept database commands.

Pascal

```
procedure Release;
```

C++

```
__fastcall Release();
```

Description

Call Release when the SQL monitor will not be monitoring any more messages. Release sends a message to the object that intercepts database commands, telling it to free itself.

To temporarily disable the monitor, use the Enabled (see page 4105) property instead.

29.1.1.5 TIBCustomSQLMonitor Properties

The properties of the TIBCustomSQLMonitor class are listed here.

Protected Properties

	Name	Description
	Enabled (see page 4105)	Indicates whether the IB (see page 3541) SQL monitor is actively monitoring communication with the server.
	TraceFlags (see page 4106)	Indicates which database operations are tracked.

Public Properties

	Name	Description
R	Handle (see page 4105)	Specifies the window handle that the SQL monitor uses to receive asynchronous messages.

29.1.1.5.1 TIBCustomSQLMonitor.Enabled Property

Indicates whether the IB (see page 3541) SQL monitor is actively monitoring communication with the server.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to turn the SQL monitor on and off. When Enabled is true, the SQL monitor receives an OnSQL (see page 4106) event for every database operation specified by the TraceFlags (see page 4106) property. When Enabled is false, the SQL monitor does not receive any events.

29.1.1.5.2 TIBCustomSQLMonitor.Handle Property

Specifies the window handle that the SQL monitor uses to receive asynchronous messages.

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

A separate thread of execution monitors the database operations that trigger OnSQL (see page 4106) events. When that

thread detects a relevant database operation, it sends a Windows message to the window identified by this Handle.

29.1.1.5.3 TIBCustomSQLMonitor.TraceFlags Property

Indicates which database operations are tracked.

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags enables performance tuning and SQL debugging when working with remote SQL database servers.

Note: Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

29.1.1.6 TIBCustomSQLMonitor Events

The events of the TIBCustomSQLMonitor class are listed here.

Protected Events

	Name	Description
	OnSQL (see page 4106)	Reports dynamic SQL activity on InterBase applications.

29.1.1.6.1 TIBCustomSQLMonitor.OnSQL Event

Reports dynamic SQL activity on InterBase applications.

Pascal

```
property OnSQL: TSQLEvent;
```

C++

```
__property TSQLEvent OnSQL;
```

Description

Write an OnSQL event handler to report dynamic SQL activity on InterBase applications.

Note: You must enable the TraceFlags (see page 4106) in each TIBDatabase component in order for the SQL monitor to receive status information from each database connection.

29.1.2 TIBSQLMonitor Class

TIBSQLMonitor monitors dynamic SQL passed to the InterBase server.

Class Hierarchy

Pascal

```
TIBSQLMonitor = class(TIBCustomSQLMonitor);
```

C++

```
class TIBSQLMonitor : public TIBCustomSQLMonitor;
```

File

IBSQLMonitor

Description

Use TIBSQLMonitor to monitor dynamic SQL taking place in InterBase applications. Enable TraceFlags (see page 4108) in each TIBDatabase component in order for the SQL monitor to receive status information from each database connection.

See Also

Getting Started with InterBase Express

29.1.2.1 TIBSQLMonitor Members

The following tables list the members exposed by TIBSQLMonitor.

Public Methods

	Name	Description
	Create (see page 4104)	Creates an instance of TIBCustomSQLMonitor (see page 4103).
	Destroy (see page 4104)	Destroys the instance of TIBCustomSQLMonitor (see page 4103).
	Release (see page 4104)	Frees the helper objects that intercept database commands.

Public Properties

	Name	Description
	Handle (see page 4105)	Specifies the window handle that the SQL monitor uses to receive asynchronous messages.

Published Events

	Name	Description
	OnSQL (see page 4108)	Reports dynamic SQL activity on InterBase applications.

Published Properties

	Name	Description
	Enabled (see page 4107)	Indicates whether the IB SQL monitor is actively monitoring communication with the server.
	TraceFlags (see page 4108)	Indicates which database operations are tracked.

29.1.2.2 TIBSQLMonitor Properties

The properties of the TIBSQLMonitor class are listed here.

Published Properties

	Name	Description
	Enabled (see page 4107)	Indicates whether the IB SQL monitor is actively monitoring communication with the server.
	TraceFlags (see page 4108)	Indicates which database operations are tracked.

29.1.2.2.1 TIBSQLMonitor.Enabled Property

Indicates whether the IB SQL monitor is actively monitoring communication with the server.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to turn the SQL monitor on and off. When Enabled is true, the SQL monitor receives an OnSQL event for every database operation specified by the TraceFlags property. When Enabled is false, the SQL monitor does not receive any events.

29.1.2.2.2 TIBSQLMonitor.TraceFlags Property

Indicates which database operations are tracked.

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags enables performance tuning and SQL debugging when working with remote SQL database servers.

Note: Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

29.1.2.3 TIBSQLMonitor Events

The events of the TIBSQLMonitor class are listed here.

Published Events

	Name	Description
	OnSQL (see page 4108)	Reports dynamic SQL activity on InterBase applications.

29.1.2.3.1 TIBSQLMonitor.OnSQL Event

Reports dynamic SQL activity on InterBase applications.

Pascal

```
property OnSQL: TSQLEvent;
```

C++

```
__property TSQLEvent OnSQL;
```

Description

Write an OnSQL event handler to report dynamic SQL activity on InterBase applications.

Note: You must enable the TraceFlags in each TIBDatabase component in order for the SQL monitor to receive status information from each database connection.

29.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IIBSQLMonitorHook (see page 4109)	This is class IBSQLMonitor.IIBSQLMonitorHook.

29.2.1 IIBSQLMonitorHook Interface

Class Hierarchy

IBSQLMonitor.IIBSQLMonitorHook

Pascal

```
IIBSQLMonitorHook = interface;
```

C++

```
__interface IIBSQLMonitorHook;
```

File

IBSQLMonitor

Description

This is class IBSQLMonitor.IIBSQLMonitorHook.

29.2.1.1 IIBSQLMonitorHook Members

The following tables list the members exposed by IIBSQLMonitorHook.

Public Methods

	Name	Description
	DBConnect (see page 4110)	This is DBConnect, a member of class IIBSQLMonitorHook.
	DBDisconnect (see page 4111)	This is DBDisconnect, a member of class IIBSQLMonitorHook.
	GetEnabled (see page 4111)	This is GetEnabled, a member of class IIBSQLMonitorHook.
	GetMonitorCount (see page 4111)	This is GetMonitorCount, a member of class IIBSQLMonitorHook.
	GetTraceFlags (see page 4111)	This is GetTraceFlags, a member of class IIBSQLMonitorHook.
	RegisterMonitor (see page 4112)	This is RegisterMonitor, a member of class IIBSQLMonitorHook.
	ReleaseMonitor (see page 4112)	This is ReleaseMonitor, a member of class IIBSQLMonitorHook.
	SendMisc (see page 4112)	This is SendMisc, a member of class IIBSQLMonitorHook.
	ServiceAttach (see page 4112)	This is ServiceAttach, a member of class IIBSQLMonitorHook.
	ServiceDetach (see page 4112)	This is ServiceDetach, a member of class IIBSQLMonitorHook.
	ServiceQuery (see page 4113)	This is ServiceQuery, a member of class IIBSQLMonitorHook.
	ServiceStart (see page 4113)	This is ServiceStart, a member of class IIBSQLMonitorHook.
	SetEnabled (see page 4113)	This is SetEnabled, a member of class IIBSQLMonitorHook.
	SetTraceFlags (see page 4113)	This is SetTraceFlags, a member of class IIBSQLMonitorHook.
	SQLExecute (see page 4114)	This is SQLExecute, a member of class IIBSQLMonitorHook.

	SQLFetch (see page 4114)	This is SQLFetch, a member of class IIBSQLMonitorHook.
	SQLPrepare (see page 4114)	This is SQLPrepare, a member of class IIBSQLMonitorHook.
	TRCommit (see page 4114)	This is TRCommit, a member of class IIBSQLMonitorHook.
	TRCommitRetaining (see page 4114)	This is TRCommitRetaining, a member of class IIBSQLMonitorHook.
	TRRollback (see page 4115)	This is TRRollback, a member of class IIBSQLMonitorHook.
	TRRollbackRetaining (see page 4115)	This is TRRollbackRetaining, a member of class IIBSQLMonitorHook.
	TRStart (see page 4115)	This is TRStart, a member of class IIBSQLMonitorHook.
	UnregisterMonitor (see page 4115)	This is UnregisterMonitor, a member of class IIBSQLMonitorHook.

Public Properties

	Name	Description
	Enabled (see page 4116)	This is Enabled, a member of class IIBSQLMonitorHook.
	TraceFlags (see page 4116)	This is TraceFlags, a member of class IIBSQLMonitorHook.

29.2.1.2 IIBSQLMonitorHook Methods

The methods of the IIBSQLMonitorHook class are listed here.

Public Methods

	Name	Description
	DBConnect (see page 4110)	This is DBConnect, a member of class IIBSQLMonitorHook.
	DBDisconnect (see page 4111)	This is DBDisconnect, a member of class IIBSQLMonitorHook.
	GetEnabled (see page 4111)	This is GetEnabled, a member of class IIBSQLMonitorHook.
	GetMonitorCount (see page 4111)	This is GetMonitorCount, a member of class IIBSQLMonitorHook.
	GetTraceFlags (see page 4111)	This is GetTraceFlags, a member of class IIBSQLMonitorHook.
	RegisterMonitor (see page 4112)	This is RegisterMonitor, a member of class IIBSQLMonitorHook.
	ReleaseMonitor (see page 4112)	This is ReleaseMonitor, a member of class IIBSQLMonitorHook.
	SendMisc (see page 4112)	This is SendMisc, a member of class IIBSQLMonitorHook.
	ServiceAttach (see page 4112)	This is ServiceAttach, a member of class IIBSQLMonitorHook.
	ServiceDetach (see page 4112)	This is ServiceDetach, a member of class IIBSQLMonitorHook.
	ServiceQuery (see page 4113)	This is ServiceQuery, a member of class IIBSQLMonitorHook.
	ServiceStart (see page 4113)	This is ServiceStart, a member of class IIBSQLMonitorHook.
	SetEnabled (see page 4113)	This is SetEnabled, a member of class IIBSQLMonitorHook.
	SetTraceFlags (see page 4113)	This is SetTraceFlags, a member of class IIBSQLMonitorHook.
	SQLExecute (see page 4114)	This is SQLExecute, a member of class IIBSQLMonitorHook.
	SQLFetch (see page 4114)	This is SQLFetch, a member of class IIBSQLMonitorHook.
	SQLPrepare (see page 4114)	This is SQLPrepare, a member of class IIBSQLMonitorHook.
	TRCommit (see page 4114)	This is TRCommit, a member of class IIBSQLMonitorHook.
	TRCommitRetaining (see page 4114)	This is TRCommitRetaining, a member of class IIBSQLMonitorHook.
	TRRollback (see page 4115)	This is TRRollback, a member of class IIBSQLMonitorHook.
	TRRollbackRetaining (see page 4115)	This is TRRollbackRetaining, a member of class IIBSQLMonitorHook.
	TRStart (see page 4115)	This is TRStart, a member of class IIBSQLMonitorHook.
	UnregisterMonitor (see page 4115)	This is UnregisterMonitor, a member of class IIBSQLMonitorHook.

29.2.1.2.1 IIBSQLMonitorHook.DBConnect Method

Pascal

```
procedure DBConnect(db: TIBDatabase);
```

C++

```
__fastcall DBConnect(TIBDatabase db);
```

Description

This is DBConnect, a member of class IIBSQLMonitorHook.

29.2.1.2.2 IIBSQLMonitorHook.DBDisconnect Method

Pascal

```
procedure DBDisconnect(db: TIBDatabase);
```

C++

```
__fastcall DBDisconnect(TIBDatabase db);
```

Description

This is DBDisconnect, a member of class IIBSQLMonitorHook.

29.2.1.2.3 IIBSQLMonitorHook.GetEnabled Method

Pascal

```
function GetEnabled: boolean;
```

C++

```
__fastcall boolean GetEnabled();
```

Description

This is GetEnabled, a member of class IIBSQLMonitorHook.

29.2.1.2.4 IIBSQLMonitorHook.GetMonitorCount Method

Pascal

```
function GetMonitorCount: Integer;
```

C++

```
__fastcall int GetMonitorCount();
```

Description

This is GetMonitorCount, a member of class IIBSQLMonitorHook.

29.2.1.2.5 IIBSQLMonitorHook.GetTraceFlags Method

Pascal

```
function GetTraceFlags: TTraceFlags;
```

C++

```
__fastcall TTraceFlags GetTraceFlags();
```

Description

This is GetTraceFlags, a member of class IIBSQLMonitorHook.

29.2.1.2.6 IIBSQLMonitorHook.RegisterMonitor Method

Pascal

```
procedure RegisterMonitor(SQLMonitor: TIBCustomSQLMonitor);
```

C++

```
__fastcall RegisterMonitor(TIBCustomSQLMonitor SQLMonitor);
```

Description

This is RegisterMonitor, a member of class IIBSQLMonitorHook.

29.2.1.2.7 IIBSQLMonitorHook.ReleaseMonitor Method

Pascal

```
procedure ReleaseMonitor(Arg: TIBCustomSQLMonitor);
```

C++

```
__fastcall ReleaseMonitor(TIBCustomSQLMonitor Arg);
```

Description

This is ReleaseMonitor, a member of class IIBSQLMonitorHook.

29.2.1.2.8 IIBSQLMonitorHook.SendMisc Method

Pascal

```
procedure SendMisc(Msg: String);
```

C++

```
__fastcall SendMisc(UnicodeString Msg);
```

Description

This is SendMisc, a member of class IIBSQLMonitorHook.

29.2.1.2.9 IIBSQLMonitorHook.ServiceAttach Method

Pascal

```
procedure ServiceAttach(service: TIBCustomService);
```

C++

```
__fastcall ServiceAttach(TIBCustomService service);
```

Description

This is ServiceAttach, a member of class IIBSQLMonitorHook.

29.2.1.2.10 IIBSQLMonitorHook.ServiceDetach Method

Pascal

```
procedure ServiceDetach(service: TIBCustomService);
```

C++

```
__fastcall ServiceDetach(TIBCustomService service);
```

Description

This is ServiceDetach, a member of class IIBSQLMonitorHook.

29.2.1.2.11 IIBSQLMonitorHook.ServiceQuery Method

Pascal

```
procedure ServiceQuery(service: TIBCustomService);
```

C++

```
__fastcall ServiceQuery(TIBCustomService service);
```

Description

This is ServiceQuery, a member of class IIBSQLMonitorHook.

29.2.1.2.12 IIBSQLMonitorHook.ServiceStart Method

Pascal

```
procedure ServiceStart(service: TIBCustomService);
```

C++

```
__fastcall ServiceStart(TIBCustomService service);
```

Description

This is ServiceStart, a member of class IIBSQLMonitorHook.

29.2.1.2.13 IIBSQLMonitorHook.SetEnabled Method

Pascal

```
procedure SetEnabled(const Value: Boolean);
```

C++

```
__fastcall SetEnabled(const Boolean Value);
```

Description

This is SetEnabled, a member of class IIBSQLMonitorHook.

29.2.1.2.14 IIBSQLMonitorHook.SetTraceFlags Method

Pascal

```
procedure SetTraceFlags(const Value: TTraceFlags);
```

C++

```
__fastcall SetTraceFlags(const TTraceFlags Value);
```

Description

This is SetTraceFlags, a member of class IIBSQLMonitorHook.

29.2.1.2.15 IIBSQLMonitorHook.SQLExecute Method

Pascal

```
procedure SQLExecute(qry: TIBSQL);
```

C++

```
__fastcall SQLExecute(TIBSQL qry);
```

Description

This is SQLExecute, a member of class IIBSQLMonitorHook.

29.2.1.2.16 IIBSQLMonitorHook.SQLFetch Method

Pascal

```
procedure SQLFetch(qry: TIBSQL);
```

C++

```
__fastcall SQLFetch(TIBSQL qry);
```

Description

This is SQLFetch, a member of class IIBSQLMonitorHook.

29.2.1.2.17 IIBSQLMonitorHook.SQLPrepare Method

Pascal

```
procedure SQLPrepare(qry: TIBSQL);
```

C++

```
__fastcall SQLPrepare(TIBSQL qry);
```

Description

This is SQLPrepare, a member of class IIBSQLMonitorHook.

29.2.1.2.18 IIBSQLMonitorHook.TRCommit Method

Pascal

```
procedure TRCommit(tr: TIBTransaction);
```

C++

```
__fastcall TRCommit(TIBTransaction tr);
```

Description

This is TRCommit, a member of class IIBSQLMonitorHook.

29.2.1.2.19 IIBSQLMonitorHook.TRCommitRetaining Method

Pascal

```
procedure TRCommitRetaining(tr: TIBTransaction);
```

C++

```
__fastcall TRCommitRetaining(TIBTransaction tr);
```

Description

This is TRCommitRetaining, a member of class IIBSQLMonitorHook.

29.2.1.2.20 IIBSQLMonitorHook.TRRollback Method

Pascal

```
procedure TRRollback(tr: TIBTransaction);
```

C++

```
__fastcall TRRollback(TIBTransaction tr);
```

Description

This is TRRollback, a member of class IIBSQLMonitorHook.

29.2.1.2.21 IIBSQLMonitorHook.TRRollbackRetaining Method

Pascal

```
procedure TRRollbackRetaining(tr: TIBTransaction);
```

C++

```
__fastcall TRRollbackRetaining(TIBTransaction tr);
```

Description

This is TRRollbackRetaining, a member of class IIBSQLMonitorHook.

29.2.1.2.22 IIBSQLMonitorHook.TRStart Method

Pascal

```
procedure TRStart(tr: TIBTransaction);
```

C++

```
__fastcall TRStart(TIBTransaction tr);
```

Description

This is TRStart, a member of class IIBSQLMonitorHook.

29.2.1.2.23 IIBSQLMonitorHook.UnregisterMonitor Method

Pascal

```
procedure UnregisterMonitor(SQLMonitor: TIBCustomSQLMonitor);
```

C++

```
__fastcall UnregisterMonitor(TIBCustomSQLMonitor SQLMonitor);
```

Description

This is UnregisterMonitor, a member of class IIBSQLMonitorHook.

29.2.1.3 IIBSQLMonitorHook Properties

The properties of the IIBSQLMonitorHook class are listed here.

Public Properties

	Name	Description
	Enabled (see page 4116)	This is Enabled, a member of class IIBSQLMonitorHook.
	TraceFlags (see page 4116)	This is TraceFlags, a member of class IIBSQLMonitorHook.

29.2.1.3.1 IIBSQLMonitorHook.Enabled Property

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

This is Enabled, a member of class IIBSQLMonitorHook.

29.2.1.3.2 IIBSQLMonitorHook.TraceFlags Property

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

This is TraceFlags, a member of class IIBSQLMonitorHook.

29.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DisableMonitoring (see page 4117)	Disables SQL monitoring.
	EnableMonitoring (see page 4117)	This is function IBSQLMonitor.EnableMonitoring.
	MonitorHook (see page 4118)	This is function IBSQLMonitor.MonitorHook.
	MonitoringEnabled (see page 4118)	This is function IBSQLMonitor.MonitoringEnabled.

29.3.1 IBSQLMonitor.DisableMonitoring Function

Disables SQL monitoring.

Pascal

```
procedure DisableMonitoring;
```

C++

```
DisableMonitoring();
```

File

IBSQLMonitor

Description

Use the DisableMonitoring procedure to disable SQL monitoring.

29.3.2 IBSQLMonitor.EnableMonitoring Function

Pascal

```
procedure EnableMonitoring;
```

C++

```
EnableMonitoring();
```

File

IBSQLMonitor

Description

This is function IBSQLMonitor.EnableMonitoring.

29.3.3 IBSQLMonitor.MonitorHook Function

Pascal

```
function MonitorHook: IIBSQLMonitorHook;
```

C++

```
IIBSQLMonitorHook MonitorHook();
```

File

IBSQLMonitor

Description

This is function IBSQLMonitor.MonitorHook.

29.3.4 IBSQLMonitor.MonitoringEnabled Function

Pascal

```
function MonitoringEnabled: Boolean;
```

C++

```
Boolean MonitoringEnabled();
```

File

IBSQLMonitor

Description

This is function IBSQLMonitor.MonitoringEnabled.

29.4 Types

The following table lists types in this documentation.

Types

Name	Description
TSQLEvent (see page 4119)	TSQLEvent is the type of event handlers that respond to activity on an InterBase server.

29.4.1 IBSQLMonitor.TSQLEvent Type

TSQLEvent is the type of event handlers that respond to activity on an InterBase server.

Pascal

```
TSQLEvent = procedure (EventText: String; EventTime : TDateTime) of object;
```

C++

```
(EventText: String; EventTime : TDateTime) ( TSQLEvent)();
```

File

IBSQLMonitor

Description

EventText is the text of the SQL command that triggered the event.

EventTime indicates the time when the command passed to or from the InterBase server.

29.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WM_IBSQL_SQL_EVENT (see page 4120)	This is constant IBSQLMonitor.WM_IBSQL_SQL_EVENT.
WM_MAX_IBSQL_MONITOR (see page 4120)	This is constant IBSQLMonitor.WM_MAX_IBSQL_MONITOR.
WM_MIN_IBSQL_MONITOR (see page 4120)	This is constant IBSQLMonitor.WM_MIN_IBSQL_MONITOR.

29.5.1 IBSQLMonitor.WM_IBSQL_SQL_EVENT Constant

Pascal

```
WM_IBSQL_SQL_EVENT = WM_MIN_IBSQL_MONITOR + 1;
```

C++

```
WM_IBSQL_SQL_EVENT = WM_MIN_IBSQL_MONITOR + 1;
```

File

IBSQLMonitor

Description

This is constant IBSQLMonitor.WM_IBSQL_SQL_EVENT.

29.5.2 IBSQLMonitor.WM_MAX_IBSQL_MONITOR Constant

Pascal

```
WM_MAX_IBSQL_MONITOR = WM_USER + 512;
```

C++

```
WM_MAX_IBSQL_MONITOR = WM_USER + 512;
```

File

IBSQLMonitor

Description

This is constant IBSQLMonitor.WM_MAX_IBSQL_MONITOR.

29.5.3 IBSQLMonitor.WM_MIN_IBSQL_MONITOR Constant

Pascal

```
WM_MIN_IBSQL_MONITOR = WM_USER;
```

C++

```
WM_MIN_IBSQL_MONITOR = WM_USER;
```

File

IBSQLMonitor

Description

This is constant IBSQLMonitor.WM_MIN_IBSQL_MONITOR.

30 IBStoredProc Namespace

This is namespace IBStoredProc.

Topics

Name	Description
Classes (see page 4124)	The following table lists classes in this documentation.

Classes

	Name	Description
	TIBStoredProc (see page 4124)	TIBStoredProc encapsulates a stored procedure on a database server.

30.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBStoredProc (see page 4124)	TIBStoredProc encapsulates a stored procedure on a database server.

30.1.1 TIBStoredProc Class

TIBStoredProc encapsulates a stored procedure on a database server.

Class Hierarchy

Pascal

```
TIBStoredProc = class(TIBCustomDataSet);
```

C++

```
class TIBStoredProc : public TIBCustomDataSet;
```

File

IBStoredProc

Description

Use a TIBStoredProc object when a client application must use a stored procedure on a database server. A stored procedure is a grouped set of statements, stored as part of a database server's metadata (just like tables, indexes, and domains), that performs a frequently repeated, database-related task on the server and passes results to the client.

Many stored procedures require a series of input arguments, or parameters, that are used during processing. TIBStoredProc provides a Params (see page 4135) property that enables an application to set these parameters before executing the stored procedure.

TIBStoredProc reuses the Params (see page 4135) property to hold the results returned by a stored procedure. Params (see page 4135) is an array of values. A stored procedure will return a single set of values, or none at all.

Note: TIBStoredProc components should be used with InterBase Execute stored procedures only. To use InterBase Select procedures, use TIBQuery or TIBDataSet (see page 3654). Since Execute stored procedures do not return result sets, never use Open or Active on the TIBStoredProc component. Use ExecProc (see page 4130) instead.

See Also

Types of Datasets

Getting Started with InterBase Express

30.1.1.1 TIBStoredProc Members

The following tables list the members exposed by TIBStoredProc.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.
	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.
V	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
V	DefineProperties (see page 4130)	This is DefineProperties, a member of class TIBStoredProc.
V	Disconnect (see page 4130)	This is Disconnect, a member of class TIBStoredProc.
V	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
V	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
V	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
V	DoOnNewRecord (see page 3614)	This is DoOnNewRecord, a member of class TIBCustomDataSet.
V	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
V	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
V	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
V	GetCanModify (see page 3615)	This is GetCanModify, a member of class TIBCustomDataSet.
V	GetDataSource (see page 3616)	This is GetDataSource, a member of class TIBCustomDataSet.
V	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
	GetParamsCount (see page 4131)	This is GetParamsCount, a member of class TIBStoredProc.
V	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
V	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
V	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
V	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
V	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
V	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
V	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
V	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.
V	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
V	InternalClose (see page 3619)	This is InternalClose, a member of class TIBCustomDataSet.
V	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
V	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
V	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
V	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
V	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
V	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
V	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
V	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
V	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
V	InternalOpen (see page 4131)	This is InternalOpen, a member of class TIBStoredProc.
V	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
V	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
V	InternalRefresh (see page 3622)	This is InternalRefresh, a member of class TIBCustomDataSet.
V	InternalRefreshRow (see page 3622)	This is InternalRefreshRow, a member of class TIBCustomDataSet.
V	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
V	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.
V	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.
V	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.

	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
	PSExecute (see page 4132)	This is PSExecute, a member of class TIBStoredProc.
	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
	PSGetParams (see page 4132)	This is PSGetParams, a member of class TIBStoredProc.
	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
	PSGetTableName (see page 4132)	This is PSGetTableName, a member of class TIBStoredProc.
	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.
	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
	PSSetCommandText (see page 4133)	This is PSSetCommandText, a member of class TIBStoredProc.
	PSSetParams (see page 4133)	This is PSSetParams, a member of class TIBStoredProc.
	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 4133)	This is SetFiltered, a member of class TIBStoredProc.
	SetPrepare (see page 4133)	This is SetPrepare, a member of class TIBStoredProc.
	SetPrepared (see page 4133)	This is SetPrepared, a member of class TIBStoredProc.
	SetProcName (see page 4134)	This is SetProcName, a member of class TIBStoredProc.
	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.

Protected Properties

	Name	Description
	BufferChunks (see page 3638)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 3638)	Specifies whether cached updates are enabled for a dataset.
	DeleteSQL (see page 3639)	Holds the SQL statement used to delete rows from the dataset.
	GeneratorField (see page 3640)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3640)	Holds the SQL statement used to insert rows into the dataset.
R	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
R	LiveMode (see page 3641)	Indicates what types of edits, if any, can be made to the data in the dataset.
	ModifySQL (see page 3641)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	ParamCheck (see page 3642)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
R	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.
R	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
R	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
R	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
R	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	RefreshSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
R	SelectStmtHandle (see page 3645)	Gets the handle for the SELECT statement.

	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
	StatementType (see page 3646)	Returns the statement type of the QSelect (see page 3644) query.
	UniDirectional (see page 3647)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Methods

	Name	Description
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.
	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.
	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
	CompareBookmarks (see page 3611)	Compares two bookmarks.
	CopyParams (see page 4130)	Copies a stored procedure's parameters into another parameter list.
	Create (see page 4128)	Creates an instance of a stored procedure component.
	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
	Destroy (see page 4129)	Destroys the instance of a stored procedure.
	ExecProc (see page 4130)	Executes the stored procedure on the server.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.
	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
	Locate (see page 3624)	Searches the dataset for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.
	ParamByName (see page 4131)	Accesses parameter information based on a specified parameter name.
	Post (see page 3626)	Writes a modified record to the database server.
	Prepare (see page 4132)	Prepares a stored procedure for execution.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
	UnPrepare (see page 4134)	Frees the resources allocated for a previously prepared stored procedure.
	UpdateStatus (see page 3632)	Reports the update status for the current record.

Public Properties

	Name	Description
	DBHandle (see page 3639)	Specifies the database handle for the dataset.
	ParamCount (see page 4135)	Indicates the number of parameters for the stored procedure component.
	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
	Prepared (see page 4136)	Determines whether or not a stored procedure is prepared for execution.
	RowsAffected (see page 3645)	Indicates the number of rows operated on by the latest query execution.
	StmtHandle (see page 4136)	Identifies the statement handle for the stored procedure.
	StoredProcedureNames (see page 4136)	Returns a list of stored procedures in the database.
	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateObject (see page 3647)	Specifies the update object component used to update a read-only result set.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 4137)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 4137)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 4138)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 4138)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 4138)	Occurs when the database associated with this dataset is freed.
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.
	TransactionFree (see page 4138)	Occurs when the transaction that includes this dataset's query is freed.

Published Properties

	Name	Description
	AfterCancel (see page 3634)	This is AfterCancel, a member of class TIBCustomDataSet.
	AfterClose (see page 3634)	This is AfterClose, a member of class TIBCustomDataSet.
	AfterDelete (see page 3634)	This is AfterDelete, a member of class TIBCustomDataSet.
	AfterEdit (see page 3634)	This is AfterEdit, a member of class TIBCustomDataSet.
	AfterInsert (see page 3635)	This is AfterInsert, a member of class TIBCustomDataSet.
	AfterOpen (see page 3635)	This is AfterOpen, a member of class TIBCustomDataSet.
	AfterPost (see page 3635)	This is AfterPost, a member of class TIBCustomDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3635)	This is AfterScroll, a member of class TIBCustomDataSet.
	AutoCalcFields (see page 3636)	This is AutoCalcFields, a member of class TIBCustomDataSet.
	BeforeCancel (see page 3636)	This is BeforeCancel, a member of class TIBCustomDataSet.
	BeforeClose (see page 3636)	This is BeforeClose, a member of class TIBCustomDataSet.
	BeforeDelete (see page 3636)	This is BeforeDelete, a member of class TIBCustomDataSet.
	BeforeEdit (see page 3637)	This is BeforeEdit, a member of class TIBCustomDataSet.
	BeforeInsert (see page 3637)	This is BeforeInsert, a member of class TIBCustomDataSet.
	BeforeOpen (see page 3637)	This is BeforeOpen, a member of class TIBCustomDataSet.
	BeforePost (see page 3637)	This is BeforePost, a member of class TIBCustomDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3638)	This is BeforeScroll, a member of class TIBCustomDataSet.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	Filtered (see page 4134)	Specifies whether filtering is active for a dataset.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3641)	This is OnCalcFields, a member of class TIBCustomDataSet.
	OnDeleteError (see page 3642)	This is OnDeleteError, a member of class TIBCustomDataSet.
	OnEditError (see page 3642)	This is OnEditError, a member of class TIBCustomDataSet.
	OnFilterRecord (see page 4135)	This is OnFilterRecord, a member of class TIBStoredProc.
	OnNewRecord (see page 3642)	This is OnNewRecord, a member of class TIBCustomDataSet.
	OnPostError (see page 3642)	This is OnPostError, a member of class TIBCustomDataSet.
	Params (see page 4135)	Stores the input and output parameters for a stored procedure.
	StoredProcName (see page 4137)	Identifies the name of the stored procedure on the server for which this object is an encapsulation.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.

30.1.1.2 TIBStoredProc.Create Constructor

Creates an instance of a stored procedure component.

Pascal

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

C++

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

Description

Call Create to instantiate a stored procedure declared in an application. Create calls its inherited Create constructor, creates an empty parameter list for the newly instantiated stored procedure, and initializes its parameter, server, and record buffers to nil.

30.1.1.3 TIBStoredProc.Destroy Destructor

Destroys the instance of a stored procedure.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBStoredProc();
```

Description

Do not call Destroy directly. Instead call Free to verify that the stored procedure is not already freed before calling Destroy. Destroy disconnects from the server, frees the parameter list, and calls its inherited Destroy destructor.

30.1.1.4 TIBStoredProc Methods

The methods of the TIBStoredProc class are listed here.

Protected Methods

	Name	Description
	DefineProperties (see page 4130)	This is DefineProperties, a member of class TIBStoredProc.
	Disconnect (see page 4130)	This is Disconnect, a member of class TIBStoredProc.
	GetParamsCount (see page 4131)	This is GetParamsCount, a member of class TIBStoredProc.
	InternalOpen (see page 4131)	This is InternalOpen, a member of class TIBStoredProc.
	PSExecute (see page 4132)	This is PSExecute, a member of class TIBStoredProc.
	PSGetParams (see page 4132)	This is PSGetParams, a member of class TIBStoredProc.
	PSGetTableName (see page 4132)	This is PSGetTableName, a member of class TIBStoredProc.
	PSSetCommandText (see page 4133)	This is PSSetCommandText, a member of class TIBStoredProc.
	PSSetParams (see page 4133)	This is PSSetParams, a member of class TIBStoredProc.
	SetFiltered (see page 4133)	This is SetFiltered, a member of class TIBStoredProc.
	SetPrepare (see page 4133)	This is SetPrepare, a member of class TIBStoredProc.
	SetPrepared (see page 4133)	This is SetPrepared, a member of class TIBStoredProc.
	SetProcName (see page 4134)	This is SetProcName, a member of class TIBStoredProc.

Public Methods

	Name	Description
	CopyParams (see page 4130)	Copies a stored procedure's parameters into another parameter list.
	ExecProc (see page 4130)	Executes the stored procedure on the server.
	ParamByName (see page 4131)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 4132)	Prepares a stored procedure for execution.
	UnPrepare (see page 4134)	Frees the resources allocated for a previously prepared stored procedure.

30.1.1.4.1 TIBStoredProc.CopyParams Method

Copies a stored procedure's parameters into another parameter list.

Pascal

```
procedure CopyParams(Value: TParams);
```

C++

```
__fastcall CopyParams(TParams Value);
```

Description

Call CopyParams to copy this stored procedure's parameters into a separate parameter list object. Value is the parameter list into which to assign this stored procedure's parameters. Value can be the parameter list of another stored procedure. For example:

```
TIBStoredProc1.CopyParams(TIBStoredProc2.Params);
    TIBStoredProc1->CopyParams(TIBStoredProc2->Params);
```

If the stored procedure is not prepared when an application calls CopyParams, CopyParams calls Prepare (see page 4132) before assigning the parameters to the target parameters list, and then calls UnPrepare (see page 4134) to return the stored procedure to its previous state.

30.1.1.4.2 TIBStoredProc.DefineProperties Method

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

This is DefineProperties, a member of class TIBStoredProc.

30.1.1.4.3 TIBStoredProc.Disconnect Method

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TIBStoredProc.

30.1.1.4.4 TIBStoredProc.ExecProc Method

Executes the stored procedure on the server.

Pascal

```
procedure ExecProc;
```

C++

```
__fastcall ExecProc();
```

Description

Call ExecProc to execute a stored procedure on the server. Before calling ExecProc:

1. **1** Provide any input parameters in the Params (see page 4135) property. At design time, a developer can provide parameters using the Parameters editor. At runtime an application must access Params (see page 4135) directly.
2. **2** Call Prepare (see page 4132) to bind the parameters.

If a stored procedure returns output parameters, they are stored in the Params (see page 4135) property when ExecProc returns control to the application. An application can access the output parameters by indexing into the Params (see page 4135) list, or by using the ParamByName (see page 4131) method.

30.1.1.4.5 TIBStoredProc.GetParamsCount Method

Pascal

```
function GetParamsCount: Word;
```

C++

```
__fastcall Word GetParamsCount();
```

Description

This is GetParamsCount, a member of class TIBStoredProc.

30.1.1.4.6 TIBStoredProc.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TIBStoredProc.

30.1.1.4.7 TIBStoredProc.ParamByName Method

Accesses parameter information based on a specified parameter name.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const UnicodeString Value);
```

Description

Call ParamByName to return parameter information for a specific parameter based on its name. Value is the name of the parameter for which to retrieve information. Typically ParamByName is used to set an input parameter's value at runtime, or to retrieve the value of an output parameter. The following command line assigns the value "Jane Smith" as the value for the parameter named Contact:

```
StoredProc1.ParamByName('Contact').AsString := 'Jane Smith';
    StoredProc1->ParamByName("Contact")->AsString = "Jane Smith";
```

30.1.1.4.8 TIBStoredProc.Prepare Method

Prepares a stored procedure for execution.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call Prepare to bind a stored procedure's parameters before calling ExecProc (see page 4130) to execute the procedure. Prepare readies a stored procedure's parameters and informs the server of the stored procedure's readiness. These steps allocate system resources and optimize the query for server performance.

Note: A stored procedure is automatically prepared if it is unprepared when the application calls ExecProc (see page 4130). After execution, the stored procedure is then automatically unprepared. If a stored procedure will be executed more than once, it is more efficient for an application to call Prepare explicitly once to avoid repeated and unnecessary preparing and unpreparing of the stored procedure, and then call UnPrepare (see page 4134) when the stored procedure is no longer needed.

30.1.1.4.9 TIBStoredProc.PSExecute Method

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

This is PSExecute, a member of class TIBStoredProc.

30.1.1.4.10 TIBStoredProc.PSGetParams Method

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

This is PSGetParams, a member of class TIBStoredProc.

30.1.1.4.11 TIBStoredProc.PSGetTableName Method

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetTableName();
```

Description

This is PSGetTableName, a member of class TIBStoredProc.

30.1.1.4.12 TIBStoredProc.PSSetCommandText Method

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const UnicodeString CommandText);
```

Description

This is PSSetCommandText, a member of class TIBStoredProc.

30.1.1.4.13 TIBStoredProc.PSSetParams Method

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

This is PSSetParams, a member of class TIBStoredProc.

30.1.1.4.14 TIBStoredProc.SetFiltered Method

Pascal

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

C++

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

Description

This is SetFiltered, a member of class TIBStoredProc.

30.1.1.4.15 TIBStoredProc.SetPrepare Method

Pascal

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

C++

```
__fastcall SetPrepare(Boolean Value);
```

Description

This is SetPrepare, a member of class TIBStoredProc.

30.1.1.4.16 TIBStoredProc.SetPrepared Method

Pascal

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

C++

```
__fastcall SetPrepared(Boolean Value);
```

Description

This is SetPrepared, a member of class TIBStoredProc.

30.1.1.4.17 TIBStoredProc.SetProcName Method

Pascal

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

C++

```
__fastcall SetProcName(UnicodeString Value);
```

Description

This is SetProcName, a member of class TIBStoredProc.

30.1.1.4.18 TIBStoredProc.UnPrepare Method

Frees the resources allocated for a previously prepared stored procedure.

Pascal

```
procedure UnPrepare;
```

C++

```
__fastcall UnPrepare();
```

Description

Call UnPrepare to free the resources allocated for a previously prepared stored procedure on the server and client sides.

30.1.1.5 TIBStoredProc Properties

The properties of the TIBStoredProc class are listed here.

Public Properties

	Name	Description
R	ParamCount (see page 4135)	Indicates the number of parameters for the stored procedure component.
	Prepared (see page 4136)	Determines whether or not a stored procedure is prepared for execution.
R	StmtHandle (see page 4136)	Identifies the statement handle for the stored procedure.
R	StoredProcedureNames (see page 4136)	Returns a list of stored procedures in the database.

Published Properties

	Name	Description
	Filtered (see page 4134)	Specifies whether filtering is active for a dataset.
	OnFilterRecord (see page 4135)	This is OnFilterRecord, a member of class TIBStoredProc.
	Params (see page 4135)	Stores the input and output parameters for a stored procedure.
	StoredProcName (see page 4137)	Identifies the name of the stored procedure on the server for which this object is an encapsulation.

30.1.1.5.1 TIBStoredProc.Filtered Property

Specifies whether filtering is active for a dataset.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether dataset filtering is in effect. If Filtered is true, then filtering is active. Otherwise Filtered is false. To apply filter conditions specified in the Filter property or the OnFilterRecord (see page 4135) event handler, set Filtered to true. Filtered is set to false by default.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

30.1.1.5.2 TIBStoredProc.OnFilterRecord Property

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

This is OnFilterRecord, a member of class TIBStoredProc.

30.1.1.5.3 TIBStoredProc.ParamCount Property

Indicates the number of parameters for the stored procedure component.

Pascal

```
property ParamCount: Word;
```

C++

```
__property Word ParamCount;
```

Description

Examine ParamCount to determine the number of parameters currently stored in the Params (see page 4135) property.

30.1.1.5.4 TIBStoredProc.Params Property

Stores the input and output parameters for a stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Access Params at runtime to set input parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). Params is an array of parameter values.

An application can also access Params after executing a stored procedure to retrieve the output parameters returned to the

procedure by the server.

30.1.1.5.5 TIBStoredProc.Prepared Property

Determines whether or not a stored procedure is prepared for execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Examine Prepared to determine if a stored procedure is already prepared for execution. If Prepared is true, the stored procedure is prepared, and if Prepared is false, the procedure is not prepared. A stored procedure must be prepared before it can be executed.

Note: A stored procedure is automatically prepared if it is unprepared when the application calls ExecProc (see page 4130). After execution, the stored procedure is then automatically unprepared. If a procedure will be executed a number of times, it is more efficient for the application to prepare the stored procedure once, and unprepare it when it is no longer needed.

An application can change the current setting of Prepared to prepare or unprepare a stored procedure. If Prepared is true, setting it to false calls the Unprepare method to unprepare the stored procedure. If Prepared is false, setting it to true calls the Prepare (see page 4132) method to prepare the procedure. Generally, however, it is better programming practice to call Prepare (see page 4132) and Unprepare directly. These methods automatically update the Prepared property.

30.1.1.5.6 TIBStoredProc.StmtHandle Property

Identifies the statement handle for the stored procedure.

Pascal

```
property StmtHandle: TISC_STMT_HANDLE;
```

C++

```
__property TISC_STMT_HANDLE StmtHandle;
```

Description

Retrieve StmtHandle if an application makes a direct call to the InterBase server, bypassing the methods of TIBStoredProc (see page 4124). Some API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

30.1.1.5.7 TIBStoredProc.StoredProcedureNames Property

Returns a list of stored procedures in the database.

Pascal

```
property StoredProcedureNames: TStrings;
```

C++

```
__property TStrings StoredProcedureNames;
```

Description

StoredProcedureNames is an internal property used to list the stored procedures available in the database.

30.1.1.5.8 TIBStoredProc.StoredProcName Property

Identifies the name of the stored procedure on the server for which this object is an encapsulation.

Pascal

```
property StoredProcName: string;
```

C++

```
__property UnicodeString StoredProcName;
```

Description

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

30.1.1.6 TIBStoredProc Events

The events of the TIBStoredProc class are listed here.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 4137)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 4137)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 4138)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 4138)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 4138)	Occurs when the database associated with this dataset is freed.
	TransactionFree (see page 4138)	Occurs when the transaction that includes this dataset's query is freed.

30.1.1.6.1 TIBStoredProc.AfterDatabaseDisconnect Event

Occurs immediately after the connection to the associated database is closed.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to respond when the dataset closes its connection to the database specified by the Database property.

30.1.1.6.2 TIBStoredProc.AfterTransactionEnd Event

Occurs immediately after a transaction that includes this dataset's query ends.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to respond when the transaction specified by the Transaction property ends.

30.1.1.6.3 TIBStoredProc.BeforeDatabaseDisconnect Event

Occurs immediately before the connection to the associated database is closed.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to respond when the dataset is about to close its connection to the database specified by the Database property.

30.1.1.6.4 TIBStoredProc.BeforeTransactionEnd Event

Occurs immediately before a transaction that includes this dataset's query ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to respond when the transaction specified by the Transaction property is about to end.

30.1.1.6.5 TIBStoredProc.DatabaseFree Event

Occurs when the database associated with this dataset is freed.

Pascal

```
property DatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent DatabaseFree;
```

Description

Write a DatabaseFree event handler to respond when the database specified by the Database property is freed.

30.1.1.6.6 TIBStoredProc.TransactionFree Event

Occurs when the transaction that includes this dataset's query is freed.

Pascal

```
property TransactionFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent TransactionFree;
```

Description

Write a TransactionFree event handler to respond when the transaction specified by the Transaction property is freed.

31 IBTable Namespace

This is namespace IBTable.

Topics

Name	Description
Classes (see page 4142)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4168)	The following table lists structs, records, enums in this documentation.
Types (see page 4169)	The following table lists types in this documentation.

Classes

	Name	Description
	TIBTable (see page 4142)	TIBTable is a dataset component that encapsulates a database table.

Structs, Records, Enums

	Name	Description
	TIBTableType (see page 4168)	This is record IBTable.TIBTableType.

Types

Name	Description
TIBTableTypes (see page 4169)	TIBTableTypes and TIBTableTypes indicate the types of InterBase tables and views other than user tables.
TIndexName (see page 4169)	This is type IBTable.TIndexName.

31.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBTable (see page 4142)	TIBTable is a dataset component that encapsulates a database table.

31.1.1 TIBTable Class

TIBTable is a dataset component that encapsulates a database table.

Class Hierarchy

Pascal

```
TIBTable = class(TIBCustomDataSet);
```

C++

```
class TIBTable : public TIBCustomDataSet;
```

File

IBTable

Description

Use TIBTable to access data in a single table or view. TIBTable provides direct access to every record and field in an underlying InterBase database table. A table component can also work with a subset of records within a database table using filters.

See Also

Types of Datasets

Getting Started with InterBase Express

31.1.1.1 TIBTable Members

The following tables list the members exposed by TIBTable.

Protected Methods

	Name	Description
	ActivateConnection (see page 3609)	This is ActivateConnection, a member of class TIBCustomDataSet.
	ActivateTransaction (see page 3609)	This is ActivateTransaction, a member of class TIBCustomDataSet.
V	AllocRecordBuffer (see page 3609)	This is AllocRecordBuffer, a member of class TIBCustomDataSet.
	CheckDatasetClosed (see page 3611)	This is CheckDatasetClosed, a member of class TIBCustomDataSet.
	CheckDatasetOpen (see page 3611)	This is CheckDatasetOpen, a member of class TIBCustomDataSet.
	CheckNotUniDirectional (see page 3611)	This is CheckNotUniDirectional, a member of class TIBCustomDataSet.
V	ClearCalcFields (see page 3611)	This is ClearCalcFields, a member of class TIBCustomDataSet.
	ConstraintsStored (see page 3612)	This is ConstraintsStored, a member of class TIBCustomDataSet.

	CreateFields (see page 3612)	This is CreateFields, a member of class TIBCustomDataSet.
	DataEvent (see page 4149)	This is DataEvent, a member of class TIBTable.
	DeactivateTransaction (see page 3613)	This is DeactivateTransaction, a member of class TIBCustomDataSet.
	DefChanged (see page 4149)	This is DefChanged, a member of class TIBTable.
	Disconnect (see page 3613)	This is Disconnect, a member of class TIBCustomDataSet.
	DoBeforeDelete (see page 3613)	This is DoBeforeDelete, a member of class TIBCustomDataSet.
	DoBeforeEdit (see page 3613)	This is DoBeforeEdit, a member of class TIBCustomDataSet.
	DoBeforeInsert (see page 3614)	This is DoBeforeInsert, a member of class TIBCustomDataSet.
	DoOnNewRecord (see page 4150)	This is DoOnNewRecord, a member of class TIBTable.
	FreeRecordBuffer (see page 3615)	This is FreeRecordBuffer, a member of class TIBCustomDataSet.
	GetActiveBuf (see page 3615)	This is GetActiveBuf, a member of class TIBCustomDataSet.
	GetBookmarkData (see page 3615)	This is GetBookmarkData, a member of class TIBCustomDataSet.
	GetBookmarkFlag (see page 3615)	This is GetBookmarkFlag, a member of class TIBCustomDataSet.
	GetCanModify (see page 4150)	This is GetCanModify, a member of class TIBTable.
	GetDataSource (see page 4151)	This is GetDataSource, a member of class TIBTable.
	GetFieldClass (see page 3616)	This is GetFieldClass, a member of class TIBCustomDataSet.
	GetIndexParams (see page 4151)	This is GetIndexParams, a member of class TIBTable.
	GetRecNo (see page 3617)	This is GetRecNo, a member of class TIBCustomDataSet.
	GetRecord (see page 3617)	This is GetRecord, a member of class TIBCustomDataSet.
	GetRecordCount (see page 3618)	This is GetRecordCount, a member of class TIBCustomDataSet.
	GetRecordSize (see page 3618)	This is GetRecordSize, a member of class TIBCustomDataSet.
	InitFieldDefs (see page 4152)	This is InitFieldDefs, a member of class TIBTable.
	InitRecord (see page 3618)	This is InitRecord, a member of class TIBCustomDataSet.
	InternalAddRecord (see page 3618)	This is InternalAddRecord, a member of class TIBCustomDataSet.
	InternalBatchInput (see page 3619)	This is InternalBatchInput, a member of class TIBCustomDataSet.
	InternalBatchOutput (see page 3619)	This is InternalBatchOutput, a member of class TIBCustomDataSet.
	InternalCancel (see page 3619)	This is InternalCancel, a member of class TIBCustomDataSet.
	InternalClose (see page 4152)	This is InternalClose, a member of class TIBTable.
	InternalDelete (see page 3619)	This is InternalDelete, a member of class TIBCustomDataSet.
	InternalExecQuery (see page 3620)	This is InternalExecQuery, a member of class TIBCustomDataSet.
	InternalFirst (see page 3620)	This is InternalFirst, a member of class TIBCustomDataSet.
	InternalGotoBookmark (see page 3620)	This is InternalGotoBookmark, a member of class TIBCustomDataSet.
	InternalHandleException (see page 3620)	This is InternalHandleException, a member of class TIBCustomDataSet.
	InternalInitFieldDefs (see page 3621)	This is InternalInitFieldDefs, a member of class TIBCustomDataSet.
	InternalInitRecord (see page 3621)	This is InternalInitRecord, a member of class TIBCustomDataSet.
	InternalInsert (see page 3621)	This is InternalInsert, a member of class TIBCustomDataSet.
	InternalLast (see page 3621)	This is InternalLast, a member of class TIBCustomDataSet.
	InternalOpen (see page 4152)	This is InternalOpen, a member of class TIBTable.
	InternalPost (see page 3622)	This is InternalPost, a member of class TIBCustomDataSet.
	InternalPrepare (see page 3622)	This is InternalPrepare, a member of class TIBCustomDataSet.
	InternalRefresh (see page 4153)	This is InternalRefresh, a member of class TIBTable.
	InternalRefreshRow (see page 4153)	This is InternalRefreshRow, a member of class TIBTable.
	InternalSetParamsFromCursor (see page 3623)	This is InternalSetParamsFromCursor, a member of class TIBCustomDataSet.
	InternalSetToRecord (see page 3623)	This is InternalSetToRecord, a member of class TIBCustomDataSet.
	InternalUnPrepare (see page 3623)	This is InternalUnPrepare, a member of class TIBCustomDataSet.
	IsCursorOpen (see page 3623)	This is IsCursorOpen, a member of class TIBCustomDataSet.
	Loaded (see page 3624)	This is Loaded, a member of class TIBCustomDataSet.
	PSEndTransaction (see page 3626)	This is PSEndTransaction, a member of class TIBCustomDataSet.
	PSExecuteStatement (see page 3626)	This is PSExecuteStatement, a member of class TIBCustomDataSet.
	PSGetDefaultOrder (see page 4154)	This is PSGetDefaultOrder, a member of class TIBTable.
	PSGetIndexDefs (see page 4154)	This is PSGetIndexDefs, a member of class TIBTable.
	PSGetKeyFields (see page 4154)	This is PSGetKeyFields, a member of class TIBTable.
	PSGetQuoteChar (see page 3627)	This is PSGetQuoteChar, a member of class TIBCustomDataSet.
	PSGetTableName (see page 4154)	This is PSGetTableName, a member of class TIBTable.

V	PSGetUpdateException (see page 3627)	This is PSGetUpdateException, a member of class TIBCustomDataSet.
V	PSInTransaction (see page 3627)	This is PSInTransaction, a member of class TIBCustomDataSet.
V	PSIsSQLBased (see page 3628)	This is PSIsSQLBased, a member of class TIBCustomDataSet.
V	PSIsSQLSupported (see page 3628)	This is PSIsSQLSupported, a member of class TIBCustomDataSet.
V	PSReset (see page 3628)	This is PSReset, a member of class TIBCustomDataSet.
V	PSSetCommandText (see page 4155)	This is PSSetCommandText, a member of class TIBTable.
V	PSSetParams (see page 4155)	This is PSSetParams, a member of class TIBTable.
V	PSStartTransaction (see page 3628)	This is PSStartTransaction, a member of class TIBCustomDataSet.
V	PSUpdateRecord (see page 3628)	This is PSUpdateRecord, a member of class TIBCustomDataSet.
	ReQuery (see page 3629)	This is ReQuery, a member of class TIBCustomDataSet.
V	SetActive (see page 3629)	This is SetActive, a member of class TIBCustomDataSet.
V	SetBookmarkData (see page 3630)	This is SetBookmarkData, a member of class TIBCustomDataSet.
V	SetBookmarkFlag (see page 3630)	This is SetBookmarkFlag, a member of class TIBCustomDataSet.
	SetCachedUpdates (see page 3630)	This is SetCachedUpdates, a member of class TIBCustomDataSet.
	SetDataSource (see page 3630)	This is SetDataSource, a member of class TIBCustomDataSet.
V	SetFieldData (see page 3630)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 4155)	This is SetFiltered, a member of class TIBTable.
V	SetFilterOptions (see page 4155)	This is SetFilterOptions, a member of class TIBTable.
V	SetFilterText (see page 4156)	This is SetFilterText, a member of class TIBTable.
V	SetRecNo (see page 3631)	This is SetRecNo, a member of class TIBCustomDataSet.
V	UpdateIndexDefs (see page 4156)	This is UpdateIndexDefs, a member of class TIBTable.

Protected Properties

	Name	Description
	DeleteSQL (see page 3639)	Holds the SQL statement used to delete rows from the dataset.
	GeneratorField (see page 3640)	Specifies whether the dataset should refresh its data every time it posts a record.
	InsertSQL (see page 3640)	Holds the SQL statement used to insert rows into the dataset.
R	InternalPrepared (see page 3640)	Indicates whether the dataset has prepared its queries for execution.
	ModifySQL (see page 3641)	Provides the ability to access the SQL object encapsulating the ModifySQL statement.
	ParamCheck (see page 3642)	Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.
R	Params (see page 3643)	Specifies the parameters to use for any parameterized queries.
R	QDelete (see page 3643)	Provides the ability to directly access the SQL object encapsulating the DeleteSQL (see page 3639) statement.
R	QInsert (see page 3644)	Provides the ability to directly access the SQL object encapsulating the InsertSQL (see page 3640) statement.
R	QModify (see page 3644)	Provides the ability to directly access the SQL object encapsulating the ModifySQL (see page 3641) statement.
R	QRefresh (see page 3644)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL (see page 3645) statement.
R	QSelect (see page 3644)	Provides the ability to directly access the SQL object encapsulating the SelectSQL (see page 3645) statement.
	RefreshSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.
	SelectSQL (see page 3645)	Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.
R	SelectStmtHandle (see page 3645)	Gets the handle for the SELECT statement.
R	SQLParams (see page 3646)	Specifies the parameters to use for any parameterized queries.
R	StatementType (see page 3646)	Returns the statement type of the QSelect (see page 3644) query.
	UpdateMode (see page 3647)	Indicates how records are located when updating records.

Public Methods

	Name	Description
	AddIndex (see page 4148)	Creates a new index for the table.
	ApplyUpdates (see page 3609)	Writes a dataset's pending cached updates to the database.
V	BookmarkValid (see page 3610)	Indicates the validity of a specified bookmark.

	CachedUpdateStatus (see page 3610)	Returns the status of the cached updates.
	CancelUpdates (see page 3610)	Clears all pending cached updates from the cache.
V	CompareBookmarks (see page 3611)	Compares two bookmarks.
V	Create (see page 4147)	Creates an instance of a table component.
V	CreateBlobStream (see page 3612)	Returns a TIBDSBlobStream (see page 3678) object for reading or writing the data in a specified blob field.
	CreateTable (see page 4148)	Builds a new table using new structure information.
	Current (see page 3613)	Returns the descriptor area that holds information about the current query.
	DeleteIndex (see page 4149)	Deletes an index for the table.
	DeleteTable (see page 4150)	Deletes an existing database table.
V	Destroy (see page 4147)	Destroys the instance of a table.
	EmptyTable (see page 4150)	Deletes all records from the table.
	FetchAll (see page 3614)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
V	GetCurrentRecord (see page 3616)	Retrieves the current record into a buffer.
V	GetDetailLinkFields (see page 4151)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 3616)	This is the overview for the GetFieldData method overload.
	GetIndexNames (see page 4151)	Retrieves a list of available indexes for a table.
	GotoCurrent (see page 4152)	Synchronizes the current record for this table with the current record of a specified table component.
V	IsSequenced (see page 3623)	IsSequenced indicates whether sequence numbers are available for database records.
V	Locate (see page 4153)	Searches the table for a specified record and makes that record the current record.
	LocateNext (see page 3625)	Searches the dataset for the record after a specified record and makes that record the current record.
V	Lookup (see page 3625)	Retrieves field values from a record that matches specified search values.
V	Post (see page 3626)	Writes a modified record to the database server.
	RecordModified (see page 3629)	Sets the record to modified or unmodified.
	RevertRecord (see page 3629)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	SQLType (see page 3631)	Returns the type of query to be executed.
	Undelete (see page 3632)	Restores a record deleted from the dataset.
V	UpdateStatus (see page 3632)	Reports the update status for the current record.

Public Properties

	Name	Description
R	CurrentDBKey (see page 4158)	Returns the DBKey for the current row in the table.
R	DBHandle (see page 3639)	Specifies the database handle for the dataset.
R	Exists (see page 4158)	Indicates whether the underlying database table exists.
R	IndexFieldCount (see page 4160)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 4161)	Retrieves or sets a field for an index.
	LiveMode (see page 4161)	Indicates what types of edits, if any, can be made to the data in the dataset.
R	Plan (see page 3643)	Indicates the PLAN portion of the dataset's SELECT query.
R	RowsAffected (see page 3645)	Indicates the number of rows operated on by the latest query execution.
R	TableNames (see page 4164)	Returns a list of table names.
R	TRHandle (see page 3646)	Specifies the transaction handle for the dataset.
	UpdateRecordTypes (see page 3648)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 3648)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 4165)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 4165)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 4166)	Occurs immediately before the connection to the associated database is closed.

	BeforeTransactionEnd (see page 4166)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 4166)	Occurs when the database associated with this dataset is freed.
	OnUpdateError (see page 3650)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 3651)	Occurs when cached updates are applied to a record.
	TransactionFree (see page 4166)	Occurs when the transaction that includes this dataset's query is freed.

Published Properties

	Name	Description
	Active (see page 4157)	This is Active, a member of class TIBTable.
	AfterCancel (see page 3634)	This is AfterCancel, a member of class TIBCustomDataSet.
	AfterClose (see page 3634)	This is AfterClose, a member of class TIBCustomDataSet.
	AfterDelete (see page 3634)	This is AfterDelete, a member of class TIBCustomDataSet.
	AfterEdit (see page 3634)	This is AfterEdit, a member of class TIBCustomDataSet.
	AfterInsert (see page 3635)	This is AfterInsert, a member of class TIBCustomDataSet.
	AfterOpen (see page 3635)	This is AfterOpen, a member of class TIBCustomDataSet.
	AfterPost (see page 3635)	This is AfterPost, a member of class TIBCustomDataSet.
	AfterRefresh (see page 3635)	This is AfterRefresh, a member of class TIBCustomDataSet.
	AfterScroll (see page 3635)	This is AfterScroll, a member of class TIBCustomDataSet.
	AutoCalcFields (see page 3636)	This is AutoCalcFields, a member of class TIBCustomDataSet.
	BeforeCancel (see page 3636)	This is BeforeCancel, a member of class TIBCustomDataSet.
	BeforeClose (see page 3636)	This is BeforeClose, a member of class TIBCustomDataSet.
	BeforeDelete (see page 3636)	This is BeforeDelete, a member of class TIBCustomDataSet.
	BeforeEdit (see page 3637)	This is BeforeEdit, a member of class TIBCustomDataSet.
	BeforeInsert (see page 3637)	This is BeforeInsert, a member of class TIBCustomDataSet.
	BeforeOpen (see page 3637)	This is BeforeOpen, a member of class TIBCustomDataSet.
	BeforePost (see page 3637)	This is BeforePost, a member of class TIBCustomDataSet.
	BeforeRefresh (see page 3637)	This is BeforeRefresh, a member of class TIBCustomDataSet.
	BeforeScroll (see page 3638)	This is BeforeScroll, a member of class TIBCustomDataSet.
	BufferChunks (see page 4157)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 4157)	Specifies whether cached updates are enabled for a dataset.
	Constraints (see page 4158)	This is Constraints, a member of class TIBTable.
	Database (see page 3639)	Identifies the database component for which this dataset represents one or more tables.
	DefaultIndex (see page 4158)	Specifies if the data in the table should be ordered on a default index when opened.
	FieldDefs (see page 4159)	This is FieldDefs, a member of class TIBTable.
	Filter (see page 4159)	Specifies rows in a dataset that meet the filter conditions.
	Filtered (see page 4159)	Specifies whether or not filtering is active for a table.
	ForcedRefresh (see page 3639)	Specifies whether the dataset should refresh its data every time it posts a record.
	IndexDefs (see page 4160)	Contains information about the indexes for a table.
	IndexFieldNames (see page 4160)	Lists the columns to use as an index for a table.
	IndexName (see page 4161)	Identifies an index for the table.
	MasterFields (see page 4162)	Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.
	MasterSource (see page 4162)	Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.
	ObjectView (see page 3641)	This is ObjectView, a member of class TIBCustomDataSet.
	OnCalcFields (see page 3641)	This is OnCalcFields, a member of class TIBCustomDataSet.
	OnDeleteError (see page 3642)	This is OnDeleteError, a member of class TIBCustomDataSet.
	OnEditError (see page 3642)	This is OnEditError, a member of class TIBCustomDataSet.
	OnFilterRecord (see page 4162)	This is OnFilterRecord, a member of class TIBTable.
	OnNewRecord (see page 3642)	This is OnNewRecord, a member of class TIBCustomDataSet.
	OnPostError (see page 3642)	This is OnPostError, a member of class TIBCustomDataSet.
	ReadOnly (see page 4163)	Specifies whether a table is read-only for this application.

	StoreDefs (see page 4163)	Indicates whether the table's field and index definitions persist with the data module or form.
	TableName (see page 4163)	Indicates the name of the database table or view that this component encapsulates.
	TableTypes (see page 4164)	Sets the types of relations displayed in the TableName (see page 4163) drop-down list.
	Transaction (see page 3646)	Identifies the transaction under which the query executes.
	UniDirectional (see page 4164)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 4165)	Specifies the update object component used to update a read-only result set.

31.1.1.2 TIBTable.Create Constructor

Creates an instance of a table component.

Pascal

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

C++

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

Description

Call Create to instantiate a table declared in an application if it was not placed on a form at design time. Create calls its inherited Create constructor, creates an empty index definitions list, creates an empty data link, and creates an empty list of index files.

31.1.1.3 TIBTable.Destroy Destructor

Destroys the instance of a table.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBTable();
```

Description

Do not call Destroy directly. Instead call Free to verify that the table is not already freed before calling Destroy. Destroy frees the index files list for the table, frees its data link, frees its index definitions, and then calls its inherited Destroy destructor.

31.1.1.4 TIBTable Methods

The methods of the TIBTable class are listed here.

Protected Methods

	Name	Description
	DataEvent (see page 4149)	This is DataEvent, a member of class TIBTable.
	DefChanged (see page 4149)	This is DefChanged, a member of class TIBTable.
	DoOnNewRecord (see page 4150)	This is DoOnNewRecord, a member of class TIBTable.
	GetCanModify (see page 4150)	This is GetCanModify, a member of class TIBTable.
	GetDataSource (see page 4151)	This is GetDataSource, a member of class TIBTable.
	GetIndexParams (see page 4151)	This is GetIndexParams, a member of class TIBTable.
	InitFieldDefs (see page 4152)	This is InitFieldDefs, a member of class TIBTable.
	InternalClose (see page 4152)	This is InternalClose, a member of class TIBTable.

	InternalOpen (see page 4152)	This is InternalOpen, a member of class TIBTable.
	InternalRefresh (see page 4153)	This is InternalRefresh, a member of class TIBTable.
	InternalRefreshRow (see page 4153)	This is InternalRefreshRow, a member of class TIBTable.
	PSGetDefaultOrder (see page 4154)	This is PSGetDefaultOrder, a member of class TIBTable.
	PSGetIndexDefs (see page 4154)	This is PSGetIndexDefs, a member of class TIBTable.
	PSGetKeyFields (see page 4154)	This is PSGetKeyFields, a member of class TIBTable.
	PSGetTableName (see page 4154)	This is PSGetTableName, a member of class TIBTable.
	PSSetCommandText (see page 4155)	This is PSSetCommandText, a member of class TIBTable.
	PSSetParams (see page 4155)	This is PSSetParams, a member of class TIBTable.
	SetFiltered (see page 4155)	This is SetFiltered, a member of class TIBTable.
	SetFilterOptions (see page 4155)	This is SetFilterOptions, a member of class TIBTable.
	SetFilterText (see page 4156)	This is SetFilterText, a member of class TIBTable.
	UpdateIndexDefs (see page 4156)	This is UpdateIndexDefs, a member of class TIBTable.

Public Methods

	Name	Description
	AddIndex (see page 4148)	Creates a new index for the table.
	CreateTable (see page 4148)	Builds a new table using new structure information.
	DeleteIndex (see page 4149)	Deletes an index for the table.
	DeleteTable (see page 4150)	Deletes an existing database table.
	EmptyTable (see page 4150)	Deletes all records from the table.
	GetDetailLinkFields (see page 4151)	Lists the field components that link this dataset as a detail of a master dataset.
	GetIndexNames (see page 4151)	Retrieves a list of available indexes for a table.
	GotoCurrent (see page 4152)	Synchronizes the current record for this table with the current record of a specified table component.
	Locate (see page 4153)	Searches the table for a specified record and makes that record the current record.

31.1.1.4.1 TIBTable.AddIndex Method

Creates a new index for the table.

Pascal

```
procedure AddIndex(const Name: string; const Fields: string; Options: TIndexOptions; const
DescFields: string = '');
```

C++

```
__fastcall AddIndex(const UnicodeString Name, const UnicodeString Fields, TIndexOptions
Options, const UnicodeString DescFields = '');
```

Description

Call AddIndex to create a new index for the table associated with a dataset. The index created with this procedure is added to the database table underlying the dataset. Name is the name of the new index. Fields is a semicolon-delimited list of the fields to include in the index. Options is a potentially restricted set attributes for the index. It can specify that an index

- Represents the primary index for a dataset. (ixPrimary)
- Contains no duplicate values (ixUnique).
- Sorts records in descending order (isDescending).

Warning: Attempting to create an index using options that are not applicable to the table raises an exception.

31.1.1.4.2 TIBTable.CreateTable Method

Builds a new table using new structure information.

Pascal

```
procedure CreateTable;
```

C++

```
__fastcall CreateTable();
```

Description

Call CreateTable at runtime to create a table using this dataset's current definition. CreateTable overwrites an existing table's structure and data; to avoid overwriting an existing table, check Exists (see page 4158) before calling CreateTable.

If the FieldDefs (see page 4159) property contains values, these values are used to create field definitions. Otherwise the Fields property is used. One or both of these properties must contain values in order to recreate a dataset.

If the IndexDefs (see page 4160) property contain values, these values are used to create index definitions for the dataset.

31.1.1.4.3 TIBTable.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TIBTable.

31.1.1.4.4 TIBTable.DefChanged Method

Pascal

```
procedure DefChanged(Sender: TObject); override;
```

C++

```
virtual __fastcall DefChanged(TObject * Sender);
```

Description

This is DefChanged, a member of class TIBTable.

31.1.1.4.5 TIBTable.DeleteIndex Method

Deletes an index for the table.

Pascal

```
procedure DeleteIndex(const Name: string);
```

C++

```
__fastcall DeleteIndex(const UnicodeString Name);
```

Description

Call DeleteIndex to remove an index for a table. Name is the name of the index to delete. DeleteIndex cannot remove an index used by a constraint.

31.1.1.4.6 TIBTable.DeleteTable Method

Deletes an existing database table.

Pascal

```
procedure DeleteTable;
```

C++

```
__fastcall DeleteTable();
```

Description

Call DeleteTable to delete an existing database table associated with the table component through its Database and TableName (see page 4163) properties. A table must be closed before it can be deleted.

Warning: Deleting a table erases any data the table contains and destroys the table's structure information.

31.1.1.4.7 TIBTable.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; override;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TIBTable.

31.1.1.4.8 TIBTable.EmptyTable Method

Deletes all records from the table.

Pascal

```
procedure EmptyTable;
```

C++

```
__fastcall EmptyTable();
```

Description

The EmptyTable method deletes all records from the database table specified by the Database and TableName (see page 4163) properties.

Note: Deletion of records can fail if the user lacks sufficient privileges to perform the delete operation.

31.1.1.4.9 TIBTable.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TIBTable.

31.1.1.4.10 TIBTable.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TIBTable.

31.1.1.4.11 TIBTable.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLinkFields fills two lists of TFields that define a master-detail relationship between this table and another (master) dataset. The MasterFields (see page 4162) list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from the calling dataset.

31.1.1.4.12 TIBTable.GetIndexNames Method

Retrieves a list of available indexes for a table.

Pascal

```
procedure GetIndexNames(List: TStrings);
```

C++

```
__fastcall GetIndexNames(TStrings List);
```

Description

Call GetIndexNames to retrieve a list of all available indexes for a table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

See Also

Obtaining Information About Indexes

31.1.1.4.13 TIBTable.GetIndexParams Method

Pascal

```
procedure GetIndexParams(const IndexName: string; FieldsIndex: Boolean; var IndexedName:
string);
```

C++

```
__fastcall GetIndexParams(const UnicodeString IndexName, Boolean FieldsIndex, UnicodeString
IndexedName);
```

Description

This is GetIndexParams, a member of class TIBTable.

31.1.1.4.14 TIBTable.GotoCurrent Method

Synchronizes the current record for this table with the current record of a specified table component.

Pascal

```
procedure GotoCurrent(Table: TIBTable);
```

C++

```
__fastcall GotoCurrent(TIBTable Table);
```

Description

Call GotoCurrent to synchronize the cursor position for this table based on the cursor position in another dataset that uses a different data source component, but which is connected to the same underlying database table. Table is the name of the table component whose cursor position to use for synchronizing.

Note: This procedure works only for table components that have the same Database and TableName (see page 4163) properties. Otherwise an exception is raised.

GotoCurrent is mainly for use in applications that have two table components that are linked to the same underlying database table through different data source components. It enables an application to ensure that separate views of the data appear to be linked.

31.1.1.4.15 TIBTable.InitFieldDefs Method

Pascal

```
procedure InitFieldDefs; override;
```

C++

```
virtual __fastcall InitFieldDefs();
```

Description

This is InitFieldDefs, a member of class TIBTable.

31.1.1.4.16 TIBTable.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TIBTable.

31.1.1.4.17 TIBTable.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TIBTable.

31.1.1.4.18 TIBTable.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TIBTable.

31.1.1.4.19 TIBTable.InternalRefreshRow Method

Pascal

```
procedure InternalRefreshRow; override;
```

C++

```
virtual __fastcall InternalRefreshRow();
```

Description

This is InternalRefreshRow, a member of class TIBTable.

31.1.1.4.20 TIBTable.Locate Method

Searches the table for a specified record and makes that record the current record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call Locate to search a dataset for a specific record and position the cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. For example:

```
        with CustTable do
    Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']),
[loPartialKey]);
       TLocateOptions Opts;
    Opts.Clear();
    Opts << loPartialKey;
```

```
Variant locvalues[3];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
locvalues[2] = Variant("408-431-1000");
CustTable->Locate("Company;Contact;Phone", VarArrayOf(locvalues, 2), Opts);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns true if it finds a matching record, and makes that record the current one. Otherwise Locate returns false.

Locate uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, Locate uses the index. Otherwise Locate creates a filter for the search.

31.1.1.4.21 TIBTable.PSGetDefaultOrder Method

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

This is PSGetDefaultOrder, a member of class TIBTable.

31.1.1.4.22 TIBTable.PSGetIndexDefs Method

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

This is PSGetIndexDefs, a member of class TIBTable.

31.1.1.4.23 TIBTable.PSGetKeyFields Method

Pascal

```
function PSGetKeyFields: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetKeyFields();
```

Description

This is PSGetKeyFields, a member of class TIBTable.

31.1.1.4.24 TIBTable.PSGetTableName Method

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetTableName();
```

Description

This is PSGetTableName, a member of class TIBTable.

31.1.1.4.25 TIBTable.PSSetCommandText Method

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const UnicodeString CommandText);
```

Description

This is PSSetCommandText, a member of class TIBTable.

31.1.1.4.26 TIBTable.PSSetParams Method

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

This is PSSetParams, a member of class TIBTable.

31.1.1.4.27 TIBTable.SetFiltered Method

Pascal

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

C++

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

Description

This is SetFiltered, a member of class TIBTable.

31.1.1.4.28 TIBTable.SetFilterOptions Method

Pascal

```
procedure SetFilterOptions(Value: TFilterOptions); override;
```

C++

```
virtual __fastcall SetFilterOptions(TFilterOptions Value);
```

Description

This is SetFilterOptions, a member of class TIBTable.

31.1.1.4.29 TIBTable.SetFilterText Method

Pascal

```
procedure SetFilterText(const Value: string); override;
```

C++

```
virtual __fastcall SetFilterText(const UnicodeString Value);
```

Description

This is SetFilterText, a member of class TIBTable.

31.1.1.4.30 TIBTable.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TIBTable.

31.1.1.5 TIBTable Properties

The properties of the TIBTable class are listed here.

Public Properties

	Name	Description
R	CurrentDBKey (see page 4158)	Returns the DBKey for the current row in the table.
R	Exists (see page 4158)	Indicates whether the underlying database table exists.
R	IndexFieldCount (see page 4160)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 4161)	Retrieves or sets a field for an index.
	LiveMode (see page 4161)	Indicates what types of edits, if any, can be made to the data in the dataset.
R	TableNames (see page 4164)	Returns a list of table names.

Published Properties

	Name	Description
	Active (see page 4157)	This is Active, a member of class TIBTable.
	BufferChunks (see page 4157)	Specifies the dataset buffer chunk size.
	CachedUpdates (see page 4157)	Specifies whether cached updates are enabled for a dataset.
	Constraints (see page 4158)	This is Constraints, a member of class TIBTable.
	DefaultIndex (see page 4158)	Specifies if the data in the table should be ordered on a default index when opened.
	FieldDefs (see page 4159)	This is FieldDefs, a member of class TIBTable.
	Filter (see page 4159)	Specifies rows in a dataset that meet the filter conditions.
	Filtered (see page 4159)	Specifies whether or not filtering is active for a table.
	IndexDefs (see page 4160)	Contains information about the indexes for a table.
	IndexFieldNames (see page 4160)	Lists the columns to use as an index for a table.
	IndexName (see page 4161)	Identifies an index for the table.
	MasterFields (see page 4162)	Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.

	MasterSource (see page 4162)	Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.
	OnFilterRecord (see page 4162)	This is OnFilterRecord, a member of class TIBTable.
	ReadOnly (see page 4163)	Specifies whether a table is read-only for this application.
	StoreDefs (see page 4163)	Indicates whether the table's field and index definitions persist with the data module or form.
	TableName (see page 4163)	Indicates the name of the database table or view that this component encapsulates.
	TableTypes (see page 4164)	Sets the types of relations displayed in the TableName (see page 4163) drop-down list.
	UniDirectional (see page 4164)	Determines whether bidirectional cursors are enabled for the dataset.
	UpdateObject (see page 4165)	Specifies the update object component used to update a read-only result set.

31.1.1.5.1 TIBTable.Active Property

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

This is Active, a member of class TIBTable.

31.1.1.5.2 TIBTable.BufferChunks Property

Specifies the dataset buffer chunk size.

Pascal

```
property BufferChunks: Integer;
```

C++

```
__property int BufferChunks;
```

Description

Use BufferChunks to get or set the number of records for which the dataset allocates buffer space at any time. When the dataset's buffer is full, trying to fetch an additional record causes the dataset to reallocate the buffer so that it has enough memory to hold an additional BufferChunks records.

Note: When Unidirectional is true, BufferChunks has no meaning.

31.1.1.5.3 TIBTable.CachedUpdates Property

Specifies whether cached updates are enabled for a dataset.

Pascal

```
property CachedUpdates: Boolean;
```

C++

```
__property Boolean CachedUpdates;
```

Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is true, cached updates are enabled. If CachedUpdates is false, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Note: **Note**: Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset.

31.1.1.5.4 TIBTable.Constraints Property

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

This is Constraints, a member of class TIBTable.

31.1.1.5.5 TIBTable.CurrentDBKey Property

Returns the DBKey for the current row in the table.

Pascal

```
property CurrentDBKey: TIBDBKey;
```

C++

```
__property TIBDBKey CurrentDBKey;
```

Description

Use CurrentDBKey to return the DBKey for the current row in the table. A DBKey is a unique row identifier for the duration of the current transaction. This property is primarily for internal use.

31.1.1.5.6 TIBTable.DefaultIndex Property

Specifies if the data in the table should be ordered on a default index when opened.

Pascal

```
property DefaultIndex: Boolean;
```

C++

```
__property Boolean DefaultIndex;
```

Description

When this property is set to false, an ORDER BY clause is not used when opening a table on the server. When DefaultIndex is true, the data is ordered based on the primary key or a unique index when opening the table. DefaultIndex defaults to true.

31.1.1.5.7 TIBTable.Exists Property

Indicates whether the underlying database table exists.

Pascal

```
property Exists: Boolean;
```

C++

```
__property Boolean Exists;
```

Description

Read Exists at runtime to determine whether a database table exists. If the table does not exist, you can create a table from the field definitions and index definitions using the CreateTable (see page 4148) method. This property is read-only.

31.1.1.5.8 TIBTable.FieldDefs Property

Pascal

```
property FieldDefs;
```

C++

```
__property FieldDefs;
```

Description

This is FieldDefs, a member of class TIBTable.

31.1.1.5.9 TIBTable.Filter Property

Specifies rows in a dataset that meet the filter conditions.

Pascal

```
property Filter;
```

C++

```
__property Filter;
```

Description

Use the Filter property to restrict the rows in the dataset to those that meet the filter conditions. The syntax for Filter is the same as the search condition that appears after a WHERE clause in a Select statement. For example, to view only those records where the value in the Country field contains 'France' or 'Fiji':

Country = 'France' or Country = 'Fiji'

You can use standard SQL wildcards such as percent (%) and underscore (_) in the condition when you use the LIKE operator. The following filter condition retrieves all Countries beginning with 'F':

```
Country LIKE 'F%'
```

To view rows that have a NULL value in the Country column and Contact_Name is not NULL, use the IS operator:

```
Country is NULL and Contact_Name is not NULL
```

You can also use complex expressions in filter clauses. The following example retrieves rows containing Country values that use Francs as currency.

```
Country IN (SELECT Country from Country where Currency = '_Franc')
```

31.1.1.5.10 TIBTable.Filtered Property

Specifies whether or not filtering is active for a table.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter (see page 4159) property or the OnFilterRecord (see page 4162) event handler, set Filtered to true.

31.1.1.5.11 TIBTable.IndexDefs Property

Contains information about the indexes for a table.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

IndexDefs is an array of index items, each of which describes an available index for the table. Ordinarily an application accesses or specifies indexes through the IndexFieldNames (see page 4160) or IndexFields (see page 4161) properties. If IndexDefs is updated or manually edited, the StoreDefs (see page 4163) property becomes true.

Note: The index items in IndexDefs may not always reflect the current indexes available for a table. Before examining IndexDefs, call its Update method to refresh the item list.

31.1.1.5.12 TIBTable.IndexFieldCount Property

Indicates the number of fields that comprise the current key.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Examine IndexFieldCount to determine the number of fields that comprise the current key. For indexes based on a single column, IndexFieldCount returns 1. For multi-column indexes, IndexFieldCount indicates the number of fields upon which the index is based.

31.1.1.5.13 TIBTable.IndexFieldNames Property

Lists the columns to use as an index for a table.

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property UnicodeString IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a table. In IndexFieldNames specify the name

of each column to use as an index for a table. Ordering of column names is significant. Separate names with semicolon.

Note: The IndexFieldNames and IndexName (see page 4161) properties are mutually exclusive. Setting one clears the other.

See Also

Creating an Index with IndexFieldNames

31.1.1.5.14 TIBTable.IndexFields Property

Retrieves or sets a field for an index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields provides a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexFields directly. Instead use the IndexFieldNames (see page 4160) property to order datasets on the fly at runtime.

31.1.1.5.15 TIBTable.IndexName Property

Identifies an index for the table.

Pascal

```
property IndexName: string;
```

C++

```
__property UnicodeString IndexName;
```

Description

Use IndexName to specify an index for a table. If IndexName is empty, a table's sort order is based on its primary index.

If IndexName contains a valid index name, then that index is used to determine the order of records.

Note: IndexFieldNames (see page 4160) and IndexName are mutually exclusive. Setting one clears the other.

31.1.1.5.16 TIBTable.LiveMode Property

Indicates what types of edits, if any, can be made to the data in the dataset.

Pascal

```
property LiveMode: TLiveModes;
```

C++

```
__property TLiveModes LiveMode;
```

Description

Read LiveMode to determine what types of edits should be allowed. LiveMode is set automatically to restrict the capabilities of the dataset to reflect the user's rights.

Note: For read-only datasets, such as TIBQuery, LiveMode is always an empty set.

31.1.1.5.17 TIBTable.MasterFields Property

Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.

Pascal

```
property MasterFields: string;
```

C++

```
__property UnicodeString MasterFields;
```

Description

Use MasterFields after setting the MasterSource (see page 4162) property to specify the names of one or more fields to use in establishing a detail-master relationship between this table and the one specified in MasterSource (see page 4162).

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

See Also

Making the Table a Detail of Another Dataset

31.1.1.5.18 TIBTable.MasterSource Property

Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields (see page 4162) property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

See Also

Making the Table a Detail of Another Dataset

31.1.1.5.19 TIBTable.OnFilterRecord Property

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

This is OnFilterRecord, a member of class TIBTable.

31.1.1.5.20 TIBTable.ReadOnly Property

Specifies whether a table is read-only for this application.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the table. By default, ReadOnly is false, meaning users can potentially alter a table's data.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a table. Other factors, such as insufficient SQL privileges for the application or its current user may prevent successful alterations.

To guarantee that users cannot modify or add data to a table,

1. **1** Set the Active (see page 4157) property to false.
2. **2** Set ReadOnly to true.

31.1.1.5.21 TIBTable.StoreDefs Property

Indicates whether the table's field and index definitions persist with the data module or form.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

If StoreDefs is true, the table's index and field definitions are stored with the data module or form. Setting StoreDefs to true makes the CreateTable (see page 4148) method into a one-step procedure that creates fields, indexes, and validity checks at runtime.

StoreDefs is false by default. It becomes true whenever FieldDefs (see page 4159) or IndexDefs (see page 4160) is updated or edited manually; to prevent edited (or imported) definitions from being stored, reset StoreDefs to false.

31.1.1.5.22 TIBTable.TableName Property

Indicates the name of the database table or view that this component encapsulates.

Pascal

```
property TableName: String;
```

C++

```
__property UnicodeString TableName;
```

Description

Use TableName to specify the name of the database relation this component encapsulates. To set TableName to a meaningful value, the Database property should already be set. If Database is set at design time, then select a valid table name from the TableName drop-down list in the Object Inspector.

Note: To set TableName, the Active (see page 4157) property must be false.

Tip: Setting TableName clears the FieldDefs (see page 4159) property. Therefore, if you are using TIBTable (see page 4142) to create a new table, set the value of TableName before you assign any field definitions.

31.1.1.5.23 TIBTable.TableNames Property

Returns a list of table names.

Pascal

```
property TableNames: TStrings;
```

C++

```
__property TStrings TableNames;
```

Description

The TableNames property is an internal property used to display a list of the table and view names in the database.

31.1.1.5.24 TIBTable.TableTypes Property

Sets the types of relations displayed in the TableName (see page 4163) drop-down list.

Pascal

```
property TableTypes: TIBTableTypes;
```

C++

```
__property TIBTableTypes TableTypes;
```

Description

Use TableTypes to change which types of relations are displayed in the TableName (see page 4163) drop-down list, in addition to user tables.

31.1.1.5.25 TIBTable.UniDirectional Property

Determines whether bidirectional cursors are enabled for the dataset.

Pascal

```
property UniDirectional: Boolean;
```

C++

```
__property Boolean UniDirectional;
```

Description

Use UniDirectional to determine whether bidirectional cursors are enabled for this dataset.

Note: Do not confuse the UniDirectional property with the IsUniDirectional property inherited from TDataSet. IsUniDirectional indicates whether the dataset is a dbExpress dataset, which, in addition to using a unidirectional cursor, does not buffer records.

UniDirectional is only concerned with whether the cursor permits backward or random navigation.

31.1.1.5.26 TIBTable.UpdateObject Property

Specifies the update object component used to update a read-only result set.

Pascal

```
property UpdateObject: TIBDataSetUpdateObject;
```

C++

```
__property TIBDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TIBDataSetUpdateObject component to use in an application that must be able to update a read-only result set.

In some cases, such as when using TIBQuery, the dataset does not automatically update records. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

TIBDataSetUpdateObject is especially useful for complicated updates from an OnUpdateRecord event.

31.1.1.6 TIBTable Events

The events of the TIBTable class are listed here.

Published Events

	Name	Description
	AfterDatabaseDisconnect (see page 4165)	Occurs immediately after the connection to the associated database is closed.
	AfterTransactionEnd (see page 4165)	Occurs immediately after a transaction that includes this dataset's query ends.
	BeforeDatabaseDisconnect (see page 4166)	Occurs immediately before the connection to the associated database is closed.
	BeforeTransactionEnd (see page 4166)	Occurs immediately before a transaction that includes this dataset's query ends.
	DatabaseFree (see page 4166)	Occurs when the database associated with this dataset is freed.
	TransactionFree (see page 4166)	Occurs when the transaction that includes this dataset's query is freed.

31.1.1.6.1 TIBTable.AfterDatabaseDisconnect Event

Occurs immediately after the connection to the associated database is closed.

Pascal

```
property AfterDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDatabaseDisconnect;
```

Description

Write an AfterDatabaseDisconnect event handler to respond when the dataset closes its connection to the database specified by the Database property.

31.1.1.6.2 TIBTable.AfterTransactionEnd Event

Occurs immediately after a transaction that includes this dataset's query ends.

Pascal

```
property AfterTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTransactionEnd;
```

Description

Write an AfterTransactionEnd event handler to respond when the transaction specified by the Transaction property ends.

31.1.1.6.3 TIBTable.BeforeDatabaseDisconnect Event

Occurs immediately before the connection to the associated database is closed.

Pascal

```
property BeforeDatabaseDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDatabaseDisconnect;
```

Description

Write a BeforeDatabaseDisconnect event handler to respond when the dataset is about to close its connection to the database specified by the Database property.

31.1.1.6.4 TIBTable.BeforeTransactionEnd Event

Occurs immediately before a transaction that includes this dataset's query ends.

Pascal

```
property BeforeTransactionEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTransactionEnd;
```

Description

Write a BeforeTransactionEnd event handler to respond when the transaction specified by the Transaction property is about to end.

31.1.1.6.5 TIBTable.DatabaseFree Event

Occurs when the database associated with this dataset is freed.

Pascal

```
property DatabaseFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent DatabaseFree;
```

Description

Write a DatabaseFree event handler to respond when the database specified by the Database property is freed.

31.1.1.6.6 TIBTable.TransactionFree Event

Occurs when the transaction that includes this dataset's query is freed.

Pascal

```
property TransactionFree: TNotifyEvent;
```

C++

```
__property TNotifyEvent TransactionFree;
```

Description

Write a TransactionFree event handler to respond when the transaction specified by the Transaction property is freed.

31.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TIBTableType (see page 4168)	This is record IBTable.TIBTableType.

31.2.1 IBTable.TIBTableType Enumeration

Pascal

```
TIBTableType = (
  ttSystem,
  ttView
);
```

C++

```
enum TIBTableType {
  ttSystem,
  ttView
};
```

File

IBTable

Description

This is record IBTable.TIBTableType.

31.3 Types

The following table lists types in this documentation.

Types

Name	Description
TIBTableTypes (see page 4169)	TIBTableTypes and TIBTableTypes indicate the types of InterBase tables and views other than user tables.
TIndexName (see page 4169)	This is type IBTable.TIndexName.

31.3.1 IBTable.TIBTableTypes Type

TIBTableTypes and TIBTableTypes indicate the types of InterBase tables and views other than user tables.

Pascal

```
TIBTableTypes = set of TIBTableType;
```

C++

```
TIBTableType TIBTableTypes;
```

File

IBTable

Description

TIBTableTypes is a set of TIBTableType (see page 4168) values. The following table lists the possible values:

Value Meaning

ttSystem System tables and views.

ttView User views.

31.3.2 IBTable.TIndexName Type

Pascal

```
TIndexName = String;
```

C++

```
UnicodeString TIndexName;
```

File

IBTable

Description

This is type IBTable.TIndexName.

32 IBUpdateSQL Namespace

This is namespace IBUpdateSQL.

Topics

Name	Description
Classes (see page 4172)	The following table lists classes in this documentation.

Classes

	Name	Description
	TIBUpdateSQL (see page 4172)	TIBUpdateSQL provides an object for updating read-only datasets when cached updates are enabled.

32.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIBUpdateSQL (see page 4172)	TIBUpdateSQL provides an object for updating read-only datasets when cached updates are enabled.

32.1.1 TIBUpdateSQL Class

TIBUpdateSQL provides an object for updating read-only datasets when cached updates are enabled.

Class Hierarchy

Pascal

```
TIBUpdateSQL = class(TIBDataSetUpdateObject);
```

C++

```
class TIBUpdateSQL : public TIBDataSetUpdateObject;
```

File

IBUpdateSQL

Description

Use a TIBUpdateSQL object to provide SQL (see page 4179) statements used to update read-only datasets represented by TIBQuery (see page 3916) components when cached updates are enabled. A dataset is read-only either by design or circumstance. If a dataset is read-only by design, the application itself does not provide a user interface for updating data, but may institute a programmatic scheme behind the scenes.

TIBUpdateSQL provides a mechanism for circumventing what some developers consider an SQL (see page 4179)-92 limitation. It enables a developer to provide INSERT, UPDATE, DELETE, and REFRESH statements for performing separate update queries on otherwise read-only result sets in such a manner that the separate update queries are transparent to the end user.

In practical application, a TIBUpdateSQL object is placed on a data module or form, and linked to a TIBQuery (see page 3916) component through that component's UpdateObject property. If the UpdateObject property points to a valid TIBUpdateSQL object, the SQL (see page 4179) statements belonging to the update object are automatically applied when cached updates are applied.

See Also

Getting Started with InterBase Express

32.1.1.1 TIBUpdateSQL Members

The following tables list the members exposed by TIBUpdateSQL.

Protected Methods

	Name	Description
	GetDataSet (see page 4175)	This is GetDataSet, a member of class TIBUpdateSQL.
	GetSQL (see page 4175)	This is GetSQL, a member of class TIBUpdateSQL.
	SetDataSet (see page 4176)	This is SetDataSet, a member of class TIBUpdateSQL.
	SQLChanged (see page 4176)	This is SQLChanged, a member of class TIBUpdateSQL.

Public Methods

	Name	Description
	Apply (see page 4174)	Sets the parameters for a specified SQL (see page 4179) statement type, and executes the resulting statement.
	Create (see page 4173)	Creates an instance of an update object.
	Destroy (see page 4173)	Frees an instance of an update object.
	ExecSQL (see page 4175)	Executes a specified type of SQL (see page 4179) statement to perform an update for an otherwise read-only results set when cached updates is enabled.
	SetParams (see page 4176)	Binds parameters in an SQL (see page 4179) statement prior to statement execution.

Public Properties

	Name	Description
	DataSet (see page 4177)	Identifies the dataset to which a TIBUpdateSQL (see page 4172) component belongs.
	Query (see page 4178)	Returns the query object used to perform a specified kind of update.
	SQL (see page 4179)	Returns a specified SQL statement used when applying cached updates.

Published Properties

	Name	Description
	DeleteSQL (see page 4177)	Specifies the SQL (see page 4179) DELETE statement to use when applying a cached deletion of a record.
	InsertSQL (see page 4178)	Specifies the SQL (see page 4179) INSERT statement to use when applying a cached insertion of a record.
	ModifySQL (see page 4178)	Specifies the SQL (see page 4179) UPDATE statement to use when applying an update to a record and cached updates is enabled.
	RefreshSQL (see page 3677)	Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

32.1.1.2 TIBUpdateSQL.Create Constructor

Creates an instance of an update object.

Pascal

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

C++

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

Description

Call Create to instantiate an update object at run time. You do not need to call Create for update objects placed in a data module or form at design time. Delphi automatically handles these objects.

32.1.1.3 TIBUpdateSQL.Destroy Destructor

Frees an instance of an update object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIBUpdateSQL();
```

Description

Do not call Destroy directly in an application. Usually destruction of update objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not already freed before calling Destroy.

32.1.1.4 TIBUpdateSQL Methods

The methods of the TIBUpdateSQL class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 4175)	This is GetDataSet, a member of class TIBUpdateSQL.
	GetSQL (see page 4175)	This is GetSQL, a member of class TIBUpdateSQL.
	SetDataSet (see page 4176)	This is SetDataSet, a member of class TIBUpdateSQL.
	SQLChanged (see page 4176)	This is SQLChanged, a member of class TIBUpdateSQL.

Public Methods

	Name	Description
	Apply (see page 4174)	Sets the parameters for a specified SQL (see page 4179) statement type, and executes the resulting statement.
	ExecSQL (see page 4175)	Executes a specified type of SQL (see page 4179) statement to perform an update for an otherwise read-only results set when cached updates is enabled.
	SetParams (see page 4176)	Binds parameters in an SQL (see page 4179) statement prior to statement execution.

32.1.1.4.1 TIBUpdateSQL.Apply Method

Sets the parameters for a specified SQL (see page 4179) statement type, and executes the resulting statement.

Pascal

```
procedure Apply(UpdateKind: TUpdateKind); override;
```

C++

```
virtual __fastcall Apply(TUpdateKind UpdateKind);
```

Description

Call Apply to set parameters for an SQL (see page 4179) statement and execute it to update a record. UpdateKind indicates which SQL (see page 4179) statement to bind and execute, and can be one of the following values:

Value Meaning

ukModify Bind and execute the SQL (see page 4179) statement in the ModifySQL (see page 4178) property

ukInsert Bind and execute the SQL (see page 4179) statement in the InsertSQL (see page 4178) property

ukDelete Bind and execute the SQL (see page 4179) statement in the DeleteSQL (see page 4177) property

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

Note: If an SQL (see page 4179) statement does not contain parameters, it is more efficient to call ExecSQL (see page

4175) instead of Apply.

32.1.1.4.2 TIBUpdateSQL.ExecSQL Method

Executes a specified type of SQL (see page 4179) statement to perform an update for an otherwise read-only results set when cached updates is enabled.

Pascal

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

C++

```
__fastcall ExecSQL(TUpdateKind UpdateKind);
```

Description

Call ExecSQL to execute the SQL (see page 4179) statement necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute, and can be one of the following values:

Value Meaning

ukModify Execute the SQL (see page 4179) statement used to update records in the dataset

ukInsert Execute the SQL (see page 4179) statement used to insert new records into the dataset

ukDelete Execute the SQL (see page 4179) statement used to delete records in the dataset.

If the statement to execute contains any parameters, an application must call SetParams (see page 4176) to bind the parameters before calling ExecSQL. To determine if a statement contains parameters, examine the appropriate ModifySQL (see page 4178), InsertSQL (see page 4178), or DeleteSQL (see page 4177) property, depending on the statement type intended for execution.

Note: To both bind parameters and execute a statement, call Apply (see page 4174).

32.1.1.4.3 TIBUpdateSQL.GetDataSet Method

Pascal

```
function GetDataSet: TIBCustomDataSet; override;
```

C++

```
virtual __fastcall TIBCustomDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TIBUpdateSQL.

32.1.1.4.4 TIBUpdateSQL.GetSQL Method

Pascal

```
function GetSQL(UpdateKind: TUpdateKind): TStrings; override;
```

C++

```
virtual __fastcall TStrings GetSQL(TUpdateKind UpdateKind);
```

Description

This is GetSQL, a member of class TIBUpdateSQL.

32.1.1.4.5 TIBUpdateSQL.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TIBCustomDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TIBCustomDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TIBUpdateSQL.

32.1.1.4.6 TIBUpdateSQL.SetParams Method

Binds parameters in an SQL (see page 4179) statement prior to statement execution.

Pascal

```
procedure SetParams(UpdateKind: TUpdateKind);
```

C++

```
__fastcall SetParams(TUpdateKind UpdateKind);
```

Description

Call SetParams to bind parameters in an SQL (see page 4179) statement associated with the update object prior to executing the statement. UpdateKind indicates the type of statement for which to bind parameters, and can be one of the following values:

Value Meaning

ukModify Bind parameters for the SQL (see page 4179) statement used to update records

ukInsert Bind parameters for the SQL (see page 4179) statement used to insert new records

ukDelete Bind parameters for the SQL (see page 4179) statement used to delete records

Parameters are indicated in an SQL (see page 4179) statement by a colon. Except for the leading colon in the parameter name, the parameter name must exactly match the name of an existing field name for the dataset.

Note: Parameter names can be prefaced by the 'OLD_' indicator. If so, the old value of the field is used to perform the update instead of any updates in the cache.

32.1.1.4.7 TIBUpdateSQL.SQLChanged Method

Pascal

```
procedure SQLChanged(Sender: TObject);
```

C++

```
__fastcall SQLChanged(TObject * Sender);
```

Description

This is SQLChanged, a member of class TIBUpdateSQL.

32.1.1.5 TIBUpdateSQL Properties

The properties of the TIBUpdateSQL class are listed here.

Public Properties

	Name	Description
	DataSet (see page 4177)	Identifies the dataset to which a TIBUpdateSQL (see page 4172) component belongs.
R	Query (see page 4178)	Returns the query object used to perform a specified kind of update.
	SQL (see page 4179)	Returns a specified SQL statement used when applying cached updates.

Published Properties

	Name	Description
	DeleteSQL (see page 4177)	Specifies the SQL (see page 4179) DELETE statement to use when applying a cached deletion of a record.
	InsertSQL (see page 4178)	Specifies the SQL (see page 4179) INSERT statement to use when applying a cached insertion of a record.
	ModifySQL (see page 4178)	Specifies the SQL (see page 4179) UPDATE statement to use when applying an update to a record and cached updates is enabled.

32.1.1.5.1 TIBUpdateSQL.DataSet Property

Identifies the dataset to which a TIBUpdateSQL (see page 4172) component belongs.

Pascal

```
property DataSet: TIBCustomDataSet;
```

C++

```
__property TIBCustomDataSet DataSet;
```

Description

At design time, setting the dataset object's UpdateObject property automatically sets the DataSet property of the specified TIBUpdateSQL (see page 4172) object. An application should only need to set this property if it creates a new update component at run time.

32.1.1.5.2 TIBUpdateSQL.DeleteSQL Property

Specifies the SQL (see page 4179) DELETE statement to use when applying a cached deletion of a record.

Pascal

```
property DeleteSQL: TStrings;
```

C++

```
__property TStrings DeleteSQL;
```

Description

Set DeleteSQL to the SQL (see page 4179) DELETE statement to use when applying a deletion to a record. Statements can be parameterized queries. To create a DELETE statement at design time, use the UpdateSQL editor to create statements, such as:

```
delete from Employee
    where
    Emp_No = :OLD_Emp_No
```

At run time, an application can write a statement directly to this property to set or change the DELETE statement.

Note: As the example illustrates, DeleteSQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, use the field name with the prefix 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

32.1.1.5.3 TIBUpdateSQL.InsertSQL Property

Specifies the SQL (see page 4179) INSERT statement to use when applying a cached insertion of a record.

Pascal

```
property InsertSQL: TStrings;
```

C++

```
__property TStrings InsertSQL;
```

Description

Set InsertSQL to the SQL (see page 4179) INSERT statement to use when applying an insertion to a dataset. Statements can be parameterized queries. To create a INSERT statement at design time, use the UpdateSQL editor to create statements, such as:

```
insert into Country
    (Country, Currency)
    values (:Country, :Currency)
```

At run time, an application can write a statement directly to this property to set or change the INSERT statement.

32.1.1.5.4 TIBUpdateSQL.ModifySQL Property

Specifies the SQL (see page 4179) UPDATE statement to use when applying an update to a record and cached updates is enabled.

Pascal

```
property ModifySQL: TStrings;
```

C++

```
__property TStrings ModifySQL;
```

Description

Set ModifySQL to the SQL (see page 4179) UPDATE statement to use when applying an updated record to a dataset. Statements can be parameterized queries. To create a UPDATE statement at design time, use the UpdateSQL editor to create statements, such as:

```
update Employee
    set Last_Name = :Last_Name
    where Emp_No = :OLD_Emp_No
```

At run time, an application can write a statement directly to this property to set or change the UPDATE statement.

Note: As the example illustrates, ModifySQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

32.1.1.5.5 TIBUpdateSQL.Query Property

Returns the query object used to perform a specified kind of update.

Pascal

```
property Query [UpdateKind: TUpdateKind]: TIBQuery;
```

C++

```
__property TIBQuery Query[TUpdateKind UpdateKind];
```

Description

Query returns the TIBQuery (see page 3916) object used to perform a particular form of SQL (see page 4179) update. UpdateKind specifies which query object to retrieve. UpdateKind can be one of the following:

Value Meaning

ukModify Return the query object used to execute UPDATE statements

ukInsert Return the query object used to execute INSERT statements

ukDelete Return the query object used to execute DELETE statements

Each query object executes a particular kind of SQL (see page 4179) statement. The contents of the SQL (see page 4179) statements executed by these objects can be accessed directly using the ModifySQL (see page 4178), InsertSQL (see page 4178), and DeleteSQL (see page 4177) properties.

The main purpose of Query is to provide a way for an application to set the properties for an update query object or to call the query object's methods.

Note: If a particular kind of update statement is not provided, then its corresponding query object is nil (Delphi) or NULL (C++). For example, if an application does not provide an SQL (see page 4179) statement for the DeleteSQL (see page 4177) property, then Query[ukDelete] returns nil or NULL.

32.1.1.5.6 TIBUpdateSQL.SQL Property

Returns a specified SQL statement used when applying cached updates.

Pascal

```
property SQL [UpdateKind: TUpdateKind]: TStrings;
```

C++

```
__property TStrings SQL[TUpdateKind UpdateKind];
```

Description

Returns the SQL statement in the ModifySQL (see page 4178), InsertSQL (see page 4178), or DeleteSQL (see page 4177) property, depending on the setting of UpdateKind. UpdateKind can be any of the following:

Value Meaning

ukModify Return the SQL statement used to update records in the dataset

ukInsert Return the SQL statement used to insert new records into the dataset

ukDelete Return the SQL statement used to delete records in the dataset

33 IBUtils Namespace

This is namespace IBUtils.

Topics

Name	Description
Functions (see page 4182)	The following table lists functions in this documentation.
Constants (see page 4186)	The following table lists constants in this documentation.

Constants

Name	Description
CR (see page 4186)	This is constant IBUtils.CR.
CRLF (see page 4186)	This is constant IBUtils.CRLF.
LF (see page 4187)	This is constant IBUtils.LF.
NULL_TERMINATOR (see page 4187)	This is constant IBUtils.NULL_TERMINATOR.
TAB (see page 4187)	This is constant IBUtils.TAB.

Functions

	Name	Description
	AddIBParamSQLForDetail (see page 4182)	This is function IBUtils.AddIBParamSQLForDetail.
	ExtractIdentifier (see page 4182)	This is function IBUtils.ExtractIdentifier.
	FormatIdentifier (see page 4183)	This is function IBUtils.FormatIdentifier.
	FormatIdentifierValue (see page 4183)	This is function IBUtils.FormatIdentifierValue.
	Max (see page 4183)	Returns the greater of two numeric values (Delphi only).
	Min (see page 4184)	Returns the lesser of two numeric values (Delphi only).
	QuoteIdentifier (see page 4184)	This is function IBUtils.QuoteIdentifier.
	RandomInteger (see page 4184)	This is function IBUtils.RandomInteger.
	RandomString (see page 4185)	This is function IBUtils.RandomString.
	StripString (see page 4185)	This is function IBUtils.StripString.

33.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddIBParamSQLForDetail (see page 4182)	This is function IBUtils.AddIBParamSQLForDetail.
	ExtractIdentifier (see page 4182)	This is function IBUtils.ExtractIdentifier.
	FormatIdentifier (see page 4183)	This is function IBUtils.FormatIdentifier.
	FormatIdentifierValue (see page 4183)	This is function IBUtils.FormatIdentifierValue.
	Max (see page 4183)	Returns the greater of two numeric values (Delphi only).
	Min (see page 4184)	Returns the lesser of two numeric values (Delphi only).
	QuoteIdentifier (see page 4184)	This is function IBUtils.QuoteIdentifier.
	RandomInteger (see page 4184)	This is function IBUtils.RandomInteger.
	RandomString (see page 4185)	This is function IBUtils.RandomString.
	StripString (see page 4185)	This is function IBUtils.StripString.

33.1.1 IBUtils.AddIBParamSQLForDetail Function

Pascal

```
function AddIBParamSQLForDetail(Params: TParams; SQL: string; Native: Boolean; Dialect:
Integer): string;
```

C++

```
UnicodeString AddIBParamSQLForDetail(TParams Params, UnicodeString SQL, Boolean Native, int
Dialect);
```

File

IBUtils

Description

This is function IBUtils.AddIBParamSQLForDetail.

33.1.2 IBUtils.ExtractIdentifier Function

Pascal

```
function ExtractIdentifier(Dialect: Integer; Value: String): String;
```

C++

```
UnicodeString ExtractIdentifier(int Dialect, UnicodeString Value);
```

File

IBUtils

Description

This is function IBUtils.ExtractIdentifier.

33.1.3 IBUtils.FormatIdentifier Function

Pascal

```
function FormatIdentifier(Dialect: Integer; Value: String): String;
```

C++

```
UnicodeString FormatIdentifier(int Dialect, UnicodeString Value);
```

File

IBUtils

Description

This is function IBUtils.FormatIdentifier.

33.1.4 IBUtils.FormatIdentifierValue Function

Pascal

```
function FormatIdentifierValue(Dialect: Integer; Value: String): String;
```

C++

```
UnicodeString FormatIdentifierValue(int Dialect, UnicodeString Value);
```

File

IBUtils

Description

This is function IBUtils.FormatIdentifierValue.

33.1.5 IBUtils.Max Function

Returns the greater of two numeric values (Delphi only).

Pascal

```
function Max(n1: Integer; n2: Integer): Integer;
```

C++

```
int Max(int n1, int n2);
```

File

IBUtils

Description

In Delphi, call Max to compare two numeric values. Max returns the greater value of the two.

See Also

MaxValue (see page 4746)

Min (see page 4747)

MaxIntValue (see page 4745)

33.1.6 IBUtils.Min Function

Returns the lesser of two numeric values (Delphi only).

Pascal

```
function Min(n1: Integer; n2: Integer): Integer;
```

C++

```
int Min(int n1, int n2);
```

File

IBUtils

Description

Call Min to compare two numeric values in Delphi. Min returns the smaller value of the two.

See Also

MinValue (see page 4748)

Max (see page 4745)

MinIntValue (see page 4748)

33.1.7 IBUtils.QuoteIdentifier Function

Pascal

```
function QuoteIdentifier(Dialect: Integer; Value: String): String;
```

C++

```
UnicodeString QuoteIdentifier(int Dialect, UnicodeString Value);
```

File

IBUtils

Description

This is function IBUtils.QuoteIdentifier.

33.1.8 IBUtils.RandomInteger Function

Pascal

```
function RandomInteger(iLow: Integer; iHigh: Integer): Integer;
```

C++

```
int RandomInteger(int iLow, int iHigh);
```

File

IBUtils

Description

This is function IBUtils.RandomInteger.

33.1.9 IBUtils.RandomString Function

Pascal

```
function RandomString(iLength: Integer): String;
```

C++

```
UnicodeString RandomString(int iLength);
```

File

IBUtils

Description

This is function IBUtils.RandomString.

33.1.10 IBUtils.StripString Function

Pascal

```
function StripString(st: String; CharsToStrip: String): String;
```

C++

```
UnicodeString StripString(UnicodeString st, UnicodeString CharsToStrip);
```

File

IBUtils

Description

This is function IBUtils.StripString.

33.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CR (see page 4186)	This is constant IBUtils.CR.
CRLF (see page 4186)	This is constant IBUtils.CRLF.
LF (see page 4187)	This is constant IBUtils.LF.
NULL_TERMINATOR (see page 4187)	This is constant IBUtils.NULL_TERMINATOR.
TAB (see page 4187)	This is constant IBUtils.TAB.

33.2.1 IBUtils.CR Constant

Pascal

```
CR = #13;
```

C++

```
CR = #13;
```

File

IBUtils

Description

This is constant IBUtils.CR.

33.2.2 IBUtils.CRLF Constant

Pascal

```
CRLF = #13 + #10;
```

C++

```
CRLF = #13 + #10;
```

File

IBUtils

Description

This is constant IBUtils.CRLF.

33.2.3 IBUtils.LF Constant

Pascal

```
LF = #10;
```

C++

```
LF = #10;
```

File

IBUtils

Description

This is constant IBUtils.LF.

33.2.4 IBUtils.NULL_TERMINATOR Constant

Pascal

```
NULL_TERMINATOR = #0;
```

C++

```
NULL_TERMINATOR = #0;
```

File

IBUtils

Description

This is constant IBUtils.NULL_TERMINATOR.

33.2.5 IBUtils.TAB Constant

Pascal

```
TAB = #9;
```

C++

```
TAB = #9;
```

File

IBUtils

Description

This is constant IBUtils.TAB.

34 ImgList Namespace

This is namespace ImgList.

Topics

Name	Description
Classes (see page 4190)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4227)	The following table lists structs, records, enums in this documentation.
Types (see page 4231)	The following table lists types in this documentation.

Classes

	Name	Description
	TChangeLink (see page 4190)	The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.
	TCustomImageList (see page 4193)	TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

Structs, Records, Enums

	Name	Description
	TColorDepth (see page 4227)	Color depth of list images.
	TDrawingStyle (see page 4228)	TDrawingStyle indicates the style to use when rendering an image from an image list.
	TImageType (see page 4229)	TImageType indicates whether to use a mask when rendering an image from an image list.
	TLoadResource (see page 4229)	TLoadResource and TLoadResources (see page 4232) determine how a resource is loaded into an image list.
	TResType (see page 4230)	TResType identifies a type of graphical resource to an image list.

Types

Name	Description
TImageIndex (see page 4231)	This is type ImgList.TImageIndex.
TImageListHandle (see page 4231)	This is type ImgList.TImageListHandle.
TLoadResources (see page 4232)	This is type ImgList.TLoadResources.
TOverlay (see page 4232)	This is type ImgList.TOverlay.

34.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TChangeLink (see page 4190)	The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.
	TCustomImageList (see page 4193)	TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

34.1.1 TChangeLink Class

The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.

Class Hierarchy

Pascal

```
TChangeLink = class(TObject);
```

C++

```
class TChangeLink : public TObject;
```

File

ImgList

Description

A TImageList keeps a list of its TChangeLink objects. Each is associated with a particular component, such as a tree view or list view control. A component can register its change link with an image list by calling the image list's RegisterChanges method. Before destruction, the component should then call the UnRegisterChanges method.

Component writers can use the change link object to receive notification when changes are made to an image list that a control uses. Use the OnChange (see page 4192) event of the change link to execute some code when the image list changes. For example, the tree view control uses a change link object to know when its image list has changed.

See Also

TImageList

TTreeView

TListView

TCustomImageList (see page 4193)

34.1.1.1 TChangeLink Members

The following tables list the members exposed by TChangeLink.

Public Events

	Name	Description
	OnChange (see page 4192)	Occurs when there's a change in any of the images stored in the associated image list.

Public Methods

	Name	Description
	Change (see page 4191)	Generates an OnChange (see page 4192) event.
	Destroy (see page 4191)	Destroys the change link object and releases the memory allocated to it.

Public Properties

	Name	Description
	Sender (see page 4192)	Specifies which image list component notifies the change link object of changes.

34.1.1.2 TChangeLink.Destroy Destructor

Destroys the change link object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TChangeLink();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the change link is not nil before calling Destroy.

Before freeing the memory for the change link, Destroy first calls the image list's UnRegisterChanges method to unregister itself from receiving notification when the image list changes.

See Also

System::TObject::Free

UnRegisterChanges (see page 4216)

34.1.1.3 TChangeLink Methods

The methods of the TChangeLink class are listed here.

Public Methods

	Name	Description
	Change (see page 4191)	Generates an OnChange (see page 4192) event.

34.1.1.3.1 TChangeLink.Change Method

Generates an OnChange (see page 4192) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Use the Change method to trigger the OnChange (see page 4192) event handler of the change link object.

See Also

OnChange (see page 4192)

34.1.1.4 TChangeLink Properties

The properties of the TChangeLink class are listed here.

Public Properties

	Name	Description
	Sender (see page 4192)	Specifies which image list component notifies the change link object of changes.

34.1.1.4.1 TChangeLink.Sender Property

Specifies which image list component notifies the change link object of changes.

Pascal

```
property Sender: TCustomImageList;
```

C++

```
__property TCustomImageList Sender;
```

Description

Use Sender to access the image list that has changed. The TImageList automatically sets this property when a call to RegisterChanges is made.

See Also

RegisterChanges (see page 4214)

34.1.1.5 TChangeLink Events

The events of the TChangeLink class are listed here.

Public Events

	Name	Description
	OnChange (see page 4192)	Occurs when there's a change in any of the images stored in the associated image list.

34.1.1.5.1 TChangeLink.OnChange Event

Occurs when there's a change in any of the images stored in the associated image list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use the OnChange event to notify a control when the image list that it's using changes. Specify any special processing to occur at that time in the OnChange event handler.

See Also

Change (see page 4191)

34.1.2 TCustomImageList Class

TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TCustomImageList = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TCustomImageList : public TComponent;
```

File

ImgList

Description

Image lists manage large sets of icons or bitmaps. All images in an image list are contained in a single, wide bitmap in screen device format. An image list may also include a monochrome bitmap that contains masks that are combined with the images to produce a transparent effect such as is common in icons.

See Also

TPicture (see page 2812)

TImage (see page 1048)

34.1.2.1 TCustomImageList Members

The following tables list the members exposed by TCustomImageList.

Protected Methods

	Name	Description
	AssignTo (see page 4200)	Copies the image list's properties to another image list.
	Change (see page 4200)	Generates an OnChange (see page 4226) event.
	DefineProperties (see page 4201)	Allows bitmap data to be saved and loaded.
	DoDraw (see page 4201)	Draws the specified image onto the provided Canvas in the indicated style.
	GetImages (see page 4208)	Returns a specified image and its mask as bitmaps.
	HandleNeeded (see page 4211)	This is HandleNeeded, a member of class TCustomImageList.
	Initialize (see page 4211)	Initializes a newly allocated image list.
	ReadData (see page 4213)	This is ReadData, a member of class TCustomImageList.
	WriteData (see page 4217)	This is WriteData, a member of class TCustomImageList.

Public Events

	Name	Description
	OnChange (see page 4226)	Occurs when the image list changes.

Public Methods

	Name	Description
	Add (see page 4197)	Adds an Image and its Mask (see page 4591) to the image list object.
	AddIcon (see page 4197)	Adds an icon to the image list object.
	AddImage (see page 4198)	Adds single image from another image list.
	AddImages (see page 4198)	Adds all images from another image list.
	AddMasked (see page 4199)	Adds an image to the image list along with a color which is used to generate the transparent mask.
V	Assign (see page 4199)	Copies the information from one image list to another.
	Clear (see page 4200)	Clears all images from the image list.
V	Create (see page 4195)	Creates an empty image list.
	CreateSize (see page 4195)	Creates an empty image list of a specified size.
	Delete (see page 4201)	Deletes a particular image from the image list.
V	Destroy (see page 4196)	Destroys the image list and releases the memory allocated to it.
	Draw (see page 4202)	This is the overview for the Draw method overload.
	DrawOverlay (see page 4205)	This is the overview for the DrawOverlay method overload.
	FileLoad (see page 4206)	Loads a resourced image into the image list.
	GetBitmap (see page 4206)	Retrieves a specified image as a bitmap.
V	GetHotSpot (see page 4207)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 4207)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 4208)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 4209)	This is the overview for the GetInstRes method overload.
	GetMaskBitmap (see page 4210)	Returns a handle to a bitmap containing all masks for images in the image list.
	GetResource (see page 4210)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 4211)	Indicates whether a window handle exists for the image list.
	Insert (see page 4212)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertIcon (see page 4212)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 4212)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	Move (see page 4213)	Moves a specified image to another position.
	Overlay (see page 4213)	Identifies an image in the list as an image that can be used as an overlay mask.
	RegisterChanges (see page 4214)	Registers an object to be notified when changes occur in the image list.
	Replace (see page 4214)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 4214)	Replaces an image in the image list with an icon.
	ReplaceMasked (see page 4215)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 4215)	Loads a resource into the image list.
	ResourceLoad (see page 4216)	Loads a resource into the image list.
	SetSize (see page 4216)	This is SetSize, a member of class TCustomImageList.
	UnRegisterChanges (see page 4216)	Removes an object from the list of objects to be notified when the image list changes.

Public Properties

	Name	Description
	AllocBy (see page 4217)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 4218)	Determines which background color to use when drawing an image.
	BlendColor (see page 4218)	Determines which foreground color to use when drawing an image.
R	Count (see page 4219)	Indicates the number of images in the image list.
	DrawingStyle (see page 4223)	Specifies the style to be used when the image list is drawing an image.
	Handle (see page 4224)	Designates the handle for the image list.

	Height (see page 4224)	Specifies the height of images within the image list.
	ImageType (see page 4224)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 4225)	Specifies whether the image list includes masks that can be combined with the images.
	ShareImages (see page 4225)	Specifies whether the image list destroys its handle when the image list is destroyed.
	Width (see page 4225)	Specifies the width of all images within the image list.

Published Properties

	Name	Description
	ColorDepth (see page 4218)	Color depth of images in list.

34.1.2.2 TCustomImageList.Create Constructor

Creates an empty image list.

Pascal

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

C++

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

Description

Call Create to instantiate an image list at runtime. ImageLists placed on forms or data modules at design time are created automatically. Specify the owner of the new imagelist using the AOwner parameter.

The new image list contains no images and has a default size of 16 by 16 pixels.

See Also

CreateSize (see page 4195)

34.1.2.3 TCustomImageList.CreateSize Constructor

Creates an empty image list of a specified size.

Pascal

```
constructor CreateSize(AWidth: Integer; AHeight: Integer);
```

C++

```
__fastcall TCustomImageList(int AWidth, int AHeight);
```

Description

Use CreateSize to create an image list object at runtime with an image size other than the default of 16 x 16 pixels. The initial size of the image list is determined by the AHeight and AWidth parameters. The height and width can be altered after the image list has been created using the Width (see page 4225) and Height (see page 4224) properties.

See Also

Create (see page 4195)

Height (see page 4224)

Width (see page 4225)

34.1.2.4 TCustomImageList.Destroy Destructor

Destroys the image list and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomImageList();
```

Description

Destroy first calls UnRegisterChanges (see page 4216) to unregister any objects registered to receive notification when the image list changed, and frees all helper objects associated with the image list.

Destroy seldom needs to be called. However, when an image list is constructed without an owner by the Create (see page 4195) or CreateSize (see page 4195) method, call Free to release memory and dispose of the object. Free checks to see if the pointer is nil before calling Destroy.

See Also

System::TObject::Free

UnRegisterChanges (see page 4216)

34.1.2.5 TCustomImageList Methods

The methods of the TCustomImageList class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 4200)	Copies the image list's properties to another image list.
	Change (see page 4200)	Generates an OnChange (see page 4226) event.
	DefineProperties (see page 4201)	Allows bitmap data to be saved and loaded.
	DoDraw (see page 4201)	Draws the specified image onto the provided Canvas in the indicated style.
	GetImages (see page 4208)	Returns a specified image and its mask as bitmaps.
	HandleNeeded (see page 4211)	This is HandleNeeded, a member of class TCustomImageList.
	Initialize (see page 4211)	Initializes a newly allocated image list.
	ReadData (see page 4213)	This is ReadData, a member of class TCustomImageList.
	WriteData (see page 4217)	This is WriteData, a member of class TCustomImageList.

Public Methods

	Name	Description
	Add (see page 4197)	Adds an Image and its Mask (see page 4591) to the image list object.
	AddIcon (see page 4197)	Adds an icon to the image list object.
	AddImage (see page 4198)	Adds single image from another image list.
	AddImages (see page 4198)	Adds all images from another image list.
	AddMasked (see page 4199)	Adds an image to the image list along with a color which is used to generate the transparent mask.
	Assign (see page 4199)	Copies the information from one image list to another.
	Clear (see page 4200)	Clears all images from the image list.
	Delete (see page 4201)	Deletes a particular image from the image list.
	Draw (see page 4202)	This is the overview for the Draw method overload.
	DrawOverlay (see page 4205)	This is the overview for the DrawOverlay method overload.
	FileLoad (see page 4206)	Loads a resourced image into the image list.

	GetBitmap (see page 4206)	Retrieves a specified image as a bitmap.
	GetHotSpot (see page 4207)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 4207)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 4208)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 4209)	This is the overview for the GetInstRes method overload.
	GetMaskBitmap (see page 4210)	Returns a handle to a bitmap containing all masks for images in the image list.
	GetResource (see page 4210)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 4211)	Indicates whether a window handle exists for the image list.
	Insert (see page 4212)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertIcon (see page 4212)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 4212)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	Move (see page 4213)	Moves a specified image to another position.
	Overlay (see page 4213)	Identifies an image in the list as an image that can be used as an overlay mask.
	RegisterChanges (see page 4214)	Registers an object to be notified when changes occur in the image list.
	Replace (see page 4214)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 4214)	Replaces an image in the image list with an icon.
	ReplaceMasked (see page 4215)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 4215)	Loads a resource into the image list.
	ResourceLoad (see page 4216)	Loads a resource into the image list.
	SetSize (see page 4216)	This is SetSize, a member of class TCustomImageList.
	UnRegisterChanges (see page 4216)	Removes an object from the list of objects to be notified when the image list changes.

34.1.2.5.1 TCustomImageList.Add Method

Adds an Image and its Mask (see page 4591) to the image list object.

Pascal

```
function Add(Image: TBitmap; Mask: TBitmap): Integer;
```

C++

```
__fastcall int Add(TBitmap Image, TBitmap Mask);
```

Description

Use the Add method to add images with their masks to an image list. It returns the index of the added image, or -1 if it fails. When the Masked (see page 4225) property of the image list is false, the second bitmap is ignored; it can be set to nil (Delphi) or NULL (C++).

See Also

AddIcon (see page 4197)

AddImages (see page 4198)

AddMasked (see page 4199)

Masked (see page 4225)

34.1.2.5.2 TCustomImageList.AddIcon Method

Adds an icon to the image list object.

Pascal

```
function AddIcon(Image: TIcon): Integer;
```

C++

```
__fastcall int AddIcon(TIcon Image);
```

Description

Use the AddIcon method to add the icon specified by the Image parameter to an image list. AddIcon returns the index of the added image, or -1 if it fails.

See Also

Add (see page 4197)

AddImages (see page 4198)

AddMasked (see page 4199)

34.1.2.5.3 TCustomImageList.AddImage Method

Adds single image from another image list.

Pascal

```
function AddImage(Value: TCustomImageList; Index: Integer): Integer;
```

C++

```
__fastcall int AddImage(TCustomImageList Value, int Index);
```

Description

Use AddImage to add a single image from another image list. Values specifies the source image list. Index indicates the specific image, where images are numbered from 0.

AddImage returns the new size of the image list.

See Also

Add (see page 4197)

AddIcon (see page 4197)

AddMasked (see page 4199)

AddImages (see page 4198)

34.1.2.5.4 TCustomImageList.AddImages Method

Adds all images from another image list.

Pascal

```
procedure AddImages(Value: TCustomImageList);
```

C++

```
__fastcall AddImages(TCustomImageList Value);
```

Description

Use AddImages to add all the images from another image list. Value specifies the source image list.

See Also

Add (see page 4197)

AddIcon (see page 4197)

AddMasked (see page 4199)

AddImage (see page 4198)

34.1.2.5.5 TCustomImageList.AddMasked Method

Adds an image to the image list along with a color which is used to generate the transparent mask.

Pascal

```
function AddMasked(Image: TBitmap; MaskColor: TColor): Integer;
```

C++

```
__fastcall int AddMasked(TBitmap Image, TColor MaskColor);
```

Description

AddMasked is similar to Add (see page 4197) except that a mask bitmap is not specified. Instead, a color that can be used to automatically generate the masks is specified. Each pixel of the specified color in the image bitmap is changed to black, and the corresponding bit in the mask is set to one. As a result, any pixel in the image that matches the specified color is transparent when the image is drawn. AddMasked returns -1 if it fails, 0 if it's successful.

See Also

Add (see page 4197)

AddIcon (see page 4197)

AddImages (see page 4198)

Masked (see page 4225)

34.1.2.5.6 TCustomImageList.Assign Method

Copies the information from one image list to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use this method to copy another object's information to the image list.

If Source is another image list (TCustomImageList (see page 4193) descendant), Assign discards any current image information and replaces it with the information from Source.

If Source is nil (Delphi) or NULL (C++), Assign clears the image list.

If Source is any other type of object, Assign calls its inherited method, which can copy from any source object that implements a way to copy to an image list in its AssignTo (see page 4200) method.

See Also

AddImages (see page 4198)

Add (see page 4197)

AssignTo

34.1.2.5.7 TCustomImageList.AssignTo Method

Copies the image list's properties to another image list.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications do not call the protected AssignTo method. Instead, call the public Assign (see page 4199) method of the destination image list. AssignTo is called when the image list is passed as the Source parameter of another object's Assign (see page 4199) method and that Assign (see page 4199) method can't copy information to the image list. The object whose Assign (see page 4199) method was called is passed as the Dest parameter. AssignTo gives the image list an opportunity to copy properties to objects that were not created with the image list in mind.

See Also

AddImages (see page 4198)

Add (see page 4197)

Assign (see page 4199)

34.1.2.5.8 TCustomImageList.Change Method

Generates an OnChange (see page 4226) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change calls the OnChange (see page 4226) event handler if it is assigned. Override Change to block this event or provide additional responses to changes in the image list.

See Also

RegisterChanges (see page 4214)

OnChange (see page 4226)

34.1.2.5.9 TCustomImageList.Clear Method

Clears all images from the image list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the Clear method to remove all images from the image list. Use the Delete (see page 4201) method to delete one image from the image list.

See Also

Delete (see page 4201)

34.1.2.5.10 TCustomImageList.DefineProperties Method

Allows bitmap data to be saved and loaded.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties allows the image list to create pseudo properties that are loaded and saved with the image list. TCustomImageList (see page 4193) overrides the inherited method to save or load the bitmap image in addition to the other published properties.

See Also

DefineBinaryProperty

DefineProperties

34.1.2.5.11 TCustomImageList.Delete Method

Deletes a particular image from the image list.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Use the Delete method to delete an image specified by Index in the image list. Use the Clear (see page 4200) method to delete all images in the image list.

See Also

Clear (see page 4200)

34.1.2.5.12 TCustomImageList.DoDraw Method

Draws the specified image onto the provided Canvas in the indicated style.

Pascal

```
procedure DoDraw(Index: Integer; Canvas: TCanvas; X: Integer; Y: Integer; Style: Cardinal;
Enabled: Boolean = True); virtual;
```

C++

```
virtual __fastcall DoDraw(int Index, TCanvas Canvas, int X, int Y, unsigned Style, Boolean
Enabled = True);
```

Description

Applications can't call the protected DoDraw method. It provides the underlying implementation of the public Draw (see page 4202) method.

Index specifies an image to draw.

Canvas indicates the drawing surface on which the image is rendered.

X and Y specify the top-left corner of the image on Canvas.

Style encodes how the image blends color or uses a mask (the DrawingStyle (see page 4223) and ImageType (see page 4224)).

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, DoDraw renders a gray-mapped version of the image.

See Also

Draw (see page 4202)

DrawingStyle (see page 4223)

ImageType (see page 4224)

34.1.2.5.13 Draw Method

This is the overview for the Draw method overload.

Overload List

	Name	Description
	TCustomImageList.Draw (TCanvas, Integer, Integer, Integer, Boolean) (see page 4202)	Draws the image specified by the Index parameter onto the provided Canvas.
	TCustomImageList.Draw (TCanvas, Integer, Integer, Integer, TDrawingStyle, TImageType, Boolean) (see page 4202)	Draws the image specified by the Index parameter onto the provided Canvas.

34.1.2.5.13.1 TCustomImageList.Draw Method (TCanvas, Integer, Integer, Integer, Boolean)

Draws the image specified by the Index parameter onto the provided Canvas.

Pascal

```
procedure Draw(Canvas: TCanvas; X: Integer; Y: Integer; Index: Integer; Enabled: Boolean =
True); overload;
procedure Draw(Canvas: TCanvas; X: Integer; Y: Integer; Index: Integer; ADrawingStyle:
TDrawingStyle; AImageType: TImageType; Enabled: Boolean = True); overload;
```

C++

```
__fastcall Draw(TCanvas Canvas, int X, int Y, int Index, Boolean Enabled = True);
__fastcall Draw(TCanvas Canvas, int X, int Y, int Index, TDrawingStyle ADrawingStyle,
TImageType AImageType, Boolean Enabled = True);
```

Description

Use the Draw method to draw one of the images in the image list onto a specified canvas.

Canvas is the drawing surface on which to render the image.

X and Y specify the location where the top-left corner should appear on Canvas.

Index indicates which image to draw, where 0 specifies the first image, 1 specifies the second image, and so on.

ADrawingStyle indicates how the color of the image may be altered. If this parameter is not specified, Draw uses the value of the DrawingStyle (see page 4223) property.

AImageType indicates whether to draw the image or its associated mask. If this parameter is not specified, Draw uses the value of the ImageType (see page 4224) property.

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, Draw renders a gray-mapped version of the image. Because Enabled has a default value of true, this parameter can be omitted if the image should not be grayed.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
```

See Also

DrawingStyle (see page 4223)

DrawOverlay (see page 4205)

GetImages (see page 4208)

ImageType (see page 4224)

34.1.2.5.14 DrawOverlay Method

This is the overview for the DrawOverlay method overload.

Overload List

	Name	Description
	TCustomImageList.DrawOverlay (TCanvas, Integer, Integer, Integer, TOverlay, Boolean) (see page 4205)	Draws an image and an overlay onto the provided Canvas.
	TCustomImageList.DrawOverlay (TCanvas, Integer, Integer, Integer, TOverlay, TDrawingStyle, TImageType, Boolean) (see page 4205)	Draws an image and an overlay onto the provided Canvas.

34.1.2.5.14.1 TCustomImageList.DrawOverlay Method (TCanvas, Integer, Integer, Integer, TOverlay, Boolean)

Draws an image and an overlay onto the provided Canvas.

Pascal

```
procedure DrawOverlay(Canvas: TCanvas; X: Integer; Y: Integer; ImageIndex: Integer; Overlay:
TOverlay; Enabled: Boolean = True); overload;
procedure DrawOverlay(Canvas: TCanvas; X: Integer; Y: Integer; ImageIndex: Integer; Overlay:
TOverlay; ADrawingStyle: TDrawingStyle; AImageType: TImageType; Enabled: Boolean = True);
overload;
```

C++

```
__fastcall DrawOverlay(TCanvas Canvas, int X, int Y, int ImageIndex, TOverlay Overlay, Boolean
Enabled = True);
__fastcall DrawOverlay(TCanvas Canvas, int X, int Y, int ImageIndex, TOverlay Overlay,
TDrawingStyle ADrawingStyle, TImageType AImageType, Boolean Enabled = True);
```

Description

An overlay mask is an image drawn transparently over another image. Any image can be used as an overlay mask. To draw an overlay mask over an image use the DrawOverlay method with the index of the overlay mask.

Canvas is the drawing surface on which to draw the overlay.

X and Y specify the location where the top-left corner should appear on Canvas.

ImageIndex indicates which image to draw, where 0 specifies the first image, 1 specifies the second image, and so on.

Overlay (see page 4213) specifies the index of the overlay mask to use. This index is assigned to the image that is used as an overlay mask by calling the Overlay (see page 4213) method.

ADrawingStyle indicates how the color of the image may be altered. If this parameter is not specified, Draw (see page 4202) uses the value of the DrawingStyle (see page 4223) property.

AImageType indicates whether to draw the image or its associated mask. If this parameter is not specified, Draw (see page 4202) uses the value of the ImageType (see page 4224) property.

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, DoDraw (see page 4201) renders a gray-mapped version of the image. Because Enabled has a default value of true, this parameter can be omitted if the image should not be grayed.

See Also

Overlay (see page 4213)

Draw (see page 4202)

SetDragImage

34.1.2.5.15 TCustomImageList.FileLoad Method

Loads a resourced image into the image list.

Pascal

```
function FileLoad(ResType: TResType; const Name: string; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean FileLoad(TResType ResType, const UnicodeString Name, TColor MaskColor);
```

Description

Use the FileLoad method to load a bitmap, cursor, or icon from a resource file. Use ResType to specify the type of resource. Use Name to identify the resource by name. Use MaskColor to set the transparent color for the images. FileLoad returns true if successful.

Note: Within a package, FileLoad returns an error. GetInstRes (see page 4209) must be used to load a resource from within a package.

See Also

GetResource (see page 4210)

ResourceLoad (see page 4216)

GetInstRes (see page 4209)

34.1.2.5.16 TCustomImageList.GetBitmap Method

Retrieves a specified image as a bitmap.

Pascal

```
function GetBitmap(Index: Integer; Image: TBitmap): Boolean;
```

C++

```
__fastcall Boolean GetBitmap(int Index, TBitmap Image);
```

Description

Use the GetBitmap method to obtain a particular image in the image list as a bitmap object. Index denotes the index of the image. GetBitmap assigns the image to the TBitmap (see page 2628) object passed in as the Image parameter.

C++ Examples:

```
/*
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ImageList1->GetBitmap(0, Image1->Picture->Bitmap);
  Image1->Stretch = True;
```

```
}
```

Delphi Examples:

```
{
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   ImageList1.GetBitmap(2,Image1.Picture.Bitmap);
   Image1.Stretch := True;
end;
```

See Also

GetMaskBitmap (see page 4210)

34.1.2.5.17 TCustomImageList.GetHotSpot Method

Returns the hot spot location associated with the current drag image.

Pascal

```
function GetHotSpot: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetHotSpot();
```

Description

Use the GetHotSpot method to obtain the position of the hot spot of the current drag image.

See Also

DragMove

SetDragImage

34.1.2.5.18 GetIcon Method

This is the overview for the GetIcon method overload.

Overload List

	Name	Description
	TCustomImageList.GetIcon (Integer, TIcon) (see page 4207)	Returns the image specified by Index as an icon into the Image parameter.
	TCustomImageList.GetIcon (Integer, TIcon, TDrawingStyle, TImageType) (see page 4207)	Returns the image specified by Index as an icon into the Image parameter.

34.1.2.5.18.1 TCustomImageList.GetIcon Method (Integer, TIcon)

Returns the image specified by Index as an icon into the Image parameter.

Pascal

```
procedure GetIcon(Index: Integer; Image: TIcon); overload;
procedure GetIcon(Index: Integer; Image: TIcon; ADrawingStyle: TDrawingStyle; AImageType:
TImageType); overload;
```

C++

```
__fastcall GetIcon(int Index, TIcon Image);
__fastcall GetIcon(int Index, TIcon Image, TDrawingStyle ADrawingStyle, TImageType AImageType);
```

Description

Call GetIcon to obtain a particular image in the image list as an icon.

Index identifies the desired image, where 0 specifies the first image, 1 specifies the second image, and so on.

Image is a TIcon (see page 2768) object that receives the new image.

ADrawingStyle specifies the amount of color that is blended with the image to achieve various affects. When GetIcon is called using the version that does not take an ADrawingStyle parameter, it uses the value of the DrawingStyle (see page 4223) property.

AImageType specifies whether the image is rendered using a mask. When GetIcon is called using the version that does not take an AImageType parameter, it uses the value of the ImageType (see page 4224) property.

See Also

GetImageBitmap (see page 4208)

DrawingStyle (see page 4223)

ImageType (see page 4224)

34.1.2.5.19 TCustomImageList.GetImageBitmap Method

Returns a handle to a bitmap containing all of the images in the image list.

Pascal

```
function GetImageBitmap: HBITMAP;
```

C++

```
__fastcall HBITMAP GetImageBitmap();
```

Description

Use the GetImageBitmap method to obtain a Windows handle for the bitmap that contains all images in the image list. GetImageBitmap returns 0 if unsuccessful.

See Also

GetImages (see page 4208)

GetBitmap (see page 4206)

34.1.2.5.20 TCustomImageList.GetImages Method

Returns a specified image and its mask as bitmaps.

Pascal

```
procedure GetImages(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall GetImages(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use GetImages to obtain the image (and its mask) identified by the Index parameter.

Index specifies the image (and mask) to return, where 0 identifies the first image, 1 the second image, and so on.

Image is a TBitmap (see page 2628) object that receives the image of the bitmap.

Mask (see page 4591) is a TBitmap (see page 2628) object that receives the image of the mask.

See Also

GetImageBitmap (see page 4208)

GetMaskBitmap (see page 4210)

GetBitmap (see page 4206)

34.1.2.5.21 GetInstRes Method

This is the overview for the GetInstRes method overload.

Overload List

	Name	Description
	TCustomImageList.GetInstRes (THandle, TResType, DWORD, Integer, TLoadResources, TColor) (see page 4209)	Loads a specified bitmap, cursor, or icon resource into the image list.
	TCustomImageList.GetInstRes (THandle, TResType, string, Integer, TLoadResources, TColor) (see page 4209)	Loads a specified bitmap, cursor, or icon resource into the image list.

34.1.2.5.21.1 TCustomImageList.GetInstRes Method (THandle, TResType, DWORD, Integer, TLoadResources, TColor)

Loads a specified bitmap, cursor, or icon resource into the image list.

Pascal

```
function GetInstRes(Instance: THandle; ResType: TResType; const Name: string; Width: Integer;
LoadFlags: TLoadResources; MaskColor: TColor): Boolean; overload;
function GetInstRes(Instance: THandle; ResType: TResType; ResID: DWORD; Width: Integer;
LoadFlags: TLoadResources; MaskColor: TColor): Boolean; overload;
```

C++

```
__fastcall Boolean GetInstRes(THandle Instance, TResType ResType, const UnicodeString Name,
int Width, TLoadResources LoadFlags, TColor MaskColor);
__fastcall Boolean GetInstRes(THandle Instance, TResType ResType, DWORD ResID, int Width,
TLoadResources LoadFlags, TColor MaskColor);
```

Description

Use the GetInstRes method to load bitmap, cursor, or icon resource into an image list.

Instance is the instance handle of the resource module.

ResType indicates the type of resource (rtBitmap, rtCursor or rtIcon).

Name or ResID identifies the resource by name or ID, respectively.

Width (see page 4225) specifies the width, in pixels, of the image.

LoadFlags specifies how to load the image.

MaskColor sets the transparent color for the image.

Note: GetInstRes must be used to load a resource from within a package. Within a package, GetResource (see page 4210) or FileLoad (see page 4206) returns an error.

See Also

ResInstLoad (see page 4215)

ResourceLoad (see page 4216)

FileLoad (see page 4206)

GetResource (see page 4210)

34.1.2.5.22 TCustomImageList.GetMaskBitmap Method

Returns a handle to a bitmap containing all masks for images in the image list.

Pascal

```
function GetMaskBitmap: HBITMAP;
```

C++

```
__fastcall HBITMAP GetMaskBitmap();
```

Description

Use the GetMaskBitmap to obtain a bitmap that contains all masks in the image list. GetMaskBitmap returns 0 if unsuccessful.

See Also

GetImageBitmap (see page 4208)

GetImages (see page 4208)

34.1.2.5.23 TCustomImageList.GetResource Method

Loads the specified bitmap, cursor, or icon resource into the image list.

Pascal

```
function GetResource(ResType: TResType; const Name: string; Width: Integer; LoadFlags:
TLoadResources; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean GetResource(TResType ResType, const UnicodeString Name, int Width,
TLoadResources LoadFlags, TColor MaskColor);
```

Description

Use the GetResource method to load bitmap, cursor, or icon resource into an image list.

Note: Within a package, GetResource returns an error. GetInstRes (see page 4209) must be used to load a resource from within a package.

ResType specifies the type of graphical resource that is to be loaded into the image list and can be rtBitmap, rtCursor or rtIcon.

Name identifies the resource by name.

Width (see page 4225) specifies the width, in pixels, of the image.

LoadFlags specifies how to load the image.

MaskColor sets the transparent color for the images in the file.

See Also

ResourceLoad (see page 4216)

FileLoad (see page 4206)

GetInstRes (see page 4209)

34.1.2.5.24 TCustomImageList.HandleAllocated Method

Indicates whether a window handle exists for the image list.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine if a handle exists for the image list. HandleAllocated returns true if the Windows image list has been created and a handle is available, false otherwise. HandleAllocated doesn't create a handle if it doesn't already exist.

Note: Getting the Handle (see page 4224) property of creates the Windows image list automatically if it didn't previously exist, and returns the resulting handle.

See Also

Handle (see page 4224)

34.1.2.5.25 TCustomImageList.HandleNeeded Method

Pascal

```
procedure HandleNeeded;
```

C++

```
__fastcall HandleNeeded();
```

Description

This is HandleNeeded, a member of class TCustomImageList.

34.1.2.5.26 TCustomImageList.Initialize Method

Initializes a newly allocated image list.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize is called from the constructor of the image list. It sets numerous image list properties to their default values and creates instances of helper objects used internally.

See Also

Create (see page 4195)

CreateSize (see page 4195)

34.1.2.5.27 TCustomImageList.Insert Method

Inserts a bitmap and its mask into the image list following a specified image.

Pascal

```
procedure Insert(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall Insert(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use the Insert method to insert a bitmap and its mask into the image list. The image is inserted following the image specified by Index.

See Also

Add (see page 4197)

AddMasked (see page 4199)

34.1.2.5.28 TCustomImageList.InsertIcon Method

Inserts an icon at a specified position in the image list.

Pascal

```
procedure InsertIcon(Index: Integer; Image: TIcon);
```

C++

```
__fastcall InsertIcon(int Index, TIcon Image);
```

Description

Use the InsertIcon method to insert an icon into the image list. The icon is inserted following the image specified by Index.

See Also

AddIcon (see page 4197)

34.1.2.5.29 TCustomImageList.InsertMasked Method

Inserts a bitmap at a specified position, creating the mask from a specified transparent color.

Pascal

```
procedure InsertMasked(Index: Integer; Image: TBitmap; MaskColor: TColor);
```

C++

```
__fastcall InsertMasked(int Index, TBitmap Image, TColor MaskColor);
```

Description

Use the InsertMasked method to insert a masked image into the image list. The new image is inserted following the image specified by Index. Use MaskColor to identify the color in the image used to generate the Mask (see page 4591).

See Also

AddMasked (see page 4199)

Insert (see page 4212)

34.1.2.5.30 TCustomImageList.Move Method

Moves a specified image to another position.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer);
```

C++

```
__fastcall Move(int CurIndex, int NewIndex);
```

Description

Use the Move method to move an image to another position in the image list. Specify the current position using CurIndex, and the new position using NewIndex.

See Also

Insert (see page 4212)

34.1.2.5.31 TCustomImageList.Overlay Method

Identifies an image in the list as an image that can be used as an overlay mask.

Pascal

```
function Overlay(ImageIndex: Integer; Overlay: TOverlay): Boolean;
```

C++

```
__fastcall Boolean Overlay(int ImageIndex, TOverlay Overlay);
```

Description

Use Overlay to assign an overlay index to one of the images in the list. Once an image has an overlay index, it can be used as an overlay mask. An overlay mask is an image drawn transparently over another image.

ImageIndex identifies the image that can act as an overlay mask.

Overlay is the overlay index to assign to that image. It must be a value between 0 and 3 (inclusive). Note that this means at most four images can be used as overlay masks.

If successful, Overlay returns true.

To draw an overlay mask over an image use the DrawOverlaymethodwith the index of the overlay mask.

See Also

DrawOverlay (see page 4205)

34.1.2.5.32 TCustomImageList.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TCustomImageList.

34.1.2.5.33 TCustomImageList.RegisterChanges Method

Registers an object to be notified when changes occur in the image list.

Pascal

```
procedure RegisterChanges(Value: TChangeLink);
```

C++

```
__fastcall RegisterChanges(TChangeLink Value);
```

Description

Use the RegisterChanges method to have an object notified whenever changes occur in the image list. The TChangeLink (see page 4190)'s OnChange (see page 4226) event is called whenever a change in the image list occurs.

Create (see page 4195) a change link object for each object that should be notified of changes. Register each object by calling RegisterChanges and passing its change link object as the Value parameter. The OnChange (see page 4226) event of all registered change link objects is then triggered when a change occurs in the image list. Any code assigned to the OnChange (see page 4226) event handlers will execute.

See Also

UnRegisterChanges (see page 4216)

OnChange (see page 4192)

34.1.2.5.34 TCustomImageList.Replace Method

Replaces an image in the image list with a new image and mask.

Pascal

```
procedure Replace(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall Replace(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use the Replace method to replace the image at the Index position in the image list with a new Image and overlay Mask (see page 4591).

See Also

Insert (see page 4212)

ReplaceIcon (see page 4214)

ReplaceMasked (see page 4215)

34.1.2.5.35 TCustomImageList.ReplaceIcon Method

Replaces an image in the image list with an icon.

Pascal

```
procedure ReplaceIcon(Index: Integer; Image: TIcon);
```

C++

```
__fastcall ReplaceIcon(int Index, TIcon Image);
```

Description

Use the ReplaceIcon method to replace the Image at the Index position in the image list with a new icon.

See Also

Replace (see page 4214)

InsertIcon (see page 4212)

34.1.2.5.36 TCustomImageList.ReplaceMasked Method

Replaces an image in the image list with a new image and automatically-generated mask.

Pascal

```
procedure ReplaceMasked(Index: Integer; NewImage: TBitmap; MaskColor: TColor);
```

C++

```
__fastcall ReplaceMasked(int Index, TBitmap NewImage, TColor MaskColor);
```

Description

Use the ReplaceMasked method to replace the image at Index position within the image list specified by NewImage and the mask generated from NewImage by masking out all pixels of the color specified by MaskColor.

See Also

Replace (see page 4214)

InsertMasked (see page 4212)

34.1.2.5.37 TCustomImageList.ResInstLoad Method

Loads a resource into the image list.

Pascal

```
function ResInstLoad(Instance: THandle; ResType: TResType; const Name: string; MaskColor:
TColor): Boolean;
```

C++

```
__fastcall Boolean ResInstLoad(THandle Instance, TResType ResType, const UnicodeString Name,
TColor MaskColor);
```

Description

Use the ResoInstLoad method to load a resource of type TResType (see page 4230) into the image list. Use Instance to specify the instance handle of the resource module, and Name to identify the resource within that module. MaskColor is used to create the mask for the resulting image. ResInstLoad should be used rather than ResourceLoad (see page 4216) to load a resource from within a package.

ResInstLoad returns true if it is successful, false otherwise.

See Also

GetInstRes (see page 4209)

GetResource (see page 4210)

34.1.2.5.38 TCustomImageList.ResourceLoad Method

Loads a resource into the image list.

Pascal

```
function ResourceLoad(ResType: TResType; const Name: string; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean ResourceLoad(TResType ResType, const UnicodeString Name, TColor MaskColor);
```

Description

Use the ResourceLoad method to load a resource of type TResType (see page 4230) into the image list. Use Name to identify the resource. MaskColor is used to create the mask for the resulting image. ResourceLoad returns true if successful, false otherwise.

Note: Within a package, use ResInstLoad (see page 4215) instead of ResourceLoad. ResourceLoad returns an error when called from within a package.

See Also

GetResource (see page 4210)

ResInstLoad (see page 4215)

34.1.2.5.39 TCustomImageList.SetSize Method

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer);
```

C++

```
__fastcall SetSize(int AWidth, int AHeight);
```

Description

This is SetSize, a member of class TCustomImageList.

34.1.2.5.40 TCustomImageList.UnRegisterChanges Method

Removes an object from the list of objects to be notified when the image list changes.

Pascal

```
procedure UnRegisterChanges(Value: TChangeLink);
```

C++

```
__fastcall UnRegisterChanges(TChangeLink Value);
```

Description

Use the UnRegisterChanges method to unregister a TChangeLink (see page 4190) object. The object will no longer receive notification when the image list changes.

See Also

RegisterChanges (see page 4214)

34.1.2.5.41 TCustomImageList.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TCustomImageList.

34.1.2.6 TCustomImageList Properties

The properties of the TCustomImageList class are listed here.

Public Properties

	Name	Description
	AllocBy (see page 4217)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 4218)	Determines which background color to use when drawing an image.
	BlendColor (see page 4218)	Determines which foreground color to use when drawing an image.
R	Count (see page 4219)	Indicates the number of images in the image list.
	DrawingStyle (see page 4223)	Specifies the style to be used when the image list is drawing an image.
	Handle (see page 4224)	Designates the handle for the image list.
	Height (see page 4224)	Specifies the height of images within the image list.
	ImageType (see page 4224)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 4225)	Specifies whether the image list includes masks that can be combined with the images.
	ShareImages (see page 4225)	Specifies whether the image list destroys its handle when the image list is destroyed.
	Width (see page 4225)	Specifies the width of all images within the image list.

Published Properties

	Name	Description
	ColorDepth (see page 4218)	Color depth of images in list.

34.1.2.6.1 TCustomImageList.AllocBy Property

Sets the number of images by which the image list grows when it needs to make room for new images.

Pascal

```
property AllocBy: Integer;
```

C++

```
__property int AllocBy;
```

Description

Use the AllocBy method to specify the number of new images that can be added to the resized image list.

See Also

Count (see page 4219)

Add (see page 4197)

AddIcon (see page 4197)

AddImages (see page 4198)

AddMasked (see page 4199)

34.1.2.6.2 TCustomImageList.BkColor Property

Determines which background color to use when drawing an image.

Pascal

```
property BkColor: TColor;
```

C++

```
__property TColor BkColor;
```

Description

Use BkColor to specify the color used for the masked regions of the images in the list. When BkColor is clNone (see page 2914), the image is drawn transparently using the mask to define the transparent areas. Otherwise, the value of BkColor appears in the masked out portions.

Note: BkColor has no effect when the Masked (see page 4225) property is false.

See Also

BlendColor (see page 4218)

Masked (see page 4225)

ImageType (see page 4224)

34.1.2.6.3 TCustomImageList.BlendColor Property

Determines which foreground color to use when drawing an image.

Pascal

```
property BlendColor: TColor;
```

C++

```
__property TColor BlendColor;
```

Description

Set BlendColor to cause the images in the list to be combined with the specified color when the DrawingStyle (see page 4223) is dsFocus or dsSelected. More of BlendColor is combined with the images when DrawingStyle (see page 4223) is dsSelected. clNone (see page 2914) specifies no blend color while clDefault (see page 2903) specifies the system highlight .

See Also

BkColor (see page 4218)

DrawingStyle (see page 4223)

34.1.2.6.4 TCustomImageList.ColorDepth Property

Color depth of images in list.

Pascal

```
property ColorDepth: TColorDepth;
```

C++

```
__property TColorDepth ColorDepth;
```

Description

ColorDepth is the color depth type TColorDepth (see page 4227) of images in the list.

See Also

BkColor (see page 4218)

BlendColor (see page 4218)

Height (see page 4224)

Width (see page 4225)

TColorDepth (see page 4227)

34.1.2.6.5 TCustomImageList.Count Property

Indicates the number of images in the image list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count when iterating through the images in the image list. Retrieve the individual images using the GetBitmap (see page 4206) method. Draw (see page 4202) individual images using the Draw (see page 4202) method.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
```

```
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", reinterpret_cast<TObject *>(vsIcon));
  ComboBox1->Items->AddObject("vsList", reinterpret_cast<TObject *>(vsList));
  ComboBox1->Items->AddObject("vsReport", reinterpret_cast<TObject *>(vsReport));
  ComboBox1->Items->AddObject("vsSmallIcon", reinterpret_cast<TObject *>(vsSmallIcon));
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
```

```
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  dynamic_cast<TCustomListView *>(Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
```

```
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
{
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
}
var ColumnToSort: Integer;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
const
  imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
begin
  Col2Array[0]:= 'Documentation for the C++ icon.';
  Col2Array[1]:= 'Borland icon.';
  Col2Array[2]:= 'Delphi icon.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := imagenames[I];
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
```

```
      NewColumn.Width := 200;
      ListView1.ViewStyle := vsReport;
    end;
  end;

  procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
  begin
    ColumnToSort := Column.Index;
    (Sender as TCustomListView).AlphaSort;
  end;

  {
  The OnCompare event handler causes the list view to sort on
  the selected column:
  }
  procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

AllocBy (see page 4217)

GetBitmap (see page 4206)

Draw (see page 4202)

34.1.2.6.6 TCustomImageList.DrawingStyle Property

Specifies the style to be used when the image list is drawing an image.

Pascal

```
property DrawingStyle: TDrawingStyle;
```

C++

```
__property TDrawingStyle DrawingStyle;
```

Description

Use the DrawingStyle property to control how the image is to appear. DrawingStyle controls how much of the blend color specified by BlendColor (see page 4218) is combined with the foreground, or what background color to use.

See Also

Draw (see page 4202)

DrawOverlay (see page 4205)

ImageType (see page 4224)

BlendColor (see page 4218)

BkColor (see page 4218)

34.1.2.6.7 TCustomImageList.Handle Property

Designates the handle for the image list.

Pascal

```
property Handle: HImageList;
```

C++

```
__property HImageList Handle;
```

Description

Handle is the Windows image list object handle. Use this handle when calling Windows API functions that require it.

See Also

HandleAllocated (see page 4211)

34.1.2.6.8 TCustomImageList.Height Property

Specifies the height of images within the image list.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to specify the height of images before any are added to the image list. The image list is cleared when this value changes.

See Also

Width (see page 4225)

CreateSize (see page 4195)

Create (see page 4195)

34.1.2.6.9 TCustomImageList.ImageType Property

Determines whether the image list uses the associated mask when drawing.

Pascal

```
property ImageType: TImageType;
```

C++

```
__property TImageType ImageType;
```

Description

Use the ImageType property to control whether the images are drawn using the image mask (itMask), or without the mask (itImage).

See Also

DrawingStyle (see page 4223)

Draw (see page 4202)

Masked (see page 4225)

DrawOverlay (see page 4205)

34.1.2.6.10 TCustomImageList.Masked Property

Specifies whether the image list includes masks that can be combined with the images.

Pascal

```
property Masked: Boolean;
```

C++

```
__property Boolean Masked;
```

Description

Set the Masked property to true to combine the images with masks. The masked portion of the bitmap is drawn transparently, or replaced by the color specified by the BkColor (see page 4218) property.

Masked indicates whether the image list stores masks to combine with the images. To use those masks, the ImageType (see page 4224) property indicates whether to draw the image or the mask.

See Also

AddMasked (see page 4199)

InsertMasked (see page 4212)

BkColor (see page 4218)

ImageType (see page 4224)

34.1.2.6.11 TCustomImageList.ShareImages Property

Specifies whether the image list destroys its handle when the image list is destroyed.

Pascal

```
property ShareImages: Boolean;
```

C++

```
__property Boolean ShareImages;
```

Description

Set the ShareImages property to false to ensure that the handle to the image list is freed when the image list is destroyed.

See Also

Destroy (see page 4196)

Handle (see page 4224)

34.1.2.6.12 TCustomImageList.Width Property

Specifies the width of all images within the image list.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to specify the width of images before any are added to the image list. The image list is cleared when this value changes.

See Also

Height (see page 4224)

Create (see page 4195)

CreateSize (see page 4195)

34.1.2.7 TCustomImageList Events

The events of the TCustomImageList class are listed here.

Public Events

	Name	Description
	OnChange (see page 4226)	Occurs when the image list changes.

34.1.2.7.1 TCustomImageList.OnChange Event

Occurs when the image list changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to execute code when the image list changes. Any objects registered with the RegisterChanges (see page 4214) method will also be notified.

See Also

RegisterChanges (see page 4214)

34.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TColorDepth (see page 4227)	Color depth of list images.
	TDrawingStyle (see page 4228)	TDrawingStyle indicates the style to use when rendering an image from an image list.
	TImageType (see page 4229)	TImageType indicates whether to use a mask when rendering an image from an image list.
	TLoadResource (see page 4229)	TLoadResource and TLoadResources (see page 4232) determine how a resource is loaded into an image list.
	TResType (see page 4230)	TResType identifies a type of graphical resource to an image list.

34.2.1 ImgList.TColorDepth Enumeration

Color depth of list images.

Pascal

```
TColorDepth = (
  cdDefault,
  cdDeviceDependent,
  cd4Bit,
  cd8Bit,
  cd16Bit,
  cd24Bit,
  cd32Bit
);
```

C++

```
enum TColorDepth {
  cdDefault,
  cdDeviceDependent,
  cd4Bit,
  cd8Bit,
  cd16Bit,
  cd24Bit,
  cd32Bit
};
```

File

ImgList

Description

TColorDepth is the color depth of images in the list. This is the number of bits to represent color for each pixel. The following table describes the possible values:

Value	Meaning
cdDefault	Default color depth.
cdDeviceDependent	Device dependent color depth.

cd4Bit	4 bit color depth.
cd8Bit	8 bit color depth.
cd16Bit	16 bit color depth.
cd24Bit	24 bit color depth.
cd32Bit	32 bit color depth.

See Also

ColorDepth (see page 4218)

34.2.2 ImgList.TDrawingStyle Enumeration

TDrawingStyle indicates the style to use when rendering an image from an image list.

Pascal

```
TDrawingStyle = (
  dsFocus,
  dsSelected,
  dsNormal,
  dsTransparent
);
```

C++

```
enum TDrawingStyle {
  dsFocus,
  dsSelected,
  dsNormal,
  dsTransparent
};
```

File

ImgList

Description

TDrawingStyle describes how to render an image from an image list component. The following table describes the possible values:

Value	Meaning
dsFocus	Draws the image blending 25% with the color specified by the BlendColor property. This only affects image lists that contain masks.
dsSelected	Draws the image blending 50% with the color specified by the BlendColor property. This only affects image lists that contain masks.
dsNormal	Draws the image using the color specified in the BkColor property. If the BkColor is clNone (see page 2914), the image is drawn transparently using the mask.
dsTransparent	Draws using the mask regardless of the BkColor setting.

34.2.3 ImgList.TImageType Enumeration

TImageType indicates whether to use a mask when rendering an image from an image list.

Pascal

```
TImageType = (
  itImage,
  itMask
);
```

C++

```
enum TImageType {
  itImage,
  itMask
};
```

File

ImgList

Description

TImageType indicates whether to use a mask when rendering an image in an image list. It includes the following values:

Value	Meaning
itImage	Image does not include a mask.
itMask	Image uses a mask.

34.2.4 ImgList.TLoadResource Enumeration

TLoadResource and TLoadResources (see page 4232) determine how a resource is loaded into an image list.

Pascal

```
TLoadResource = (
  lrDefaultColor,
  lrDefaultSize,
  lrFromFile,
  lrMap3DColors,
  lrTransparent,
  lrMonoChrome
);
```

C++

```
enum TLoadResource {
  lrDefaultColor,
  lrDefaultSize,
  lrFromFile,
  lrMap3DColors,
  lrTransparent,
  lrMonoChrome
};
```

File

ImgList

Description

TLoadResources (see page 4232) is a set of flags (TLoadResource values) the govern the way an image list loads image resources. The following table lists the possible TLoadResource values:

Value	Meaning
lrDefaultColor	Use the color format of the display.
lrDefaultSize	Use system metrics for Height and Width if the Width parameter to GetResource is 0. If the Width parameter is 0 and this option is not specified for cursors and icons, the resources width and height will be used.
lrFromFile	Loads the resource from a specified file.
lrMap3DColors	Replaces the images' gray shades with the 3D windows colors.
lrTransparent	Replaces the color table entry for the first pixel of the resource with the default window color (COLOR_WINDOW). This applies only to images with color tables.
lrMonoChrome	Loads the resource as black and white.

See Also

SelectDirectory (see page 1881)

34.2.5 ImgList.TResType Enumeration

TResType identifies a type of graphical resource to an image list.

Pascal

```
TResType = (
  rtBitmap,
  rtCursor,
  rtIcon
);
```

C++

```
enum TResType {
  rtBitmap,
  rtCursor,
  rtIcon
};
```

File

ImgList

Description

TResTypes is used by the image list methods ResourceLoad, GetResource and FileLoad to specify the type of graphical resource that is to be loaded or retrieved from an image list.

Value	Meaning
rtBitmap	Loads or retrieves a bitmap
rtCursor	Loads or retrieves a cursor
rtIcon	Loads or retrieves an icon

34.3 Types

The following table lists types in this documentation.

Types

Name	Description
TImageIndex (see page 4231)	This is type ImgList.TImageIndex.
TImageListHandle (see page 4231)	This is type ImgList.TImageListHandle.
TLoadResources (see page 4232)	This is type ImgList.TLoadResources.
TOverlay (see page 4232)	This is type ImgList.TOverlay.

34.3.1 ImgList.TImageIndex Type

Pascal

```
TImageIndex = type Integer;
```

C++

```
type Integer TImageIndex;
```

File

ImgList

Description

This is type ImgList.TImageIndex.

34.3.2 ImgList.TImageListHandle Type

Pascal

```
TImageListHandle = HImageList;
```

C++

```
HImageList TImageListHandle;
```

File

ImgList

Description

This is type ImgList.TImageListHandle.

34.3.3 ImgList.TLoadResources Type

Pascal

```
TLoadResources = set of TLoadResource;
```

C++

```
TLoadResource TLoadResources;
```

File

ImgList

Description

This is type ImgList.TLoadResources.

34.3.4 ImgList.TOverlay Type

Pascal

```
TOverlay = 0..3;
```

C++

```
0..3 TOverlay;
```

File

ImgList

Description

This is type ImgList.TOverlay.

35 IMouse Namespace

This is namespace IMouse.

Topics

Name	Description
Classes (see page 4235)	The following table lists classes in this documentation.
Functions (see page 4247)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4248)	The following table lists structs, records, enums in this documentation.
Types (see page 4250)	The following table lists types in this documentation.
Constants (see page 4252)	The following table lists constants in this documentation.

Classes

	Name	Description
	TPanningWindow (see page 4235)	This is a class of TCustomPanningWindow.

Constants

Name	Description
crPanAll (see page 4252)	This is constant IMouse.crPanAll.
crPanDown (see page 4252)	This is constant IMouse.crPanDown.
crPanDownLeft (see page 4253)	This is constant IMouse.crPanDownLeft.
crPanDownRight (see page 4253)	This is constant IMouse.crPanDownRight.
crPanLeft (see page 4253)	This is constant IMouse.crPanLeft.
crPanLeftRight (see page 4254)	This is constant IMouse.crPanLeftRight.
crPanRight (see page 4254)	This is constant IMouse.crPanRight.
crPanUp (see page 4254)	This is constant IMouse.crPanUp.
crPanUpDown (see page 4255)	This is constant IMouse.crPanUpDown.
crPanUpLeft (see page 4255)	This is constant IMouse.crPanUpLeft.
crPanUpRight (see page 4255)	This is constant IMouse.crPanUpRight.

Functions

	Name	Description
	StartPan (see page 4247)	This is function IMouse.StartPan.

Structs, Records, Enums

	Name	Description
	TPanDirection (see page 4248)	The type describes the panning direction.

	TPanOption (see page 4248)	This is record IMouse.TPanOption.

Types

Name	Description
TPanDirections (see page 4250)	Set of TPanDirection (see page 4248). .
TPanOptions (see page 4250)	This is type IMouse.TPanOptions.
TUpdateEvent (see page 4250)	Initiates the cursor update.

35.1 Classes

The following table lists classes in this documentation.

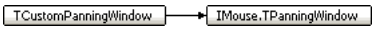
Classes

	Name	Description
	TPanningWindow (see page 4235)	This is a class of TCustomPanningWindow.

35.1.1 TPanningWindow Class

This is a class of TCustomPanningWindow.

Class Hierarchy



Pascal

```
TPanningWindow = class(TCustomPanningWindow);
```

C++

```
class TPanningWindow : public TCustomPanningWindow;
```

File

IMouse

Description

Use this type if you are creating a specialized version of TPanningWindowClass.

35.1.1.1 TPanningWindow Members

The following tables list the members exposed by TPanningWindow.

Protected Methods

	Name	Description
	CreateParams (see page 4237)	Initializes a window-creation parameter data structure.
	DoUpdate (see page 4238)	This is DoUpdate, a member of class TPanningWindow.
	GetPanOptions (see page 4238)	This is GetPanOptions, a member of class TPanningWindow.
	MouseMove (see page 4239)	Respond to mouse moving over control area..
	Paint (see page 4239)	Renders the image of a custom control.
	SetPanCursor (see page 4240)	This is SetPanCursor, a member of class TPanningWindow.
	UpdateScroll (see page 4240)	This is UpdateScroll, a member of class TPanningWindow.
	WndProc (see page 4241)	Provides specific message responses for the control.

Public Events

	Name	Description
	OnUpdate (see page 4246)	This is OnUpdate, a member of class TPanningWindow.

Public Methods

	Name	Description
	Create (see page 4236)	Creates an instance of TCustomControl.
	Destroy (see page 4236)	Destroys an instance of TCustomControl.
	GetIsPanning (see page 4238)	Returns whether or not the mouse is actively panning.
	StartPanning (see page 4240)	Tells the panning window to start the panning operation.
	StopPanning (see page 4240)	Tells the panning window to stop the panning operation.

Public Properties

	Name	Description
	IsPanning (see page 4245)	This is IsPanning, a member of class TPanningWindow.
	PanControl (see page 4245)	This is PanControl, a member of class TPanningWindow.
	PanInterval (see page 4246)	This is PanInterval, a member of class TPanningWindow.
	PanOptions (see page 4246)	This is PanOptions, a member of class TPanningWindow.

35.1.1.2 TPanningWindow.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

35.1.1.3 TPanningWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPanningWindow();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

35.1.1.4 TPanningWindow Methods

The methods of the TPanningWindow class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 4237)	Initializes a window-creation parameter data structure.
	DoUpdate (see page 4238)	This is DoUpdate, a member of class TPanningWindow.
	GetPanOptions (see page 4238)	This is GetPanOptions, a member of class TPanningWindow.
	MouseMove (see page 4239)	Respond to mouse moving over control area..
	Paint (see page 4239)	Renders the image of a custom control.
	SetPanCursor (see page 4240)	This is SetPanCursor, a member of class TPanningWindow.
	UpdateScroll (see page 4240)	This is UpdateScroll, a member of class TPanningWindow.
	WndProc (see page 4241)	Provides specific message responses for the control.

Public Methods

	Name	Description
	GetIsPanning (see page 4238)	Returns whether or not the mouse is actively panning.
	StartPanning (see page 4240)	Tells the panning window to start the panning operation.
	StopPanning (see page 4240)	Tells the panning window to stop the panning operation.

35.1.1.4.1 TPanningWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

35.1.1.4.2 TPanningWindow.DoUpdate Method

Pascal

```
procedure DoUpdate(var Delta: TPoint; var CursorDirection: TPanDirections); dynamic;
```

C++

```
__fastcall DoUpdate(TPoint Delta, TPanDirections CursorDirection);
```

Description

This is DoUpdate, a member of class TPanningWindow.

35.1.1.4.3 TPanningWindow.GetIsPanning Method

Returns whether or not the mouse is actively panning.

Pascal

```
function GetIsPanning: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsPanning();
```

Description

This method brings back a boolean value. True means that the mouse is actively panning; False indicates that no panning is occurring.

35.1.1.4.4 TPanningWindow.GetPanOptions Method

Pascal

```
function GetPanOptions: TPanOptions; virtual;
```

C++

```
virtual __fastcall TPanOptions GetPanOptions();
```

Description

This is GetPanOptions, a member of class TPanningWindow.

35.1.1.4.5 TPanningWindow.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

35.1.1.4.6 TPanningWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

35.1.1.4.7 TPanningWindow.SetPanCursor Method

Pascal

```
procedure SetPanCursor(ADirection: TPanDirections); virtual;
```

C++

```
virtual __fastcall SetPanCursor(TPanDirections ADirection);
```

Description

This is SetPanCursor, a member of class TPanningWindow.

35.1.1.4.8 TPanningWindow.StartPanning Method

Tells the panning window to start the panning operation.

Pascal

```
function StartPanning(AHandle: THandle; AControl: TControl): Boolean; override;
```

C++

```
virtual __fastcall Boolean StartPanning(THandle AHandle, TControl * AControl);
```

Description

Include this method in your code where you want the panning window to start the panning operation.

35.1.1.4.9 TPanningWindow.StopPanning Method

Tells the panning window to stop the panning operation.

Pascal

```
procedure StopPanning; override;
```

C++

```
virtual __fastcall StopPanning();
```

Description

Add this method to your code where you want the panning window to stop the panning operation.

35.1.1.4.10 TPanningWindow.UpdateScroll Method

Pascal

```
procedure UpdateScroll(Pt: TPoint); virtual;
```

C++

```
virtual __fastcall UpdateScroll(TPoint Pt);
```

Description

This is UpdateScroll, a member of class TPanningWindow.

35.1.1.4.11 TPanningWindow.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Msg: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Msg);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
```

```
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
```

```
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
```

```
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
```

```
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc

MainWndProc

WndProc

WindowProc

35.1.1.5 TPanningWindow Properties

The properties of the TPanningWindow class are listed here.

Public Properties

	Name	Description
	IsPanning (see page 4245)	This is IsPanning, a member of class TPanningWindow.
	PanControl (see page 4245)	This is PanControl, a member of class TPanningWindow.
	PanInterval (see page 4246)	This is PanInterval, a member of class TPanningWindow.
	PanOptions (see page 4246)	This is PanOptions, a member of class TPanningWindow.

35.1.1.5.1 TPanningWindow.IsPanning Property

Pascal

```
property IsPanning: Boolean;
```

C++

```
__property Boolean IsPanning;
```

Description

This is IsPanning, a member of class TPanningWindow.

35.1.1.5.2 TPanningWindow.PanControl Property

Pascal

```
property PanControl: TControl;
```

C++

```
__property TControl * PanControl;
```

Description

This is PanControl, a member of class TPanningWindow.

35.1.1.5.3 TPanningWindow.PanInterval Property

Pascal

```
property PanInterval: Integer;
```

C++

```
__property int PanInterval;
```

Description

This is PanInterval, a member of class TPanningWindow.

35.1.1.5.4 TPanningWindow.PanOptions Property

Pascal

```
property PanOptions: TPanOptions;
```

C++

```
__property TPanOptions PanOptions;
```

Description

This is PanOptions, a member of class TPanningWindow.

35.1.1.6 TPanningWindow Events

The events of the TPanningWindow class are listed here.

Public Events

	Name	Description
	OnUpdate (see page 4246)	This is OnUpdate, a member of class TPanningWindow.

35.1.1.6.1 TPanningWindow.OnUpdate Event

Pascal

```
property OnUpdate: TUpdateEvent;
```

C++

```
__property TUpdateEvent OnUpdate;
```

Description

This is OnUpdate, a member of class TPanningWindow.

35.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	StartPan (see page 4247)	This is function IMouse.StartPan.

35.2.1 IMouse.StartPan Function

Pascal

```
function StartPan(WndHandle: THandle; AControl: TControl): Boolean;
```

C++

```
Boolean StartPan(THandle WndHandle, TControl * AControl);
```

File

IMouse

Description

This is function IMouse.StartPan.

35.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPanDirection (see page 4248)	The type describes the panning direction.
	TPanOption (see page 4248)	This is record IMouse.TPanOption.

35.3.1 IMouse.TPanDirection Enumeration

The type describes the panning direction.

Pascal

```
TPanDirection = (
  pdUp,
  pdDown,
  pdLeft,
  pdRight
);
```

C++

```
enum TPanDirection {
  pdUp,
  pdDown,
  pdLeft,
  pdRight
};
```

File

IMouse

Description

The possible panning values are: pdUp, pdDown, pdLeft, pdRight.

35.3.2 IMouse.TPanOption Enumeration

Pascal

```
TPanOption = (
  poVertical,
  poHorizontal
);
```

C++

```
enum TPanOption {
  poVertical,
  poHorizontal
};
```

File

IMouse

Description

This is record IMouse.TPanOption.

35.4 Types

The following table lists types in this documentation.

Types

Name	Description
TPanDirections (see page 4250)	Set of TPanDirection (see page 4248). .
TPanOptions (see page 4250)	This is type IMouse.TPanOptions.
TUpdateEvent (see page 4250)	Initiates the cursor update.

35.4.1 IMouse.TPanDirections Type

Set of TPanDirection (see page 4248). .

Pascal

```
TPanDirections = set of TPanDirection;
```

C++

```
TPanDirection TPanDirections;
```

File

IMouse

Description

Set of TPanDirection (see page 4248).

35.4.2 IMouse.TPanOptions Type

Pascal

```
TPanOptions = set of TPanOption;
```

C++

```
TPanOption TPanOptions;
```

File

IMouse

Description

This is type IMouse.TPanOptions.

35.4.3 IMouse.TUpdateEvent Type

Initiates the cursor update.

Pascal

```
TUpdateEvent = procedure (Sender: TObject; var Delta: TPoint; var CursorDirection:
TPanDirections) of object;
```

C++

```
(Sender: TObject; var Delta: TPoint; var CursorDirection: TPanDirections) ( TUpdateEvent)();
```

File

IMouse

Description

This event is called when the cursor is to be updated.

35.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
crPanAll (see page 4252)	This is constant IMouse.crPanAll.
crPanDown (see page 4252)	This is constant IMouse.crPanDown.
crPanDownLeft (see page 4253)	This is constant IMouse.crPanDownLeft.
crPanDownRight (see page 4253)	This is constant IMouse.crPanDownRight.
crPanLeft (see page 4253)	This is constant IMouse.crPanLeft.
crPanLeftRight (see page 4254)	This is constant IMouse.crPanLeftRight.
crPanRight (see page 4254)	This is constant IMouse.crPanRight.
crPanUp (see page 4254)	This is constant IMouse.crPanUp.
crPanUpDown (see page 4255)	This is constant IMouse.crPanUpDown.
crPanUpLeft (see page 4255)	This is constant IMouse.crPanUpLeft.
crPanUpRight (see page 4255)	This is constant IMouse.crPanUpRight.

35.5.1 IMouse.crPanAll Constant

Pascal

```
crPanAll = TCursor(-50);
```

C++

```
crPanAll = TCursor(-50);
```

File

IMouse

Description

This is constant IMouse.crPanAll.

35.5.2 IMouse.crPanDown Constant

Pascal

```
crPanDown = TCursor(-51);
```

C++

```
crPanDown = TCursor(-51);
```

File

IMouse

Description

This is constant IMouse.crPanDown.

35.5.3 IMouse.crPanDownLeft Constant

Pascal

```
crPanDownLeft = TCursor(-52);
```

C++

```
crPanDownLeft = TCursor(-52);
```

File

IMouse

Description

This is constant IMouse.crPanDownLeft.

35.5.4 IMouse.crPanDownRight Constant

Pascal

```
crPanDownRight = TCursor(-53);
```

C++

```
crPanDownRight = TCursor(-53);
```

File

IMouse

Description

This is constant IMouse.crPanDownRight.

35.5.5 IMouse.crPanLeft Constant

Pascal

```
crPanLeft = TCursor(-54);
```

C++

```
crPanLeft = TCursor(-54);
```

File

IMouse

Description

This is constant IMouse.crPanLeft.

35.5.6 IMouse.crPanLeftRight Constant

Pascal

```
crPanLeftRight = TCursor(-55);
```

C++

```
crPanLeftRight = TCursor(-55);
```

File

IMouse

Description

This is constant IMouse.crPanLeftRight.

35.5.7 IMouse.crPanRight Constant

Pascal

```
crPanRight = TCursor(-56);
```

C++

```
crPanRight = TCursor(-56);
```

File

IMouse

Description

This is constant IMouse.crPanRight.

35.5.8 IMouse.crPanUp Constant

Pascal

```
crPanUp = TCursor(-57);
```

C++

```
crPanUp = TCursor(-57);
```

File

IMouse

Description

This is constant IMouse.crPanUp.

35.5.9 IMouse.crPanUpDown Constant

Pascal

```
crPanUpDown = TCursor(-58);
```

C++

```
crPanUpDown = TCursor(-58);
```

File

IMouse

Description

This is constant IMouse.crPanUpDown.

35.5.10 IMouse.crPanUpLeft Constant

Pascal

```
crPanUpLeft = TCursor(-59);
```

C++

```
crPanUpLeft = TCursor(-59);
```

File

IMouse

Description

This is constant IMouse.crPanUpLeft.

35.5.11 IMouse.crPanUpRight Constant

Pascal

```
crPanUpRight = TCursor(-60);
```

C++

```
crPanUpRight = TCursor(-60);
```

File

IMouse

Description

This is constant IMouse.crPanUpRight.

36 IniFiles Namespace

This is namespace IniFiles.

Topics

Name	Description
Classes (see page 4258)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4328)	The following table lists structs, records, enums in this documentation.
Types (see page 4329)	The following table lists types in this documentation.

Classes

	Name	Description
	EIniFileException (see page 4258)	EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 4294).
	TCustomIniFile (see page 4258)	TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.
	THashedStringList (see page 4291)	THashedStringList maintains a list of strings using an internal hash table.
	TIniFile (see page 4294)	TIniFile stores and retrieves application-specific information and settings from INI files.
	TMemIniFile (see page 4307)	TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.
	TStringHash (see page 4324)	This is class IniFiles.TStringHash.

Structs, Records, Enums

	Name	Description
	THashItem (see page 4328)	This is record IniFiles.THashItem.

Types

Name	Description
PHashItem (see page 4329)	This is type IniFiles.PHashItem.
PPHashItem (see page 4329)	This is type IniFiles.PPHashItem.

36.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EIniFileException (see page 4258)	EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 4294).
	TCustomIniFile (see page 4258)	TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.
	THashedStringList (see page 4291)	THashedStringList maintains a list of strings using an internal hash table.
	TIniFile (see page 4294)	TIniFile stores and retrieves application-specific information and settings from INI files.
	TMemIniFile (see page 4307)	TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.
	TStringHash (see page 4324)	This is class IniFiles.TStringHash.

36.1.1 EIniFileException Class

EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 4294).

Class Hierarchy

Pascal

```
EIniFileException = class(Exception);
```

C++

```
class EIniFileException : public Exception;
```

File

IniFiles

Description

EIniFileException is raised when TIniFile (see page 4294) fails in an attempt to modify its associated INI file.

See Also

TIniFile (see page 4294)

36.1.2 TCustomIniFile Class

TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.

Class Hierarchy

Pascal

```
TCustomIniFile = class(TObject);
```

C++

```
class TCustomIniFile : public TObject;
```

File

IniFiles

Description

Note: Use TCustomIniFile as a base class when defining components that access ini files. Ini files are text files that are divided into sections with variable assignments in each section. TCustomIniFile introduces properties and methods to handle storage and retrieval of application-specific information and settings in a standard ini file.

Note: On Windows, TCustomIniFile is also the base class for TRegistryIniFile, which allows applications that use both ini files and the system registry to handle both media in a unified manner. This differs from the features of TRegistry, which corresponds more directly to Windows system registry native semantics.

Do not create instances of TCustomIniFile. Instead, use or create descendants of TCustomIniFile, such as TIniFile (see page 4294), TRegistryIniFile, and TMemIniFile (see page 4307).

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
```

```
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
```

```
        false: WindowState := wsNormal;
      end;

    finally
      SettingsFile.Free;
    end;
  end;

  procedure TForm2.btStoreClick(Sender: TObject);
  var
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

TIniFile (see page 4294)

TMemIniFile (see page 4307)

TRegistry (see page 6610)

TRegistryIniFile (see page 6663)

36.1.2.1 TCustomIniFile Members

The following tables list the members exposed by TCustomIniFile.

Public Methods

	Name	Description
	Create (see page 4262)	Creates a TCustomIniFile (see page 4258) object for an application.
	DeleteKey (see page 4263)	Erases the data value associated with an ini file entry.
	EraseSection (see page 4264)	Erases an entire section of an ini file.
	ReadBinaryStream (see page 4264)	Retrieves a binary value from the ini file.
	ReadBool (see page 4265)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 4268)	Retrieves a date value from an ini file.
	ReadDateTime (see page 4268)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 4269)	Retrieves a float value from an ini file.
	ReadInteger (see page 4269)	Retrieves an integer value from an ini file.
	ReadSection (see page 4272)	Reads all the key names from a specified section of an ini file into a string list.
	ReadSections (see page 4273)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 4274)	Reads the values from all keys in a section of an ini file into a string list.
	ReadString (see page 4274)	Retrieves a string value from an ini file.
	ReadTime (see page 4277)	Retrieves a time value from an ini file.
	SectionExists (see page 4278)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 4278)	Flushes buffered ini file data to disk.
	ValueExists (see page 4278)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 4279)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 4279)	Writes a Boolean value to an ini file.
	WriteDate (see page 4282)	Writes a date value to an ini file.
	WriteDateTime (see page 4283)	Writes a date-time value to an ini file.
	WriteFloat (see page 4283)	Writes a float value to an ini file.
	WriteInteger (see page 4284)	Writes an integer value to an ini file.
	WriteString (see page 4287)	Writes a string value to an ini file.
	WriteTime (see page 4290)	Writes a time value to an ini file.

Public Properties

	Name	Description
	FileName (see page 4291)	Contains the name of the ini file from which to read and to which to write information.

36.1.2.2 TCustomIniFile.Create Constructor

Creates a TCustomIniFile (see page 4258) object for an application.

Pascal

```
constructor Create(const FileName: string);
```

C++

```
__fastcall TCustomIniFile(const UnicodeString FileName);
```

Description

Create constructs a TCustomIniFile (see page 4258) object for an application. Create assigns the FileName (see page 4291) parameter to the FileName (see page 4291) property, which is used to specify the name of the ini file to use.

See Also

FileName (see page 4291)

System::TObject::Create

36.1.2.3 TCustomIniFile Methods

The methods of the TCustomIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 4263)	Erases the data value associated with an ini file entry.
	EraseSection (see page 4264)	Erases an entire section of an ini file.
	ReadBinaryStream (see page 4264)	Retrieves a binary value from the ini file.
	ReadBool (see page 4265)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 4268)	Retrieves a date value from an ini file.
	ReadDateTime (see page 4268)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 4269)	Retrieves a float value from an ini file.
	ReadInteger (see page 4269)	Retrieves an integer value from an ini file.
	ReadSection (see page 4272)	Reads all the key names from a specified section of an ini file into a string list.
	ReadSections (see page 4273)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 4274)	Reads the values from all keys in a section of an ini file into a string list.
	ReadString (see page 4274)	Retrieves a string value from an ini file.
	ReadTime (see page 4277)	Retrieves a time value from an ini file.
	SectionExists (see page 4278)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 4278)	Flushes buffered ini file data to disk.
	ValueExists (see page 4278)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 4279)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 4279)	Writes a Boolean value to an ini file.
	WriteDate (see page 4282)	Writes a date value to an ini file.
	WriteDateTime (see page 4283)	Writes a date-time value to an ini file.
	WriteFloat (see page 4283)	Writes a float value to an ini file.
	WriteInteger (see page 4284)	Writes an integer value to an ini file.
	WriteString (see page 4287)	Writes a string value to an ini file.
	WriteTime (see page 4290)	Writes a time value to an ini file.

36.1.2.3.1 TCustomIniFile.DeleteKey Method

Erases the data value associated with an ini file entry.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); virtual; abstract;
```

C++

```
virtual __fastcall DeleteKey(const UnicodeString Section, const UnicodeString Ident) = 0;
```

Description

Call DeleteKey to erase a data value associated with an ini file entry. Section is string containing the name of an ini file section, and Ident is a string containing the name of the key for which to set a nil (Delphi) or NULL (C++) value.

Note: Attempting to erase a data value in a nonexistent section or attempting to erase data from a nonexistent key are not errors. In these cases, DeleteKey creates the section and key and sets its initial value to an empty value.

See Also

EraseSection (see page 4264)

36.1.2.3.2 TCustomIniFile.EraseSection Method

Erases an entire section of an ini file.

Pascal

```
procedure EraseSection(const Section: string); virtual; abstract;
```

C++

```
virtual __fastcall EraseSection(const UnicodeString Section) = 0;
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an ini file. Section identifies the ini file section to remove. If a section cannot be removed, an exception is raised.

See Also

SectionExists (see page 4278)

DeleteKey (see page 4263)

36.1.2.3.3 TCustomIniFile.ReadBinaryStream Method

Retrieves a binary value from the ini file.

Pascal

```
function ReadBinaryStream(const Section: string; const Name: string; Value: TStream): Integer;
virtual;
```

C++

```
virtual __fastcall int ReadBinaryStream(const UnicodeString Section, const UnicodeString Name,
TStream Value);
```

Description

Call ReadBinaryStream to read a binary value into a specified stream. After reading the binary value, which it converts from a series of hexadecimal characters, from the ini file into the current position of the stream, ReadBinaryStream returns the number of bytes written to the stream. If the stream is a memory stream (TMemoryStream or one of its descendants), the stream is left positioned to the same byte as it was before the call to ReadBinaryStream (immediately before the new binary value).

Section is the section that contains the desired key.

Name is the key under which the binary data is stored.

Value is a stream to which the binary value is written.

Note: When used with a Windows .ini file, ReadBinaryStream is limited to 1023 characters,

See Also

ReadString (see page 4274)

writebinarystream (see page 4279)

36.1.2.3.4 TCustomIniFile.ReadBool Method

Retrieves a Boolean value from an ini file.

Pascal

```
function ReadBool(const Section: string; const Ident: string; Default: Boolean): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ReadBool(const UnicodeString Section, const UnicodeString Ident,
Boolean Default);
```

Description

Call ReadBool to read a Boolean value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the Boolean value. Default is the Boolean value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
```

```
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
```

```
        false: WindowState := wsNormal;
      end;

    finally
      SettingsFile.Free;
    end;
  end;

  procedure TForm2.btStoreClick(Sender: TObject);
  var
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadString (see page 4274)

ReadTime (see page 4277)

36.1.2.3.5 TCustomIniFile.ReadDate Method

Retrieves a date value from an ini file.

Pascal

```
function ReadDate(const Section: string; const Name: string; Default: TDateTime): TDateTime;
virtual;
```

C++

```
virtual __fastcall TDateTime ReadDate(const UnicodeString Section, const UnicodeString Name,
TDateTime Default);
```

Description

Call ReadDate to read a date value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date value. Default is the date value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadString (see page 4274)

ReadTime (see page 4277)

36.1.2.3.6 TCustomIniFile.ReadDateTime Method

Retrieves a date-time value from an ini file.

Pascal

```
function ReadDateTime(const Section: string; const Name: string; Default: TDateTime):
TDateTime; virtual;
```

C++

```
virtual __fastcall TDateTime ReadDateTime(const UnicodeString Section, const UnicodeString
Name, TDateTime Default);
```

Description

Call ReadDateTime to read a date-time value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date-time value. Default is the date-time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadString (see page 4274)

ReadTime (see page 4277)

36.1.2.3.7 TCustomIniFile.ReadFloat Method

Retrieves a float value from an ini file.

Pascal

```
function ReadFloat(const Section: string; const Name: string; Default: Double): Double;
virtual;
```

C++

```
virtual __fastcall Double ReadFloat(const UnicodeString Section, const UnicodeString Name,
Double Default);
```

Description

Call ReadFloat to read a float value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the float value. Default is the float value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadInteger (see page 4269)

ReadString (see page 4274)

ReadTime (see page 4277)

36.1.2.3.8 TCustomIniFile.ReadInteger Method

Retrieves an integer value from an ini file.

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
virtual;
```

C++

```
virtual __fastcall Longint ReadInteger(const UnicodeString Section, const UnicodeString Ident,
Longint Default);
```

Description

Call ReadInteger to read an integer value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the integer value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
```

```
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
```

```
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadString (see page 4274)

ReadTime (see page 4277)

36.1.2.3.9 TCustomIniFile.ReadSection Method

Reads all the key names from a specified section of an ini file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ReadSection(const UnicodeString Section, TStrings Strings) = 0;
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an ini file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant such as a string list, or to a component property, such as Items for a TListBox component.

See Also

SectionExists (see page 4278)

ReadSections (see page 4273)

ReadSectionValues (see page 4274)

36.1.2.3.10 ReadSections Method

This is the overview for the ReadSections method overload.

Overload List

	Name	Description
	TCustomIniFile.ReadSections (string, TStrings) (see page 4273)	Reads the names of all sections in an ini file into a string list.
	TCustomIniFile.ReadSections (TStrings) (see page 4273)	Reads the names of all sections in an ini file into a string list.

36.1.2.3.10.1 TCustomIniFile.ReadSections Method (TStrings)

Reads the names of all sections in an ini file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); virtual; abstract; overload;
procedure ReadSections(const Section: string; Strings: TStrings); virtual; overload;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings) = 0;
virtual __fastcall ReadSections(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an ini file into a string list.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Section identifies the section from which to retrieve a list of key names.

See Also

ReadSection (see page 4272)

ReadSectionValues (see page 4274)

36.1.2.3.11 TCustomIniFile.ReadSectionValues Method

Reads the values from all keys in a section of an ini file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ReadSectionValues(const UnicodeString Section, TStrings Strings) = 0;
```

Description

Call ReadSectionValues to read the values from all keys in a specified section of an ini file into a string list object. The values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component.

See Also

SectionExists (see page 4278)

ReadSection (see page 4272)

ReadSections (see page 4273)

36.1.2.3.12 TCustomIniFile.ReadString Method

Retrieves a string value from an ini file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ReadString(const UnicodeString Section, const UnicodeString
Ident, const UnicodeString Default) = 0;
```

Description

Call ReadString to read a string value from a .ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

Note: On Windows, if the string in the .ini file is surrounded by quotation marks, the return value has those marks stripped away. On Linux, the quotation marks are preserved.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
```

```
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
```

```
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;

end;
```

```
function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadTime (see page 4277)

ReadBinaryStream (see page 4264)

36.1.2.3.13 TCustomIniFile.ReadTime Method

Retrieves a time value from an ini file.

Pascal

```
function ReadTime(const Section: string; const Name: string; Default: TDateTime): TDateTime;
virtual;
```

C++

```
virtual __fastcall TDateTime ReadTime(const UnicodeString Section, const UnicodeString Name,
TDateTime Default);
```

Description

Call ReadTime to read a time value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the time value. Default is the time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadString (see page 4274)

36.1.2.3.14 TCustomIniFile.SectionExists Method

Indicates whether a section exists in the ini file.

Pascal

```
function SectionExists(const Section: string): Boolean;
```

C++

```
__fastcall Boolean SectionExists(const UnicodeString Section);
```

Description

Use SectionExists to determine whether a section exists within the ini file specified in FileName (see page 4291).

Section is the ini file section SectionExists determines the existence of.

SectionExists returns a Boolean value that indicates whether the section in question exists.

See Also

FileName (see page 4291)

ValueExists (see page 4278)

36.1.2.3.15 TCustomIniFile.UpdateFile Method

Flushes buffered ini file data to disk.

Pascal

```
procedure UpdateFile; virtual; abstract;
```

C++

```
virtual __fastcall UpdateFile() = 0;
```

Description

Call UpdateFile to flush buffered reads from and writes to the ini file to disk. [SK comment -- UpdateFile is useful under Windows95but has no effect under Windows NT as NT does not buffer ini file reads and writes. do not delete!]

See Also

FileName (see page 4291)

36.1.2.3.16 TCustomIniFile.ValueExists Method

Indicates whether a key exists in the ini file.

Pascal

```
function ValueExists(const Section: string; const Ident: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ValueExists(const UnicodeString Section, const UnicodeString Ident);
```

Description

Use ValueExists to determine whether a key exists in the ini file specified in FileName (see page 4291).

Section is the section in the ini file in which to search for the key.

Ident is the name of the key to search for.

ValueExists returns a boolean value that indicates whether the key exists in the specified section.

See Also

FileName (see page 4291)

SectionExists (see page 4278)

36.1.2.3.17 TCustomIniFile.WriteBinaryStream Method

Pascal

```
procedure WriteBinaryStream(const Section: string; const Name: string; Value: TStream);
virtual;
```

C++

```
virtual __fastcall WriteBinaryStream(const UnicodeString Section, const UnicodeString Name,
TStream Value);
```

Description

This is WriteBinaryStream, a member of class TCustomIniFile.

36.1.2.3.18 TCustomIniFile.WriteBool Method

Writes a Boolean value to an ini file.

Pascal

```
procedure WriteBool(const Section: string; const Ident: string; Value: Boolean); virtual;
```

C++

```
virtual __fastcall WriteBool(const UnicodeString Section, const UnicodeString Ident, Boolean
Value);
```

Description

Call WriteBool to write a Boolean value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the Boolean value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteBool creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
```

```
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
```

```
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;
```

```
end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteString (see page 4287)

WriteTime (see page 4290)

36.1.2.3.19 TCustomIniFile.WriteDate Method

Writes a date value to an ini file.

Pascal

```
procedure WriteDate(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteDate(const UnicodeString Section, const UnicodeString Name, TDateTime
Value);
```

Description

Call WriteDate to write a date value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDate creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteString (see page 4287)

WriteTime (see page 4290)

36.1.2.3.20 TCustomIniFile.WriteDateTime Method

Writes a date-time value to an ini file.

Pascal

```
procedure WriteDateTime(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteDateTime(const UnicodeString Section, const UnicodeString Name,
TDateTime Value);
```

Description

Call WriteDateTime to write a date-time value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date-time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDateTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteString (see page 4287)

WriteTime (see page 4290)

36.1.2.3.21 TCustomIniFile.WriteFloat Method

Writes a float value to an ini file.

Pascal

```
procedure WriteFloat(const Section: string; const Name: string; Value: Double); virtual;
```

C++

```
virtual __fastcall WriteFloat(const UnicodeString Section, const UnicodeString Name, Double
Value);
```

Description

Call WriteFloat to write a float value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the float value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteFloat creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteInteger (see page 4284)

WriteString (see page 4287)

WriteTime (see page 4290)

36.1.2.3.22 TCustomIniFile.WriteInteger Method

Writes an integer value to an ini file.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint); virtual;
```

C++

```
virtual __fastcall WriteInteger(const UnicodeString Section, const UnicodeString Ident,
Longint Value);
```

Description

Call WriteInteger to write an integer value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the integer value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteInteger creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}
```

```
void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
```

```
      Read all saved values from the last session. The section name
      is the name of the form. Also use form's properties as defaults
      }
      Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
      Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
      Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
      Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
      Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

      { Load last window state }
      case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
        true : WindowState := wsMaximized;
        false: WindowState := wsNormal;
      end;

    finally
      SettingsFile.Free;
    end;
  end;

  procedure TForm2.btStoreClick(Sender: TObject);
  var
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
```

```
      end;
    end;
  end;
```

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteString (see page 4287)

WriteTime (see page 4290)

36.1.2.3.23 TCustomIniFile.WriteString Method

Writes a string value to an ini file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
virtual; abstract;
```

C++

```
virtual __fastcall WriteString(const UnicodeString Section, const UnicodeString Ident, const
UnicodeString Value) = 0;
```

Description

Call WriteString to write a string value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }
```

```
    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
```

```
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;

end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
```

```
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteTime (see page 4290)

36.1.2.3.24 TCustomIniFile.WriteTime Method

Writes a time value to an ini file.

Pascal

```
procedure WriteTime(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteTime(const UnicodeString Section, const UnicodeString Name, TDateTime
Value);
```

Description

Call WriteTime to write a time value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteString (see page 4287)

36.1.2.4 TCustomIniFile Properties

The properties of the TCustomIniFile class are listed here.

Public Properties

	Name	Description
	FileName (see page 4291)	Contains the name of the ini file from which to read and to which to write information.

36.1.2.4.1 TCustomIniFile.FileName Property

Contains the name of the ini file from which to read and to which to write information.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Read FileName to retrieve the name of the ini file that this object can be used to access. The methods that read and write ini file information automatically handle opening and closing the file specified by FileName.

FileName is read-only. The name of the ini file the object uses is supplied as the parameter for the constructor.

See Also

Create (see page 4262)

36.1.3 THashedStringList Class

THashedStringList maintains a list of strings using an internal hash table.

Class Hierarchy

Pascal

```
THashedStringList = class(TStringList);
```

C++

```
class THashedStringList : public TStringList;
```

File

IniFiles

Description

THashedStringList is a string list that uses a hash table internally to speed the process of locating strings. It is used internally by TMemIniFile (see page 4307) to manage the strings from an INI file, but can be used in the same way as any other string list. By using THashedStringList instead of TStringList, you can improve performance when the list contains a large number of strings.

See Also

TStringList

TMemIniFile (see page 4307)

Working with Lists

36.1.3.1 THashedStringList Members

The following tables list the members exposed by THashedStringList.

Protected Methods

	Name	Description
	Changed (see page 4292)	This is Changed, a member of class THashedStringList.

Public Methods

	Name	Description
	Destroy (see page 4292)	Destroys an instance of THashedStringList (see page 4291).
	IndexOf (see page 4293)	Returns the position of a string in the list.
	IndexOfName (see page 4293)	Returns the position of the first string with the form Name=Value with the specified name part.

36.1.3.2 THashedStringList.Destroy Destructor

Destroys an instance of THashedStringList (see page 4291).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THashedStringList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THashedStringList (see page 4291) reference is not nil and only then calls Destroy.

See Also

System::TObject::Free

36.1.3.3 THashedStringList Methods

The methods of the THashedStringList class are listed here.

Protected Methods

	Name	Description
	Changed (see page 4292)	This is Changed, a member of class THashedStringList.

Public Methods

	Name	Description
	IndexOf (see page 4293)	Returns the position of a string in the list.
	IndexOfName (see page 4293)	Returns the position of the first string with the form Name=Value with the specified name part.

36.1.3.3.1 THashedStringList.Changed Method

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class THashedStringList.

36.1.3.3.2 THashedStringList.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const UnicodeString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. A string matches S if it is identical to S or differs only in case.

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

Find

IndexOfName (see page 4293)

IndexOfObject

Strings

36.1.3.3.3 THashedStringList.IndexOfName Method

Returns the position of the first string with the form Name=Value with the specified name part.

Pascal

```
function IndexOfName(const Name: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOfName(const UnicodeString Name);
```

Description

Call IndexOfName to locate the first occurrence of a string with the form Name=Value where the name part is equal to the Name parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

The Name that identifies the string is to the left of the equal sign (=), and the current Value of the Name identifier is on the right side. There should be no spaces present before or after the equal sign.

Note: If there is more than one string with a name portion matching the Name parameter, IndexOfName returns the position of the first such string.

See Also

IndexOf (see page 4293)

Names

Values

IndexOfObject

36.1.4 TIniFile Class

TIniFile stores and retrieves application-specific information and settings from INI files.

Class Hierarchy

Pascal

```
TIniFile = class(TCustomIniFile);
```

C++

```
class TIniFile : public TCustomIniFile;
```

File

IniFiles

Description

TIniFile enables handling the storage and retrieval of application-specific information and settings in a standard INI file. An INI file stores information in logical groupings, called "sections." Within each section, actual data values are stored in named keys. Keys take the form:

```
<keyname>=<value>
```

A FileName (see page 4291) is passed to the TIniFile constructor and identifies the INI file that the object accesses.

Note: On Windows, a related object, TMemIniFile (see page 4307), works the same way as TIniFile, but buffers writes in memory to minimize disk access.

Note: On Linux, TIniFile is the same as TMemIniFile (see page 4307).

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
```

```
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
```

```
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;

end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
```

```
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

TRegistry (see page 6610)

TRegistryIniFile (see page 6663)

36.1.4.1 TIniFile Members

The following tables list the members exposed by TIniFile.

Public Methods

	Name	Description
	Create (see page 4262)	Creates a TCustomIniFile (see page 4258) object for an application.
	DeleteKey (see page 4299)	Removes an INI file entry for a specified key value.
	Destroy (see page 4298)	Disposes of an object instance.
	EraseSection (see page 4299)	Erases an entire section of an INI file.
	ReadBinaryStream (see page 4264)	Retrieves a binary value from the ini file.
	ReadBool (see page 4265)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 4268)	Retrieves a date value from an ini file.
	ReadDateTime (see page 4268)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 4269)	Retrieves a float value from an ini file.
	ReadInteger (see page 4269)	Retrieves an integer value from an ini file.
	ReadSection (see page 4299)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 4300)	Reads the names of all sections in an INI file into a string list.
	ReadSections (see page 4273)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 4300)	Reads the values from all keys within a section of an INI file into a string list.
	ReadString (see page 4301)	Retrieves a string value from an INI file.
	ReadTime (see page 4277)	Retrieves a time value from an ini file.
	SectionExists (see page 4278)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 4304)	Flushes buffered INI file data to disk.
	ValueExists (see page 4278)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 4279)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 4279)	Writes a Boolean value to an ini file.
	WriteDate (see page 4282)	Writes a date value to an ini file.
	WriteDateTime (see page 4283)	Writes a date-time value to an ini file.
	WriteFloat (see page 4283)	Writes a float value to an ini file.
	WriteInteger (see page 4284)	Writes an integer value to an ini file.
	WriteString (see page 4304)	Writes a string value to an INI file.
	WriteTime (see page 4290)	Writes a time value to an ini file.

Public Properties

	Name	Description
	FileName (see page 4291)	Contains the name of the ini file from which to read and to which to write information.

36.1.4.2 TIniFile.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIniFile();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

36.1.4.3 TIniFile Methods

The methods of the TIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 4299)	Removes an INI file entry for a specified key value.
	EraseSection (see page 4299)	Erases an entire section of an INI file.
	ReadSection (see page 4299)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 4300)	Reads the names of all sections in an INI file into a string list.
	ReadSectionValues (see page 4300)	Reads the values from all keys within a section of an INI file into a string list.
	ReadString (see page 4301)	Retrieves a string value from an INI file.
	UpdateFile (see page 4304)	Flushes buffered INI file data to disk.
	WriteString (see page 4304)	Writes a string value to an INI file.

36.1.4.3.1 TIniFile.DeleteKey Method

Removes an INI file entry for a specified key value.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); override;
```

C++

```
virtual __fastcall DeleteKey(const UnicodeString Section, const UnicodeString Ident);
```

Description

Call DeleteKey to erase an INI file entry. Section is string containing the name of an INI file section, and Ident is a string containing the name of the key from for which to set a nil (Delphi) or NULL (C++) value.

This version of DeleteKey is only available if the application is compiled for Windows. On Linux, DeleteKey is inherited from TMemIniFile (see page 4307).

Note: Attempting to delete a key in a nonexistent section or attempting to delete a nonexistent key are not errors. In these cases, DeleteKey does nothing.

See Also

EraseSection (see page 4299)

36.1.4.3.2 TIniFile.EraseSection Method

Erases an entire section of an INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const UnicodeString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

This version of EraseSection is only available if the application is compiled for Windows. On Linux, EraseSection is inherited from TMemIniFile (see page 4307).

See Also

DeleteKey (see page 4299)

36.1.4.3.3 TIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the TStrings descendant to hold the retrieved names. Strings can point to a TStrings descendant such as a string list (TStringList), or to a component property that is of type TStrings, such as the Items property of TListBox.

This version of ReadSection is only available if the application is compiled for Windows. On Linux, ReadSection is inherited from TMemIniFile (see page 4307).

See Also

ReadSections (see page 4300)

ReadSectionValues (see page 4300)

36.1.4.3.4 TIniFile.ReadSections Method

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list object.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component.

This version of ReadSections is only available if the application is compiled for Windows. On Linux, ReadSections is inherited from TMemIniFile (see page 4307).

See Also

ReadSection (see page 4299)

ReadSectionValues (see page 4300)

36.1.4.3.5 TIniFile.ReadSectionValues Method

Reads the values from all keys within a section of an INI file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read the keys, and the values from all keys, within a specified section of an INI file into a string list object (TStrings descendant). The key names and values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

This version of ReadSectionValues is only available if the application is compiled for Windows. On Linux, ReadSectionValues is inherited from TMemIniFile (see page 4307).

See Also

ReadSection (see page 4299)

ReadSections (see page 4300)

36.1.4.3.6 TIniFile.ReadString Method

Retrieves a string value from an INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall UnicodeString ReadString(const UnicodeString Section, const UnicodeString
Ident, const UnicodeString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

This version of ReadString is only available if the application is compiled for Windows. On Linux, ReadString is inherited from TMemIniFile (see page 4307).

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }
```

```
    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();
```

```
  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;

end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
```

```
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

ReadBool (see page 4265)

ReadInteger (see page 4269)

36.1.4.3.7 TIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to flush buffered reads from and writes to the INI file to disk. UpdateFile is useful under Windows9x, but has no effect under Windows NT because NT does not buffer INI file reads and writes.

This version of UpdateFile is only available if the application is compiled for Windows. On Linux, Update is inherited from TMemIniFile (see page 4307).

See Also

FileName (see page 4291)

TMemIniFile (see page 4307)

36.1.4.3.8 TIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
override;
```

C++

```
virtual __fastcall WriteString(const UnicodeString Section, const UnicodeString Ident, const
UnicodeString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

This version of WriteString is only available if the application is compiled for Windows. On Linux, WriteString is inherited from TMemIniFile (see page 4307).

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
```

```
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
```

```
    SettingsFile.Free;
  end;

end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

WriteBool (see page 4279)

WriteInteger (see page 4284)

36.1.5 TMemIniFile Class

TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.

Class Hierarchy

Pascal

```
TMemIniFile = class(TCustomIniFile);
```

C++

```
class TMemIniFile : public TCustomIniFile;
```

File

IniFiles

Description

Use TMemIniFile to store and retrieve application-specific information and settings in a Windows INI file. An INI file stores information in logical groupings, called "sections." Within each section, actual data values are stored in named keys. Keys take the form:

```
<keyname>=<value>
```

TMemIniFile buffers all changes to the INI file. The INI file is read once, when the object is first created. Data from the INI file is stored in nested string lists. Each section in the INI file occupies one element in the top-most string list, and each element in this may itself contain a string list. Each element in each of the contained string list represents a key within the section. After the data is read, any changes to the data are stored in memory. To write the data from memory back to the associated INI file, call the UpdateFile (see page 4320) method.

All TMemIniFile methods to read, write, and erase sections, keys, and values operate on the in-memory copy of the INI file.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
```

```
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
```

```
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

TStrings

TIniFile (see page 4294)

TCustomIniFile (see page 4258)

TRegistry (see page 6610)

TRegistryIniFile (see page 6663)

36.1.5.1 TMemIniFile Members

The following tables list the members exposed by TMemIniFile.

Public Methods

	Name	Description
	Clear (see page 4313)	Erases all data from the INI file in memory.
	Create (see page 4311)	This is the overview for the Create constructor overload.
	DeleteKey (see page 4313)	Deletes a specified entry from the .ini file.

	Destroy (see page 4312)	Frees the memory associated with the TMemIniFile (see page 4307) object.
	EraseSection (see page 4314)	Erases an entire section of the INI file.
	GetStrings (see page 4314)	Populates a string list with INI file data.
	ReadBinaryStream (see page 4264)	Retrieves a binary value from the ini file.
	ReadBool (see page 4265)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 4268)	Retrieves a date value from an ini file.
	ReadDateTime (see page 4268)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 4269)	Retrieves a float value from an ini file.
	ReadInteger (see page 4269)	Retrieves an integer value from an ini file.
	ReadSection (see page 4314)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 4315)	Reads the names of all sections in an INI file into a string list.
	ReadSections (see page 4273)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 4315)	Reads the values from all keys within a section of an INI file into a string list.
	ReadString (see page 4316)	Retrieves a string value from the INI file.
	ReadTime (see page 4277)	Retrieves a time value from an ini file.
	Rename (see page 4319)	Associates the TMemIniFile (see page 4307) with a different INI file.
	SectionExists (see page 4278)	Indicates whether a section exists in the ini file.
	SetStrings (see page 4319)	Populates the memory buffer with contents of a string list.
	UpdateFile (see page 4320)	Flushes buffered INI file data to disk.
	ValueExists (see page 4278)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 4279)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 4279)	Writes a Boolean value to an ini file.
	WriteDate (see page 4282)	Writes a date value to an ini file.
	WriteDateTime (see page 4283)	Writes a date-time value to an ini file.
	WriteFloat (see page 4283)	Writes a float value to an ini file.
	WriteInteger (see page 4284)	Writes an integer value to an ini file.
	WriteString (see page 4320)	Writes a string value to an INI file.
	WriteTime (see page 4290)	Writes a time value to an ini file.

Public Properties

	Name	Description
	CaseSensitive (see page 4324)	Controls whether section and key names are handled in a case-sensitive manner.
	Encoding (see page 4324)	This is Encoding, a member of class TMemIniFile.
	FileName (see page 4291)	Contains the name of the ini file from which to read and to which to write information.

36.1.5.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TMemIniFile.Create (string) (see page 4311)	Creates a TMemIniFile (see page 4307) object for an application.
	TMemIniFile.Create (string, TEncoding) (see page 4311)	Creates a TMemIniFile (see page 4307) object for an application.

36.1.5.2.1 TMemIniFile.Create Constructor (string)

Creates a TMemIniFile (see page 4307) object for an application.

Pascal

```
constructor Create(const FileName: string); overload;
constructor Create(const FileName: string; Encoding: TEncoding); overload;
```

C++

```
__fastcall TMemIniFile(const UnicodeString FileName);
__fastcall TMemIniFile(const UnicodeString FileName, TEncoding Encoding);
```

Description

Call Create to instantiate a TMemIniFile (see page 4307) object for an application. Create sets the FileName (see page 4291) property, and loads a copy of the INI file into memory, in an internal string list.

FileName (see page 4291) is the name of the INI file the TMemIniFile (see page 4307) uses.

Note: On Windows, INI files are meant to be stored in the \WINDOWS directory or the application's directory. To work with an INI file in another location, specify the full path name of the file in FileName (see page 4291).

See Also

FileName (see page 4291)

System::TObject::Create

36.1.5.3 TMemIniFile.Destroy Destructor

Frees the memory associated with the TMemIniFile (see page 4307) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMemIniFile();
```

Description

Do not call Destroy directly; instead call Free.

Warning: The destructor does not write the current memory image to disk. Be sure to save changes made to TMemIniFile (see page 4307) by calling UpdateFile (see page 4320) before destroying a TMemIniFile (see page 4307) instance.

See Also

System::TObject::Free

UpdateFile (see page 4320)

Create (see page 4311)

36.1.5.4 TMemIniFile Methods

The methods of the TMemIniFile class are listed here.

Public Methods

	Name	Description
	Clear (see page 4313)	Erases all data from the INI file in memory.
	DeleteKey (see page 4313)	Deletes a specified entry from the .ini file.
	EraseSection (see page 4314)	Erases an entire section of the INI file.

	GetStrings (see page 4314)	Populates a string list with INI file data.
V	ReadSection (see page 4314)	Reads all the key names from a specified section of an INI file into a string list.
V	ReadSections (see page 4315)	Reads the names of all sections in an INI file into a string list.
V	ReadSectionValues (see page 4315)	Reads the values from all keys within a section of an INI file into a string list.
V	ReadString (see page 4316)	Retrieves a string value from the INI file.
	Rename (see page 4319)	Associates the TMemIniFile (see page 4307) with a different INI file.
	SetStrings (see page 4319)	Populates the memory buffer with contents of a string list.
V	UpdateFile (see page 4320)	Flushes buffered INI file data to disk.
V	WriteString (see page 4320)	Writes a string value to an INI file.

36.1.5.4.1 TMemIniFile.Clear Method

Erases all data from the INI file in memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to erase all data from the INI file that is currently buffered in memory. All sections, keys, and values are erased.

Warning: No exception is generated when using Clear and the data has not been saved to the INI file with the UpdateFile (see page 4320) method.

See Also

UpdateFile (see page 4320)

SetStrings (see page 4319)

36.1.5.4.2 TMemIniFile.DeleteKey Method

Deletes a specified entry from the .ini file.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); override;
```

C++

```
virtual __fastcall DeleteKey(const UnicodeString Section, const UnicodeString Ident);
```

Description

Call DeleteKey to erase .ini file entry. Section is string containing the name of a .ini file section, and Ident is a string containing the name of the key to remove.

Note: Attempting to erase a key in a nonexistent section or attempting to erase a nonexistent key are not errors. In these cases, DeleteKey does nothing.

DeleteKey only affects the in-memory copy of the .ini file, not the copy on disk. The change is not written to disk until you call UpdateFile (see page 4320).

See Also

SectionExists (see page 4278)

ValueExists (see page 4278)

EraseSection (see page 4314)

UpdateFile (see page 4278)

CaseSensitive (see page 4324)

36.1.5.4.3 TMemIniFile.EraseSection Method

Erases an entire section of the INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const UnicodeString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

EraseSection only affects the in-memory copy of the INI file, not the copy on disk. The change is not written to disk until you call UpdateFile (see page 4320).

See Also

SectionExists (see page 4278)

DeleteKey (see page 4313)

UpdateFile (see page 4320)

CaseSensitive (see page 4324)

36.1.5.4.4 TMemIniFile.GetStrings Method

Populates a string list with INI file data.

Pascal

```
procedure GetStrings(List: TStrings);
```

C++

```
__fastcall GetStrings(TStrings List);
```

Description

Call GetStrings to copy the INI file data currently stored in memory into a pre-existing string list.

List is a string list object, created and maintained by the application, into which the INI file data is copied.

See Also

SetStrings (see page 4319)

36.1.5.4.5 TMemIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string to hold the retrieved names. Strings can point to a TStrings descendant such as a string list, or to a component property, such as Items for a TListBox component.

Data read by ReadSection comes from the in-memory copy of the INI file, not the copy on disk.

See Also

SectionExists (see page 4278)

ReadSections (see page 4315)

ReadSectionValues (see page 4315)

CaseSensitive (see page 4324)

36.1.5.4.6 TMemIniFile.ReadSections Method

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Data read by ReadSections comes from the in-memory copy of the INI file, not the copy on disk.

See Also

ReadSection (see page 4314)

ReadSectionValues (see page 4315)

36.1.5.4.7 TMemIniFile.ReadSectionValues Method

Reads the values from all keys within a section of an INI file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read the values from all keys within a specified section of an INI file into a string list object. The values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Data read by ReadSectionValues comes from the in-memory copy of the INI file, not the copy on disk.

See Also

SectionExists (see page 4278)

ReadSection (see page 4314)

ReadSections (see page 4315)

CaseSensitive (see page 4324)

36.1.5.4.8 TMemIniFile.ReadString Method

Retrieves a string value from the INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall UnicodeString ReadString(const UnicodeString Section, const UnicodeString
Ident, const UnicodeString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

The ReadString method inherited from TCustomIniFile (see page 4258) is modified to operate on the in-memory copy of the INI file data. The read methods for all other data types use ReadString internally and then translate to the appropriate data type.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
```

```
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
```

```
    SettingsFile : TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    try
      {
      Read all saved values from the last session. The section name
      is the name of the form. Also use form's properties as defaults
      }
      Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
      Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
      Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
      Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
      Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

      { Load last window state }
      case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
        true : WindowState := wsMaximized;
        false: WindowState := wsNormal;
      end;

    finally
      SettingsFile.Free;
    end;
  end;

  procedure TForm2.btStoreClick(Sender: TObject);
  var
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
```

```
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

ReadBool (see page 4265)

ReadDate (see page 4268)

ReadDateTime (see page 4268)

ReadFloat (see page 4269)

ReadInteger (see page 4269)

ReadTime (see page 4277)

CaseSensitive (see page 4324)

36.1.5.4.9 TMemIniFile.Rename Method

Associates the TMemIniFile (see page 4307) with a different INI file.

Pascal

```
procedure Rename(const FileName: string; Reload: Boolean);
```

C++

```
__fastcall Rename(const UnicodeString FileName, Boolean Reload);
```

Description

Call Rename to change the INI file used by the TMemIniFile (see page 4307) object to a file other than the one specified when the object was created.

FileName (see page 4291) is name of the new INI file to use.

Reload determines whether data is immediately loaded into memory from the INI file specified in FileName (see page 4291). When Reload is true, the TMemIniFile (see page 4307) object is repopulated with the data from the specified file. When Reload is false, the in-memory image is unchanged. This is useful, for example, when copying the data from one INI file to another file.

See Also

FileName (see page 4291)

Create (see page 4311)

36.1.5.4.10 TMemIniFile.SetStrings Method

Populates the memory buffer with contents of a string list.

Pascal

```
procedure SetStrings(List: TStrings);
```

C++

```
__fastcall SetStrings(TStrings List);
```

Description

Call SetStrings to populate the in-memory copy of the INI file data with the contents of a string list.

List is a string list object, created and maintained by the application, from which the INI file data is copied.

SetStrings uses the Clear (see page 4313) method to erase any existing INI file data prior to copying the new data.

SetStrings can be used to reread data from the INI file from disk after it is erased with the Clear (see page 4313) method. The LoadFromFile method of a string list can be used to retrieve the contents of the INI file and then SetStrings can copy the data into the memory buffer.

See Also

LoadFromFile

Clear (see page 4313)

GetStrings (see page 4314)

36.1.5.4.11 TMemIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to copy INI file data stored in memory to the copy of the INI file on disk UpdateFile overwrites all data in the disk copy of the INI file with the INI file data stored in memory.

If the Rename (see page 4319) method is used to change the INI file the TMemIniFile (see page 4307) uses, UpdateFile copies the INI file data in memory to a file of the new name. If the file does not already exist, it is created. If the new file already exists, it is overwritten.

See Also

FileName (see page 4291)

36.1.5.4.12 TMemIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
override;
```

C++

```
virtual __fastcall WriteString(const UnicodeString Section, const UnicodeString Ident, const
UnicodeString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write.

Ident is the name of the key for which to set a value. Value is the string value to write.

The WriteString method inherited from TCustomIniFile (see page 4258) is modified to operate on the in-memory copy of the INI file data. The write methods for all other data types use WriteString internally after data values are translated from the original data type.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }
```

```
    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();
```

```
    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

WriteBool (see page 4279)

WriteDate (see page 4282)

WriteDateTime (see page 4283)

WriteFloat (see page 4283)

WriteInteger (see page 4284)

WriteTime (see page 4290)

CaseSensitive (see page 4324)

36.1.5.5 TMemIniFile Properties

The properties of the TMemIniFile class are listed here.

Public Properties

	Name	Description
	CaseSensitive (see page 4324)	Controls whether section and key names are handled in a case-sensitive manner.
	Encoding (see page 4324)	This is Encoding, a member of class TMemIniFile.

36.1.5.5.1 TMemIniFile.CaseSensitive Property

Controls whether section and key names are handled in a case-sensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether the names of sections and keys should be treated in a case-sensitive or case-insensitive manner. Set CaseSensitive to true to allow multiple sections or keys that differ only based on case. Set CaseSensitive to false to treat names that differ only by case as the same.

See Also

ReadSection (see page 4314)

ReadString (see page 4316)

SectionExists (see page 4278)

ValueExists (see page 4278)

36.1.5.5.2 TMemIniFile.Encoding Property

Pascal

```
property Encoding: TEncoding;
```

C++

```
__property TEncoding Encoding;
```

Description

This is Encoding, a member of class TMemIniFile.

36.1.6 TStringHash Class

Class Hierarchy

IniFiles.TStringHash

Pascal

```
TStringHash = class;
```

C++

```
class TStringHash;
```

File

IniFiles

Description

This is class IniFiles.TStringHash.

36.1.6.1 TStringHash Members

The following tables list the members exposed by TStringHash.

Protected Methods

	Name	Description
	Find (see page 4326)	This is Find, a member of class TStringHash.
	HashOf (see page 4327)	This is HashOf, a member of class TStringHash.

Public Methods

	Name	Description
	Add (see page 4326)	This is Add, a member of class TStringHash.
	Clear (see page 4326)	This is Clear, a member of class TStringHash.
	Create (see page 4325)	This is Create, a member of class TStringHash.
	Destroy (see page 4325)	This is Destroy, a member of class TStringHash.
	Modify (see page 4327)	This is Modify, a member of class TStringHash.
	Remove (see page 4327)	This is Remove, a member of class TStringHash.
	ValueOf (see page 4327)	This is ValueOf, a member of class TStringHash.

36.1.6.2 TStringHash.Create Constructor

Pascal

```
constructor Create(Size: Cardinal = 256);
```

C++

```
__fastcall TStringHash(unsigned Size = 256);
```

Description

This is Create, a member of class TStringHash.

36.1.6.3 TStringHash.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStringHash();
```

Description

This is Destroy, a member of class TStringHash.

36.1.6.4 TStringHash Methods

The methods of the TStringHash class are listed here.

Protected Methods

	Name	Description
	Find (see page 4326)	This is Find, a member of class TStringHash.
	HashOf (see page 4327)	This is HashOf, a member of class TStringHash.

Public Methods

	Name	Description
	Add (see page 4326)	This is Add, a member of class TStringHash.
	Clear (see page 4326)	This is Clear, a member of class TStringHash.
	Modify (see page 4327)	This is Modify, a member of class TStringHash.
	Remove (see page 4327)	This is Remove, a member of class TStringHash.
	ValueOf (see page 4327)	This is ValueOf, a member of class TStringHash.

36.1.6.4.1 TStringHash.Add Method

Pascal

```
procedure Add(const Key: string; Value: Integer);
```

C++

```
__fastcall Add(const UnicodeString Key, int Value);
```

Description

This is Add, a member of class TStringHash.

36.1.6.4.2 TStringHash.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TStringHash.

36.1.6.4.3 TStringHash.Find Method

Pascal

```
function Find(const Key: string): PPHashItem;
```

C++

```
__fastcall PPHashItem Find(const UnicodeString Key);
```

Description

This is Find, a member of class TStringHash.

36.1.6.4.4 TStringHash.HashOf Method

Pascal

```
function HashOf(const Key: string): Cardinal; virtual;
```

C++

```
virtual __fastcall unsigned HashOf(const UnicodeString Key);
```

Description

This is HashOf, a member of class TStringHash.

36.1.6.4.5 TStringHash.Modify Method

Pascal

```
function Modify(const Key: string; Value: Integer): Boolean;
```

C++

```
__fastcall Boolean Modify(const UnicodeString Key, int Value);
```

Description

This is Modify, a member of class TStringHash.

36.1.6.4.6 TStringHash.Remove Method

Pascal

```
procedure Remove(const Key: string);
```

C++

```
__fastcall Remove(const UnicodeString Key);
```

Description

This is Remove, a member of class TStringHash.

36.1.6.4.7 TStringHash.ValueOf Method

Pascal

```
function ValueOf(const Key: string): Integer;
```

C++

```
__fastcall int ValueOf(const UnicodeString Key);
```

Description

This is ValueOf, a member of class TStringHash.

36.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	THashItem (see page 4328)	This is record IniFiles.THashItem.

36.2.1 IniFiles.THashItem Record

Pascal

```
THashItem = record
  Next: PHashItem;
  Key: string;
  Value: Integer;
end;
```

C++

```
struct THashItem {
  PHashItem Next;
  UnicodeString Key;
  int Value;
};
```

File

IniFiles

Description

This is record IniFiles.THashItem.

36.3 Types

The following table lists types in this documentation.

Types

Name	Description
PHashItem (see page 4329)	This is type IniFiles.PHashItem.
PPHashItem (see page 4329)	This is type IniFiles.PPHashItem.

36.3.1 IniFiles.PHashItem Type

Pascal

```
PHashItem = ^THashItem;
```

C++

```
^THashItem PHashItem;
```

File

IniFiles

Description

This is type IniFiles.PHashItem.

36.3.2 IniFiles.PPHashItem Type

Pascal

```
PPHashItem = ^PHashItem;
```

C++

```
^PHashItem PPHashItem;
```

File

IniFiles

Description

This is type IniFiles.PPHashItem.

37 IntfInfo Namespace

This is namespace IntfInfo.

Topics

Name	Description
Classes (see page 4333)	The following table lists classes in this documentation.
Functions (see page 4334)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4339)	The following table lists structs, records, enums in this documentation.
Types (see page 4342)	The following table lists types in this documentation.
Constants (see page 4345)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInterfaceRTTIException (see page 4333)	EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

Constants

Name	Description
CallingConventionName (see page 4345)	This is constant IntfInfo.CallingConventionName.
TypeInfoNames (see page 4345)	This is constant IntfInfo.TypeInfoNames.

Functions

	Name	Description
	GetClsMemberTypeInfo (see page 4334)	This is function IntfInfo.GetClsMemberTypeInfo.
	GetDynArrayElTypeInfo (see page 4334)	This is function IntfInfo.GetDynArrayElTypeInfo.
	GetDynArrayNextInfo (see page 4335)	This is function IntfInfo.GetDynArrayNextInfo.
	GetDynArrayNextInfo2 (see page 4335)	This is function IntfInfo.GetDynArrayNextInfo2.
	GetIntfMetaData (see page 4335)	This is function IntfInfo.GetIntfMetaData.
	GetIntfMetaData (see page 4336)	This is function IntfInfo.GetIntfMetaData.
	GetMethNum (see page 4336)	This is function IntfInfo.GetMethNum.
	GetPropListFlat (see page 4336)	This is function IntfInfo.GetPropListFlat.
	IsStoredPropConst (see page 4337)	This is function IntfInfo.IsStoredPropConst.
	OtherTypeName (see page 4337)	This is function IntfInfo.OtherTypeName.
	SameTypeInfo (see page 4337)	This is function IntfInfo.SameTypeInfo.
	TypeNamesMatch (see page 4338)	This is function IntfInfo.TypeNamesMatch.

Structs, Records, Enums

	Name	Description
	TCallConv (see page 4339)	This is record IntfInfo.TCallConv.
	TFillMethodArrayOpt (see page 4339)	This is record IntfInfo.TFillMethodArrayOpt.
	TIntfMetaData (see page 4340)	This is record IntfInfo.TIntfMetaData.
	TIntfMethEntry (see page 4340)	This is record IntfInfo.TIntfMethEntry.
	TIntfParamEntry (see page 4341)	This is record IntfInfo.TIntfParamEntry.

Types

Name	Description
PIntfMetaData (see page 4342)	This is type IntfInfo.PIntfMetaData.
PIntfMethEntry (see page 4342)	This is type IntfInfo.PIntfMethEntry.
PIntfParamEntry (see page 4343)	This is type IntfInfo.PIntfParamEntry.
TIntfMethEntryArray (see page 4343)	This is type IntfInfo.TIntfMethEntryArray.
TIntfParamEntryArray (see page 4343)	This is type IntfInfo.TIntfParamEntryArray.
TPIntfMethEntryArray (see page 4344)	This is type IntfInfo.TPIntfMethEntryArray.

37.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInterfaceRTTIException (see page 4333)	EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

37.1.1 EInterfaceRTTIException Class

EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

Class Hierarchy

Pascal

```
EInterfaceRTTIException = class(Exception);
```

C++

```
class EInterfaceRTTIException : public Exception;
```

File

IntfInfo

Description

EInterfaceRTTIException is thrown when an application expects an interface to be invokable but discovers that it does not include any runtime type information (RTTI). For example, the invocation registry raises an EInterfaceRTTIException exception when an application tries to register an interface that is not compiled with RTTI. Invoker components raise an EInterfaceRTTIException exception when an incoming request includes a call to an interface that does not include RTTI.

See Also

TInvokableClassRegistry (see page 4379)

TSoapPascalInvoker

IInvokable

37.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetClsMemberTypeInfo (see page 4334)	This is function IntfInfo.GetClsMemberTypeInfo.
	GetDynArrayElTypeInfo (see page 4334)	This is function IntfInfo.GetDynArrayElTypeInfo.
	GetDynArrayNextInfo (see page 4335)	This is function IntfInfo.GetDynArrayNextInfo.
	GetDynArrayNextInfo2 (see page 4335)	This is function IntfInfo.GetDynArrayNextInfo2.
	GetIntfMetaData (see page 4335)	This is function IntfInfo.GetIntfMetaData.
	GetIntfMetaData (see page 4336)	This is function IntfInfo.GetIntfMetaData.
	GetMethNum (see page 4336)	This is function IntfInfo.GetMethNum.
	GetPropListFlat (see page 4336)	This is function IntfInfo.GetPropListFlat.
	IsStoredPropConst (see page 4337)	This is function IntfInfo.IsStoredPropConst.
	OtherTypeName (see page 4337)	This is function IntfInfo.OtherTypeName.
	SameTypeInfo (see page 4337)	This is function IntfInfo.SameTypeInfo.
	TypeNamesMatch (see page 4338)	This is function IntfInfo.TypeNamesMatch.

37.2.1 IntfInfo.GetClsMemberTypeInfo Function

Pascal

```
function GetClsMemberTypeInfo(const ObjectTypeInfo: PTypeInfo; const MemberName: string = ''):
PTypeInfo;
```

C++

```
PTypeInfo GetClsMemberTypeInfo(const PTypeInfo ObjectTypeInfo, const UnicodeString MemberName
= '');
```

File

IntfInfo

Description

This is function IntfInfo.GetClsMemberTypeInfo.

37.2.2 IntfInfo.GetDynArrayElTypeInfo Function

Pascal

```
procedure GetDynArrayElTypeInfo(typeInfo: PTypeInfo; var EltInfo: PTypeInfo; var Dims:
Integer);
```

C++

```
GetDynArrayElTypeInfo(PTypeInfo typeInfo, PTypeInfo EltInfo, int Dims);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayElTypeInfo.

37.2.3 IntfInfo.GetDynArrayNextInfo Function

Pascal

```
function GetDynArrayNextInfo(typeInfo: PTypeInfo): PTypeInfo;
```

C++

```
PTypeInfo GetDynArrayNextInfo(PTypeInfo typeInfo);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayNextInfo.

37.2.4 IntfInfo.GetDynArrayNextInfo2 Function

Pascal

```
function GetDynArrayNextInfo2(typeInfo: PTypeInfo; var Name: string): PTypeInfo;
```

C++

```
PTypeInfo GetDynArrayNextInfo2(PTypeInfo typeInfo, UnicodeString Name);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayNextInfo2.

37.2.5 IntfInfo.GetIntfMetaData Function

Pascal

```
procedure GetIntfMetaData(Info: PTypeInfo; var IntfMD: TIntfMetaData; IncludeAllAncMethods:
Boolean = False); overload;
```

C++

```
GetIntfMetaData(PTypeInfo Info, TIntfMetaData IntfMD, Boolean IncludeAllAncMethods = False);
```

File

IntfInfo

Description

This is function IntfInfo.GetIntfMetaData.

37.2.6 IntfInfo.GetIntfMetaData Function

Pascal

```
procedure GetIntfMetaData(Info: PTypeInfo; var IntfMD: TIntfMetaData; MethodArrayOpt:
TFillMethodArrayOpt); overload;
```

C++

```
GetIntfMetaData(PTypeInfo Info, TIntfMetaData IntfMD, TFillMethodArrayOpt MethodArrayOpt);
```

File

IntfInfo

Description

This is function IntfInfo.GetIntfMetaData.

37.2.7 IntfInfo.GetMethNum Function

Pascal

```
function GetMethNum(const IntfMD: TIntfMetaData; const MethName: string; ParamCount: Integer =
-1): Integer;
```

C++

```
int GetMethNum(const TIntfMetaData IntfMD, const UnicodeString MethName, int ParamCount = -1);
```

File

IntfInfo

Description

This is function IntfInfo.GetMethNum.

37.2.8 IntfInfo.GetPropListFlat Function

Pascal

```
function GetPropListFlat(TypeInfo: PTypeInfo; out PropList: PPropList): Integer;
```

C++

```
int GetPropListFlat(PTypeInfo TypeInfo, PPropList PropList);
```

File

IntfInfo

Description

This is function IntfInfo.GetPropListFlat.

37.2.9 IntfInfo.IsStoredPropConst Function

Pascal

```
function IsStoredPropConst(Instance: TObject; PropInfo: PPropInfo): Boolean;
```

C++

```
Boolean IsStoredPropConst(TObject * Instance, PPropInfo PropInfo);
```

File

IntfInfo

Description

This is function IntfInfo.IsStoredPropConst.

37.2.10 IntfInfo.OtherTypeName Function

Pascal

```
function OtherTypeName(const TypeName: string): string;
```

C++

```
UnicodeString OtherTypeName(const UnicodeString TypeName);
```

File

IntfInfo

Description

This is function IntfInfo.OtherTypeName.

37.2.11 IntfInfo.SameTypeInfo Function

Pascal

```
function SameTypeInfo(const RegInfo: PTypeInfo; const OtherInfo: PTypeInfo): boolean;
```

C++

```
boolean SameTypeInfo(const PTypeInfo RegInfo, const PTypeInfo OtherInfo);
```

File

IntfInfo

Description

This is function IntfInfo.SameTypeInfo.

37.2.12 IntfInfo.TypeNamesMatch Function

Pascal

```
function TypeNamesMatch(const RegName: string; const OtherName: string): boolean;
```

C++

```
boolean TypeNamesMatch(const UnicodeString RegName, const UnicodeString OtherName);
```

File

IntfInfo

Description

This is function IntfInfo.TypeNamesMatch.

37.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCallConv (see page 4339)	This is record IntfInfo.TCallConv.
	TFillMethodArrayOpt (see page 4339)	This is record IntfInfo.TFillMethodArrayOpt.

Records

	Name	Description
	TIntfMetaData (see page 4340)	This is record IntfInfo.TIntfMetaData.
	TIntfMethEntry (see page 4340)	This is record IntfInfo.TIntfMethEntry.
	TIntfParamEntry (see page 4341)	This is record IntfInfo.TIntfParamEntry.

37.3.1 IntfInfo.TCallConv Enumeration

Pascal

```
TCallConv = (
  ccReg,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
);
```

C++

```
enum TCallConv {
  ccReg,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
};
```

File

IntfInfo

Description

This is record IntfInfo.TCallConv.

37.3.2 IntfInfo.TFillMethodArrayOpt Enumeration

Pascal

```
TFillMethodArrayOpt = (
  fmoAllBaseMethods,
  fmoRTTIBaseMethods,
```

```
    fmoNoBaseMethods
  );
```

C++

```
  enum TFillMethodArrayOpt {
    fmoAllBaseMethods,
    fmoRTTIBaseMethods,
    fmoNoBaseMethods
  };
```

File

IntfInfo

Description

This is record IntfInfo.TFillMethodArrayOpt.

37.3.3 IntfInfo.TIntfMetaData Record

Pascal

```
  TIntfMetaData = record
    Name: string;
    UnitName: string;
    MDA: TIntfMethEntryArray;
    IID: TGUID;
    Info: PTypeInfo;
    AncInfo: PTypeInfo;
    NumAnc: Integer;
  end;
```

C++

```
  struct TIntfMetaData {
    UnicodeString Name;
    UnicodeString UnitName;
    TIntfMethEntryArray MDA;
    TGUID IID;
    PTypeInfo Info;
    PTypeInfo AncInfo;
    int NumAnc;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfMetaData.

37.3.4 IntfInfo.TIntfMethEntry Record

Pascal

```
  TIntfMethEntry = record
    Name: string;
    CC: TCallConv;
    Pos: Integer;
    ParamCount: Integer;
```

```
    ResultInfo: PTypeInfo;
    SelfInfo: PTypeInfo;
    Params: TIntfParamEntryArray;
    HasRTTI: Boolean;
  end;
```

C++

```
  struct TIntfMethEntry {
    UnicodeString Name;
    TCallConv CC;
    int Pos;
    int ParamCount;
    PTypeInfo ResultInfo;
    PTypeInfo SelfInfo;
    TIntfParamEntryArray Params;
    Boolean HasRTTI;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfMethEntry.

37.3.5 IntfInfo.TIntfParamEntry Record

Pascal

```
  TIntfParamEntry = record
    Flags: TParamFlags;
    Name: string;
    Info: PTypeInfo;
  end;
```

C++

```
  struct TIntfParamEntry {
    TParamFlags Flags;
    UnicodeString Name;
    PTypeInfo Info;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfParamEntry.

37.4 Types

The following table lists types in this documentation.

Types

Name	Description
PIntfMetaData (see page 4342)	This is type IntfInfo.PIntfMetaData.
PIntfMethEntry (see page 4342)	This is type IntfInfo.PIntfMethEntry.
PIntfParamEntry (see page 4343)	This is type IntfInfo.PIntfParamEntry.
TIntfMethEntryArray (see page 4343)	This is type IntfInfo.TIntfMethEntryArray.
TIntfParamEntryArray (see page 4343)	This is type IntfInfo.TIntfParamEntryArray.
TPIntfMethEntryArray (see page 4344)	This is type IntfInfo.TPIntfMethEntryArray.

37.4.1 IntfInfo.PIntfMetaData Type

Pascal

```
PIntfMetaData = ^TIntfMetaData;
```

C++

```
^TIntfMetaData PIntfMetaData;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfMetaData.

37.4.2 IntfInfo.PIntfMethEntry Type

Pascal

```
PIntfMethEntry = ^TIntfMethEntry;
```

C++

```
^TIntfMethEntry PIntfMethEntry;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfMethEntry.

37.4.3 IntfInfo.PIntfParamEntry Type

Pascal

```
PIntfParamEntry = ^TIntfParamEntry;
```

C++

```
^TIntfParamEntry PIntfParamEntry;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfParamEntry.

37.4.4 IntfInfo.TIntfMethEntryArray Type

Pascal

```
TIntfMethEntryArray = array of TIntfMethEntry;
```

C++

```
array of TIntfMethEntry TIntfMethEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TIntfMethEntryArray.

37.4.5 IntfInfo.TIntfParamEntryArray Type

Pascal

```
TIntfParamEntryArray = array of TIntfParamEntry;
```

C++

```
array of TIntfParamEntry TIntfParamEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TIntfParamEntryArray.

37.4.6 IntfInfo.TPIntfMethEntryArray Type

Pascal

```
TPIntfMethEntryArray = array of PIntfMethEntry;
```

C++

```
array of PIntfMethEntry TPIntfMethEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TPIntfMethEntryArray.

37.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CallingConventionName (see page 4345)	This is constant IntfInfo.CallingConventionName.
TypeInfoNames (see page 4345)	This is constant IntfInfo.TypeInfoNames.

37.5.1 IntfInfo.CallingConventionName Constant

Pascal

```
CallingConventionName: array[ccReg..ccSafeCall] of string = ('REGISTER', 'CDECL', 'PASCAL',
'STDCALL', 'SAFECALL');
```

C++

```
array[ccReg..ccSafeCall] of string CallingConventionName = ('REGISTER', 'CDECL', 'PASCAL',
'STDCALL', 'SAFECALL');
```

File

IntfInfo

Description

This is constant IntfInfo.CallingConventionName.

37.5.2 IntfInfo.TypeInfoNames Constant

Pascal

```
TypeInfoNames: array[0..37] of string = ( 'Boolean', 'bool', 'AnsiChar', 'char', 'AnsiChar',
'signed char', 'Char', 'wchar_t', 'Byte', 'unsigned char', 'SmallInt', 'short', 'Word',
'unsigned short', 'Integer', 'int', 'Cardinal', 'unsigned int', 'Integer', 'long', 'Cardinal',
'unsigned long', 'Int64', '__int64', 'Int64', 'unsigned __int64', 'Single', 'float', 'Double',
'double', 'Extended', 'long double', 'string', 'UnicodeString', 'AnsiString', 'AnsiString',
'WideString', 'WideString');
```

C++

```
array[0..37] of string TypeInfoNames = ( 'Boolean', 'bool', 'AnsiChar', 'char', 'AnsiChar',
'signed char', 'Char', 'wchar_t', 'Byte', 'unsigned char', 'SmallInt', 'short', 'Word',
'unsigned short', 'Integer', 'int', 'Cardinal', 'unsigned int', 'Integer', 'long', 'Cardinal',
'unsigned long', 'Int64', '__int64', 'Int64', 'unsigned __int64', 'Single', 'float', 'Double',
'double', 'Extended', 'long double', 'string', 'UnicodeString', 'AnsiString', 'AnsiString',
'WideString', 'WideString');
```

File

IntfInfo

Description

This is constant IntfInfo.TypeInfoNames.

38 InvokeRegistry Namespace

This is namespace InvokeRegistry.

Topics

Name	Description
Classes (see page 4350)	The following table lists classes in this documentation.
Interfaces (see page 4460)	The following table lists interfaces in this documentation.
Functions (see page 4470)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4475)	The following table lists structs, records, enums in this documentation.
Types (see page 4487)	The following table lists types in this documentation.
Variables (see page 4496)	The following table lists variables in this documentation.
Constants (see page 4497)	The following table lists constants in this documentation.

Classes

	Name	Description
	ERemotableException (see page 4350)	ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.
	ETypeRegistryException (see page 4353)	ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.
	TDataContext (see page 4354)	TDataContext stores temporary information about a set of objects, Variants, and other related values.
	THeaderList (see page 4363)	THeaderList maintains a list of SOAP headers.
	TInvContext (see page 4369)	TInvContext stores information about a method call on an invokeable interface.
	TInvokableClass (see page 4373)	TInvokableClass is the base class for the implementation classes of invokable interfaces.
	TInvokableClassRegistry (see page 4379)	TInvokableClassRegistry is the type of the invocation registry.
	TRemotable (see page 4408)	TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.
	TRemotableTypeRegistry (see page 4414)	TRemotableTypeRegistry is the type of the remotable type registry.
	TRemotableXS (see page 4432)	TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.
	TSOAPAttachment (see page 4435)	TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.
	TSOAPHeader (see page 4448)	TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.
	TSOAPHeaders (see page 4452)	TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 4464) interface.
	TSOAPHeadersBase (see page 4457)	TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

Constants

Name	Description
AS_ATTRIBUTE (see page 4497)	Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.
XMLBase64Types (see page 4497)	This is constant InvokeRegistry.XMLBase64Types.
XMLSchemaInstNamepspaces (see page 4498)	This is constant InvokeRegistry.XMLSchemaInstNamepspaces.
XMLSchemaNamepspaces (see page 4498)	This is constant InvokeRegistry.XMLSchemaNamepspaces.

Functions

	Name	Description
	GetRemotableDataContext (see page 4470)	Returns the TDataContext (see page 4354) object that provides temporary storage during a call to an invokable interface.
	InvRegistry (see page 4471)	Provides access to the invocation registry.
	RemClassRegistry (see page 4471)	Provides access to the remotable class registry.
	RemTypeRegistry (see page 4472)	Provides access to the remotable type registry.
	SetRemotableDataContext (see page 4473)	Sets the data context that provides temporary storage during a call to an invokable interface.
	SubstituteStrings (see page 4474)	This is function InvokeRegistry.SubstituteStrings.

Interfaces

	Name	Description
	IHeadersSetter (see page 4460)	This is class InvokeRegistry.IHeadersSetter.
	IObjConverter (see page 4461)	IObjConverter is the interface for converting between a remotable class and its SOAP representation.
	ISOAPHeaders (see page 4464)	ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

Structs, Records, Enums

	Name	Description
	eHeaderMethodType (see page 4475)	eHeaderMethodType and THeaderMethodTypeArray (see page 4490) indicate the types of messages that should include a given header type.
	ExtNameMapItem (see page 4476)	This is record InvokeRegistry.ExtNameMapItem.
	IntfExceptionItem (see page 4476)	IntfExceptionItem and TExceptionItemArray (see page 4489) describe the association between a remotable exception and an interface that can raise it.
	IntfHeaderItem (see page 4477)	IntfHeaderItem and THeaderItemArray (see page 4489) describe the association between a header and the SOAP messages that include it.
	InvRegClassEntry (see page 4478)	This is record InvokeRegistry.InvRegClassEntry.
	InvRegIntfEntry (see page 4478)	InvRegIntfEntry stores information about an invokable interface.
	MethParamNameMapItem (see page 4480)	This is record InvokeRegistry.MethParamNameMapItem.
	ObjectConvertOptions (see page 4480)	ObjectConvertOptions and TObjectConvertOptions (see page 4491) are flags that customize how to convert a remotable object to or from its SOAP representation.
	SerializationOptions (see page 4481)	SerializationOptions and TSerializationOptions (see page 4494) are flags that customize how to convert a remotable object to or from its SOAP representation.
	TDynToClear (see page 4482)	This is record InvokeRegistry.TDynToClear.
	TIntfInvokeOption (see page 4483)	TIntfInvokeOption and TIntfInvokeOptions (see page 4490) represent flags that customize how to handle calls to an invokable interface.
	TObjMultiOptions (see page 4484)	TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.
	TRemHeaderEntry (see page 4485)	This is record InvokeRegistry.TRemHeaderEntry.
	TRemRegEntry (see page 4485)	TRemRegEntry stores information for the remotable type registry.

Types

Name	Description
ERemotableExceptionClass (see page 4487)	This is type InvokeRegistry.ERemotableExceptionClass.
InvString (see page 4488)	This is type InvokeRegistry.InvString.
InvStringDynArray (see page 4488)	This is type InvokeRegistry.InvStringDynArray.
PTRemotable (see page 4488)	This is type InvokeRegistry.PTRemotable.

TCreateInstanceProc (see page 4489)	TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.
TExceptionItemArray (see page 4489)	This is type InvokeRegistry.TExceptionItemArray.
THeaderItemArray (see page 4489)	This is type InvokeRegistry.THeaderItemArray.
THeaderMethodTypeArray (see page 4490)	This is type InvokeRegistry.THeaderMethodTypeArray.
TIntfInvokeOptions (see page 4490)	This is type InvokeRegistry.TIntfInvokeOptions.
TInvokableClassClass (see page 4490)	This is type InvokeRegistry.TInvokableClassClass.
TInvRegIntfEntryArray (see page 4491)	This is type InvokeRegistry.TInvRegIntfEntryArray.
TObjectConvertOptions (see page 4491)	This is type InvokeRegistry.TObjectConvertOptions.
TPascalRemotableClassRegistry (see page 4491)	TPascalRemotableClassRegistry is the type of the remotable class registry.
TPascalRemotableTypeRegistry (see page 4492)	TPascalRemotableTypeRegistry is the type of the remotable type registry.
TRemotableClass (see page 4492)	This is type InvokeRegistry.TRemotableClass.
TRemotableClassRegistry (see page 4492)	This is type InvokeRegistry.TRemotableClassRegistry.
TRemotableXSClass (see page 4493)	This is type InvokeRegistry.TRemotableXSClass.
TRemRegEntryArray (see page 4493)	This is type InvokeRegistry.TRemRegEntryArray.
TRequiredArray (see page 4493)	TRequiredArray is a dynamic array of boolean values.
TSerializationOptions (see page 4494)	This is type InvokeRegistry.TSerializationOptions.
TSOAPAttachmentClass (see page 4494)	This is type InvokeRegistry.TSOAPAttachmentClass.
TSOAPHeaderClass (see page 4494)	This is type InvokeRegistry.TSOAPHeaderClass.

Variables

Name	Description
AppNameSpacePrefix (see page 4496)	Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

38.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERemotableException (see page 4350)	ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.
	ETypeRegistryException (see page 4353)	ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.
	TDataContext (see page 4354)	TDataContext stores temporary information about a set of objects, Variants, and other related values.
	THeaderList (see page 4363)	THeaderList maintains a list of SOAP headers.
	TInvContext (see page 4369)	TInvContext stores information about a method call on an invokeable interface.
	TInvokableClass (see page 4373)	TInvokableClass is the base class for the implementation classes of invokable interfaces.
	TInvokableClassRegistry (see page 4379)	TInvokableClassRegistry is the type of the invocation registry.
	TRemotable (see page 4408)	TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.
	TRemotableTypeRegistry (see page 4414)	TRemotableTypeRegistry is the type of the remotable type registry.
	TRemotableXS (see page 4432)	TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.
	TSOAPAttachment (see page 4435)	TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.
	TSOAPHeader (see page 4448)	TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.
	TSOAPHeaders (see page 4452)	TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 4464) interface.
	TSOAPHeadersBase (see page 4457)	TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

38.1.1 ERemotableException Class

ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.

Class Hierarchy

Pascal

```
ERemotableException = class(Exception);
```

C++

```
class ERemotableException : public Exception;
```

File

InvokeRegistry

Description

ERemotableException is the exception class for exceptions that a client raises based on a SOAP fault packet that it receives from a Web Service application. The ERemotableException properties represent standard elements of the SOAP fault packet.

Create (see page 4351) an ERemotableException descendant to define a custom exception class for problems encountered when executing the methods of an invokable interface in a Web Service application. When a Web Service application raises an ERemotableException descendant in the course of executing a request from a client application, it adds the values of any published properties to the fault packet, in addition to the value of the Message property. Note that these published properties must be remotable types: that is, the value must be a scalar type, or it must be a registered descendant of TRemotable (see page 4408).

Before the properties of an ERemotableException descendant can be transmitted in a fault packet, it must be registered with the remotable type registry (RemTypeRegistry (see page 4472)). Call the RegisterXSClass method of the remotable type registry, on both the client and the server application, to enable your exception class to be transmitted.

See Also

RemTypeRegistry (see page 4472)

TRemotableTypeRegistry (see page 4414)

Creating Custom Exception Classes for Web Services

38.1.1.1 ERemotableException Members

The following tables list the members exposed by ERemotableException.

Public Methods

	Name	Description
	Create (see page 4351)	This is the overview for the Create constructor overload.

Public Properties

	Name	Description
	FaultActor (see page 4352)	Indicates the agent that raised the fault.
	FaultCode (see page 4352)	Indicates where the fault was encountered.
	FaultDetail (see page 4353)	Provides application-specific details about the exception that occurred.

38.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	ERemotableException.Create () (see page 4351)	Creates and initializes an instance of ERemotableException (see page 4350).
	ERemotableException.Create (string, InvString, InvString) (see page 4351)	Creates and initializes an instance of ERemotableException (see page 4350).

38.1.1.2.1 ERemotableException.Create Constructor ()

Creates and initializes an instance of ERemotableException (see page 4350).

Pascal

```
constructor Create; virtual; overload;
constructor Create(const Msg: string; const AFaultCode: InvString = ''; const AFaultActor:
InvString = ''); virtual; overload;
```

C++

```
virtual __fastcall ERemotableException();
virtual __fastcall ERemotableException(const UnicodeString Msg, const InvString AFaultCode =
'', const InvString AFaultActor = '');
```

Description

Create generates an ERemotableException (see page 4350) object.

Msg is the error message associated with the exception. It becomes the value of the Message property.

AFaultCode indicates where the fault that gave rise to the exception was encountered. It becomes the value of the FaultCode (see page 4352) property.

AFaultActor indicates the agent that gave rise to the fault. It becomes the value of the FaultActor (see page 4352) property.

See Also

Message

FaultCode (see page 4352)

FaultActor (see page 4352)

38.1.1.3 ERemotableException Properties

The properties of the ERemotableException class are listed here.

Public Properties

	Name	Description
	FaultActor (see page 4352)	Indicates the agent that raised the fault.
	FaultCode (see page 4352)	Indicates where the fault was encountered.
	FaultDetail (see page 4353)	Provides application-specific details about the exception that occurred.

38.1.1.3.1 ERemotableException.FaultActor Property

Indicates the agent that raised the fault.

Pascal

```
property FaultActor: InvString;
```

C++

```
__property InvString FaultActor;
```

Description

FaultActor is the URI of the agent that caused the fault. This is used when the SOAP request passes through multiple agents when processed. The FaultActor can be a proxy server, the Web Service application to which the request was addressed, or another agent that the target Web Service application calls on when processing the request.

This property is only set if the Web Service provider specifies a FaultActor in its SOAP fault packet.

See Also

FaultCode (see page 4352)

38.1.1.3.2 ERemotableException.FaultCode Property

Indicates where the fault was encountered.

Pascal

```
property FaultCode: InvString;
```

C++

```
__property InvString FaultCode;
```

Description

FaultCode is one of the following strings:

Value	Meaning
Server	The Web Service application raised the exception before it was able to begin processing the content of the SOAP request packet. For example, if the server can't process a request because it is out of memory, it specifies a FaultCode of 'Server'.
Client	The Web Service application raised the exception because of an error in the SOAP request packet that the client provided. For example, if the SOAP request packet specified a nonexistent method or invalid parameter, the server specifies a FaultCode of 'Client'.
VersionMismatch	The Web Service application is a different version than the one that the SOAP request packet specifies.
MustUnderstand	The Web Service application did not understand one of the SOAP headers in the request packet and the request packet set its MustUnderstand header to indicate that the server must handle all of its SOAP headers.

See Also

FaultActor (see page 4352)

38.1.1.3.3 ERemotableException.FaultDetail Property

Provides application-specific details about the exception that occurred.

Pascal

```
property FaultDetail: InvString;
```

C++

```
__property InvString FaultDetail;
```

Description

FaultDetail is a fragment of XML that the server generates to provide detailed information about the exception that occurred. FaultDetail provides context information in addition to the fault message that becomes the value of the Message property.

Each server defines the structure and meaning of FaultDetail in an application-specific way. When an ERemotableException (see page 4350) descendant includes published properties, their values are added to FaultDetail.

See Also

Message

38.1.2 ETypeRegistryException Class

ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.

Class Hierarchy

Pascal

```
ETypeRegistryException = class(Exception);
```

C++

```
class ETypeRegistryException : public Exception;
```

File

InvokeRegistry

Description

ETypeRegistryException is thrown when

An application tries to register an implementation class with the invocation registry and that class does not implement any interfaces.

An application tries to look up a type in the remotable type registry that is not remotable.

See Also

TInvokableClassRegistry (see page 4379)

TRemotableTypeRegistry (see page 4414)

38.1.3 TDataContext Class

TDataContext stores temporary information about a set of objects, Variants, and other related values.

Class Hierarchy

Pascal

```
TDataContext = class;
```

C++

```
class TDataContext;
```

File

InvokeRegistry

Description

TDataContext is a repository for information that belongs together. It introduces a number of methods for storing different types of information, including objects, Variants, pointers, and raw data (untyped bytes). When the TDataContext object is freed, it performs all necessary cleanup of the information it stores, freeing the objects, clearing the Variants, and so on.

Do not use TDataContext directly. It is intended as a base class for classes that need a temporary repository that serves a specific purpose. TDataContext does not allocate the internal dynamic arrays that store pointers and raw data: This task is left to descendant classes, such as TInvContext (see page 4369).

See Also

TInvContext (see page 4369)

38.1.3.1 TDataContext Members

The following tables list the members exposed by TDataContext.

Protected Fields

	Name	Description
	Data (see page 4356)	This is Data, a member of class TDataContext.
	DataOffset (see page 4356)	This is DataOffset, a member of class TDataContext.
	DataP (see page 4357)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 4357)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 4357)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 4357)	This is StrToClear, a member of class TDataContext.
	UStrToClear (see page 4357)	This is UStrToClear, a member of class TDataContext.
	VarToClear (see page 4358)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 4358)	This is WStrToClear, a member of class TDataContext.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 4358)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 4359)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 4359)	Stores a string that can be cleared when the data context is finished.
	AddUStrToClear (see page 4360)	This is AddUStrToClear, a member of class TDataContext.
	AddVariantToClear (see page 4360)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 4361)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 4361)	Returns a pointer to the data context's cache of raw memory.
	Create (see page 4355)	Creates an instance of TDataContext (see page 4354).
	Destroy (see page 4355)	Destroys an instance of TDataContext (see page 4354).
	GetDataPointer (see page 4362)	Returns a pointer from the internal cache of pointers.
	RemoveObjectToDestroy (see page 4362)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 4362)	Stores a pointer in the internal cache of pointers.

38.1.3.2 TDataContext.Create Constructor

Creates an instance of TDataContext (see page 4354).

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataContext();
```

Description

Call Create to instantiate a TDataContext (see page 4354) object.

38.1.3.3 TDataContext.Destroy Destructor

Destroys an instance of TDataContext (see page 4354).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataContext();
```

Description

Destroy frees all the temporary data stored in the TDataContext (see page 4354) instance.

See Also

delete

38.1.3.4 TDataContext Fields

The fields of the TDataContext class are listed here.

Protected Fields

	Name	Description
	Data (see page 4356)	This is Data, a member of class TDataContext.
	DataOffset (see page 4356)	This is DataOffset, a member of class TDataContext.
	DataP (see page 4357)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 4357)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 4357)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 4357)	This is StrToClear, a member of class TDataContext.
	UStrToClear (see page 4357)	This is UStrToClear, a member of class TDataContext.
	VarToClear (see page 4358)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 4358)	This is WStrToClear, a member of class TDataContext.

38.1.3.4.1 TDataContext.Data Field

Pascal

```
Data: array of Byte;
```

C++

```
array of Byte Data;
```

Description

This is Data, a member of class TDataContext.

38.1.3.4.2 TDataContext.DataOffset Field

Pascal

```
DataOffset: Integer;
```

C++

```
int DataOffset;
```

Description

This is DataOffset, a member of class TDataContext.

38.1.3.4.3 TDataContext.DataP Field

Pascal

```
DataP: array of Pointer;
```

C++

```
array of Pointer DataP;
```

Description

This is DataP, a member of class TDataContext.

38.1.3.4.4 TDataContext.DynArrayToClear Field

Pascal

```
DynArrayToClear: array of TDynToClear;
```

C++

```
array of TDynToClear DynArrayToClear;
```

Description

This is DynArrayToClear, a member of class TDataContext.

38.1.3.4.5 TDataContext.FObjsToDestroy Field

Pascal

```
FObjsToDestroy: array of TObject;
```

C++

```
array of TObject FObjsToDestroy;
```

Description

This is FObjsToDestroy, a member of class TDataContext.

38.1.3.4.6 TDataContext.StrToClear Field

Pascal

```
StrToClear: array of Pointer;
```

C++

```
array of Pointer StrToClear;
```

Description

This is StrToClear, a member of class TDataContext.

38.1.3.4.7 TDataContext.UStrToClear Field

Pascal

```
UStrToClear: array of Pointer;
```

C++

```
array of Pointer UStrToClear;
```

Description

This is UStrToClear, a member of class TDataContext.

38.1.3.4.8 TDataContext.VarToClear Field

Pascal

```
VarToClear: array of Pointer;
```

C++

```
array of Pointer VarToClear;
```

Description

This is VarToClear, a member of class TDataContext.

38.1.3.4.9 TDataContext.WStrToClear Field

Pascal

```
WStrToClear: array of Pointer;
```

C++

```
array of Pointer WStrToClear;
```

Description

This is WStrToClear, a member of class TDataContext.

38.1.3.5 TDataContext Methods

The methods of the TDataContext class are listed here.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 4358)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 4359)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 4359)	Stores a string that can be cleared when the data context is finished.
	AddUStrToClear (see page 4360)	This is AddUStrToClear, a member of class TDataContext.
	AddVariantToClear (see page 4360)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 4361)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 4361)	Returns a pointer to the data context's cache of raw memory.
	GetDataPointer (see page 4362)	Returns a pointer from the internal cache of pointers.
	RemoveObjectToDestroy (see page 4362)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 4362)	Stores a pointer in the internal cache of pointers.

38.1.3.5.1 TDataContext.AddDynArrayToClear Method

Stores a dynamic array that can be cleared when the data context is finished.

Pascal

```
procedure AddDynArrayToClear(P: Pointer; Info: PTypeInfo);
```

C++

```
__fastcall AddDynArrayToClear(void * P, PTypeInfo Info);
```

Description

Call AddDynArrayToClear to add a dynamic array to the list of temporary dynamic arrays that TDataContext (see page 4354) maintains. When the TDataContext (see page 4354) instance is freed, it clears all dynamic arrays in this list of temporary arrays.

P is a pointer to the dynamic array.

Info is the TypeInfo pointer for the dynamic array.

Note: TDataContext (see page 4354) does not provide access to this list. The caller must keep a reference to the dynamic arrays for use while the TDataContext (see page 4354) instance exists.

See Also

Destroy (see page 4355)

AddObjectToDestroy (see page 4359)

AddVariantToClear (see page 4360)

AddStrToClear (see page 4359)

AddWStrToClear (see page 4361)

38.1.3.5.2 TDataContext.AddObjectToDestroy Method

Stores an object that can be freed when the data context is finished.

Pascal

```
procedure AddObjectToDestroy(Obj: TObject);
```

C++

```
__fastcall AddObjectToDestroy(TObject * Obj);
```

Description

Call AddObjectToDestroy to add an object instance to the list of temporary objects that TDataContext (see page 4354) maintains. When the TDataContext (see page 4354) instance is freed, it frees all the objects in this list of temporary objects.

Note: TDataContext (see page 4354) does not provide access to this list. The caller must keep a reference to the object for use while the TDataContext (see page 4354) instance exists.

See Also

Destroy (see page 4355)

AddDynArrayToClear (see page 4358)

AddVariantToClear (see page 4360)

RemoveObjectToDestroy (see page 4362)

AddStrToClear (see page 4359)

38.1.3.5.3 TDataContext.AddStrToClear Method

Stores a string that can be cleared when the data context is finished.

Pascal

```
procedure AddStrToClear(P: Pointer);
```

C++

```
__fastcall AddStrToClear(void * P);
```

Description

Call AddStrToClear to add a string to the list of temporary strings that TDataContext (see page 4354) maintains. When the TDataContext (see page 4354) instance is freed, it clears all strings in this list.

P is a pointer to the string.

Note: TDataContext (see page 4354) does not provide access to this list. The caller must keep a reference to the string for use while the TDataContext (see page 4354) instance exists.

See Also

Destroy (see page 4355)

AddObjectToDestroy (see page 4359)

AddVariantToClear (see page 4360)

AddDynArrayToClear (see page 4358)

AddWStrToClear (see page 4361)

38.1.3.5.4 TDataContext.AddUStrToClear Method

Pascal

```
procedure AddUStrToClear(P: Pointer);
```

C++

```
__fastcall AddUStrToClear(void * P);
```

Description

This is AddUStrToClear, a member of class TDataContext.

38.1.3.5.5 TDataContext.AddVariantToClear Method

Stores the value of a Variant that can be cleared when the data context is finished.

Pascal

```
procedure AddVariantToClear(P: PVarData);
```

C++

```
__fastcall AddVariantToClear(PVarData P);
```

Description

Call AddVariantToClear to add a Variant to the list of temporary Variant values that TDataContext (see page 4354) maintains. When the TDataContext (see page 4354) instance is freed, it clears all the Variants in this list of temporary Variant values.

Note: TDataContext (see page 4354) does not provide access to this list. The caller must keep a reference to the Variant for use while the TDataContext (see page 4354) instance exists.

See Also

Destroy (see page 4355)

AddStrToClear (see page 4359)

AddWStrToClear (see page 4361)

38.1.3.5.6 TDataContext.AddWStrToClear Method

Stores a wide string that can be cleared when the data context is finished.

Pascal

```
procedure AddWStrToClear(P: Pointer);
```

C++

```
__fastcall AddWStrToClear(void * P);
```

Description

Call AddWStrToClear to add a wide string to the list of temporary wide strings that TDataContext (see page 4354) maintains. When the TDataContext (see page 4354) instance is freed, it clears all strings in this list.

P is a pointer to the WideString.

Note: TDataContext (see page 4354) does not provide access to this list. The caller must keep a reference to the string for use while the TDataContext (see page 4354) instance exists.

See Also

Destroy (see page 4355)

AddObjectToDestroy (see page 4359)

AddVariantToClear (see page 4360)

AddDynArrayToClear (see page 4358)

AddStrToClear (see page 4359)

38.1.3.5.7 TDataContext.AllocData Method

Returns a pointer to the data context's cache of raw memory.

Pascal

```
function AllocData(Size: Integer): Pointer;
```

C++

```
__fastcall void * AllocData(int Size);
```

Description

Use AllocData to obtain a pointer to a chunk of temporary memory.

Size is the number of bytes desired.

AllocData returns a pointer to Size bytes.

Note: TDataContext (see page 4354) does not actually allocate the memory that AllocData returns. Descendant classes must initialize the internal cache that the returned pointer addresses.

See Also

GetDataPointer (see page 4362)

38.1.3.5.8 TDataContext.GetDataPointer Method

Returns a pointer from the internal cache of pointers.

Pascal

```
function GetDataPointer(Index: Integer): Pointer;
```

C++

```
__fastcall void * GetDataPointer(int Index);
```

Description

GetDataPointer returns a pointer from the internal cache of pointers. This value is assigned by a call to the SetDataPointer (see page 4362) method.

Index is the index of the desired pointer, where 0 is the first pointer, 1 is the second pointer, and so on.

Note: TDataContext (see page 4354) does not actually allocate the memory to store these pointers. Descendant classes must initialize the internal dynamic array that TDataContext (see page 4354) uses to store pointers.

See Also

SetDataPointer (see page 4362)

38.1.3.5.9 TDataContext.RemoveObjectToDestroy Method

Remove an object from the list of objects to free when the data context is freed.

Pascal

```
procedure RemoveObjectToDestroy(Obj: TObject);
```

C++

```
__fastcall RemoveObjectToDestroy(TObject * Obj);
```

Description

Call RemoveObjectToDestroy to remove an object instance from the list of temporary objects that TDataContext (see page 4354) maintains. This prevents TDataContext (see page 4354) from freeing the object when it is itself freed.

For example, TRemotable (see page 4408) descendants that are created by a Web Service application when it executes an invokable interface call are automatically added to a data context that persists for the length of the method call. When an application changes the remotable object's DataContext property, it automatically calls RemoveObjectToDestroy so that the object is not automatically freed when the method call is complete.

See Also

DataContext (see page 4413)

AddObjectToDestroy (see page 4359)

38.1.3.5.10 TDataContext.SetDataPointer Method

Stores a pointer in the internal cache of pointers.

Pascal

```
procedure SetDataPointer(Index: Integer; P: Pointer);
```

C++

```
__fastcall SetDataPointer(int Index, void * P);
```

Description

SetDataPointer stores a pointer that can later be retrieved by a call to GetDataPointer (see page 4362). TDataContext (see page 4354) owns only the pointer, not the data to which it points. When you free the TDataContext (see page 4354) instance, it frees its list of pointers, but not the memory to which they point, unless they are first obtained by a call to the AllocData (see page 4361) method.

Index is the index of the pointer in the internal cache of pointers, where 0 specifies the first pointer, 1 specifies the second pointer, and so on.

P is the pointer to store.

Note: TDataContext (see page 4354) does not actually allocate the memory to store these pointers. Descendant classes must initialize the internal dynamic array that TDataContext (see page 4354) uses to store pointers.

See Also

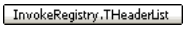
GetDataPointer (see page 4362)

AllocData (see page 4361)

38.1.4 THeaderList Class

THeaderList maintains a list of SOAP headers.

Class Hierarchy



Pascal

```
THeaderList = class;
```

C++

```
class THeaderList;
```

File

InvokeRegistry

Description

Use THeaderList to store and maintain a list of SOAP headers (TSOAPHeader (see page 4448) descendants).

See Also

TSoapHeader

38.1.4.1 THeaderList Members

The following tables list the members exposed by THeaderList.

Protected Methods

	Name	Description
	GetCount (see page 4366)	This is GetCount, a member of class THeaderList.
	GetHeader (see page 4367)	This is GetHeader, a member of class THeaderList.
	GetOwnsObjects (see page 4367)	This is GetOwnsObjects, a member of class THeaderList.
	SetOwnsObjects (see page 4367)	This is SetOwnsObjects, a member of class THeaderList.

Public Methods

	Name	Description
	Add (see page 4365)	This is the overview for the Add method overload.
	Clear (see page 4366)	Empties the list and frees the TSOAPHeader (see page 4448) objects.
	Create (see page 4364)	Creates and initializes a THeaderList (see page 4363) instance.
V	Destroy (see page 4364)	Destroys an instance of THeaderList (see page 4363).
	Extract (see page 4366)	Removes a specified header from the list without freeing it.
	IndexOf (see page 4367)	This is IndexOf, a member of class THeaderList.

Public Properties

	Name	Description
R	Count (see page 4368)	Indicates the number of headers in the list.
R	Headers (see page 4368)	Lists the headers.
	OwnsObjects (see page 4369)	Allows THeaderList (see page 4363) to free objects when they the list is cleared or destroyed.

38.1.4.2 THeaderList.Create Constructor

Creates and initializes a THeaderList (see page 4363) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall THeaderList();
```

Description

Create creates a THeaderList (see page 4363) instance.

38.1.4.3 THeaderList.Destroy Destructor

Destroys an instance of THeaderList (see page 4363).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THeaderList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the THeaderList (see page 4363) reference is not nil, and only then calls Destroy.

Destroy frees the THeaderList (see page 4363) instance. If the OwnsObjects (see page 4369) property is true, it also frees the header objects in the list.

See Also

System::TObject::Free

OwnsObjects (see page 4369)

38.1.4.4 THeaderList Methods

The methods of the THeaderList class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 4366)	This is GetCount, a member of class THeaderList.
	GetHeader (see page 4367)	This is GetHeader, a member of class THeaderList.
	GetOwnsObjects (see page 4367)	This is GetOwnsObjects, a member of class THeaderList.
	SetOwnsObjects (see page 4367)	This is SetOwnsObjects, a member of class THeaderList.

Public Methods

	Name	Description
	Add (see page 4365)	This is the overview for the Add method overload.
	Clear (see page 4366)	Empties the list and frees the TSOAPHeader (see page 4448) objects.
	Extract (see page 4366)	Removes a specified header from the list without freeing it.
	IndexOf (see page 4367)	This is IndexOf, a member of class THeaderList.

38.1.4.4.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	THeaderList.Add (TObject) (see page 4365)	Adds a new header to the end of the list.
	THeaderList.Add (TSOAPHeader) (see page 4365)	Adds a new header to the end of the list.

38.1.4.4.1.1 THeaderList.Add Method (TObject)

Adds a new header to the end of the list.

Pascal

```
procedure Add(Header: TSOAPHeader); overload;
procedure Add(Header: TObject); overload;
```

C++

```
__fastcall Add(TSOAPHeader Header);
__fastcall Add(TObject * Header);
```

Description

Call Add to add a header to the end of the Headers (see page 4368) property array. When you call Add, the new Header is added to the end of the array and the value of the Count (see page 4368) property is incremented.

Header is the TSOAPHeader (see page 4448) descendants that is added to the list.

See Also

Headers (see page 4368)

Count (see page 4368)

Clear (see page 4366)

38.1.4.4.2 THeaderList.Clear Method

Empties the list and frees the TSOAPHeader (see page 4448) objects.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear removes all of the headers from the list and, if the OwnsObjects (see page 4369) property is true, frees each one . After a call to Clear, the value of the Count (see page 4368) property is 0.

See Also

Headers (see page 4368)

Count (see page 4368)

OwnsObjects (see page 4369)

38.1.4.4.3 THeaderList.Extract Method

Removes a specified header from the list without freeing it.

Pascal

```
function Extract(Obj: TObject): TObject;
```

C++

```
__fastcall TObject * Extract(TObject * Obj);
```

Description

Call Extract to remove a header from the list without freeing the header object. After a header is removed, all the headers that follow it are moved up in index position and Count (see page 4368) is decremented.

See Also

Add (see page 4365)

Count (see page 4368)

Headers (see page 4368)

OwnsObjects (see page 4369)

38.1.4.4.4 THeaderList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class THeaderList.

38.1.4.4.5 THeaderList.GetHeader Method

Pascal

```
function GetHeader(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetHeader(int Index);
```

Description

This is GetHeader, a member of class THeaderList.

38.1.4.4.6 THeaderList.GetOwnsObjects Method

Pascal

```
function GetOwnsObjects: Boolean;
```

C++

```
__fastcall Boolean GetOwnsObjects();
```

Description

This is GetOwnsObjects, a member of class THeaderList.

38.1.4.4.7 THeaderList.IndexOf Method

Pascal

```
function IndexOf(Obj: TObject): Integer;
```

C++

```
__fastcall int IndexOf(TObject * Obj);
```

Description

This is IndexOf, a member of class THeaderList.

38.1.4.4.8 THeaderList.SetOwnsObjects Method

Pascal

```
procedure SetOwnsObjects(Val: Boolean);
```

C++

```
__fastcall SetOwnsObjects(Boolean Val);
```

Description

This is SetOwnsObjects, a member of class THeaderList.

38.1.4.5 THeaderList Properties

The properties of the THeaderList class are listed here.

Public Properties

	Name	Description
	Count (see page 4368)	Indicates the number of headers in the list.
	Headers (see page 4368)	Lists the headers.
	OwnsObjects (see page 4369)	Allows THeaderList (see page 4363) to free objects when they the list is cleared or destroyed.

38.1.4.5.1 THeaderList.Count Property

Indicates the number of headers in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of headers that the Headers (see page 4368) property lists. To change the value of Count, you can add headers using the Add (see page 4365) method, remove them using the Extract (see page 4366) method, or remove all headers using the Clear (see page 4366) method.

See Also

Headers (see page 4368)

Add (see page 4365)

Clear (see page 4366)

Extract (see page 4366)

38.1.4.5.2 THeaderList.Headers Property

Lists the headers.

Pascal

```
property Headers [Index: Integer]: TObject;
```

C++

```
__property TObject * Headers[int Index];
```

Description

Use Headers to access headers in the list. Headers is a zero-based array of TSOAPHeader (see page 4448) descendants: The first header is indexed as 0, the second header is indexed as 1, and so forth. You can read the value at a specific index, or use Headers with the Count (see page 4368) property to iterate through the list.

Note: In Delphi, Headers is the default property of THeaderList (see page 4363). This means you can omit the property name.

Note: In C++, you can use the subscript operator to achieve a similar effect.

See Also

OwnsObjects (see page 4369)

Add (see page 4365)

Extract (see page 4366)

Count (see page 4368)

Clear (see page 4366)

operator_sb

38.1.4.5.3 THeaderList.OwnsObjects Property

Allows THeaderList (see page 4363) to free objects when they the list is cleared or destroyed.

Pascal

```
property OwnsObjects: Boolean;
```

C++

```
__property Boolean OwnsObjects;
```

Description

OwnsObjects allows THeaderList (see page 4363) to control the memory of its TSOAPHeader (see page 4448) objects. If OwnsObjects is true,

calling Clear (see page 4366) frees all the headers in the list in addition to emptying the list.

freeing the THeaderList (see page 4363) object frees all the objects in the list in addition to destroying the THeaderList (see page 4363) itself.

Note: Even if OwnsObjects is true, the Extract (see page 4366) method removes objects from the list without freeing them.

See Also

Headers (see page 4368)

Clear

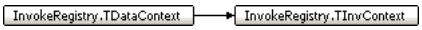
Destroy (see page 4364)

Extract (see page 4366)

38.1.5 TInvContext Class

TInvContext stores information about a method call on an invokeable interface.

Class Hierarchy



Pascal

```
TInvContext = class(TDataContext);
```

C++

```
class TInvContext : public TDataContext;
```

File

InvokeRegistry

Description

TInvContext is a helper object created and used by TRIO and TSoapPascalInvoker for generating calls to the methods on an invokeable interface. It stores temporary data needed when making the method call. TInvContext introduces methods for retrieving and setting information about the method being called, including its type signature, parameter values, and so on.

Component writers implementing their own classes to implement the IOPConvert interface must translate between the information stored in a TInvContext object and their class-specific encoding of a method call.

See Also

IOPConvert (see page 6152)

TRio

TSOAPPascalInvoker

38.1.5.1 TInvContext Members

The following tables list the members exposed by TInvContext.

Protected Fields

	Name	Description
	Data (see page 4356)	This is Data, a member of class TDataContext.
	DataOffset (see page 4356)	This is DataOffset, a member of class TDataContext.
	DataP (see page 4357)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 4357)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 4357)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 4357)	This is StrToClear, a member of class TDataContext.
	UStrToClear (see page 4357)	This is UStrToClear, a member of class TDataContext.
	VarToClear (see page 4358)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 4358)	This is WStrToClear, a member of class TDataContext.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 4358)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 4359)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 4359)	Stores a string that can be cleared when the data context is finished.
	AddUStrToClear (see page 4360)	This is AddUStrToClear, a member of class TDataContext.
	AddVariantToClear (see page 4360)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 4361)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 4361)	Returns a pointer to the data context's cache of raw memory.
	AllocServerData (see page 4371)	Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.
	Create (see page 4355)	Creates an instance of TDataContext (see page 4354).
	Destroy (see page 4355)	Destroys an instance of TDataContext (see page 4354).
	GetDataPointer (see page 4362)	Returns a pointer from the internal cache of pointers.
	GetParamPointer (see page 4371)	Returns a pointer to the value of a parameter on the called method.
	GetResultPointer (see page 4372)	Returns a pointer to the return value of the method call.
	RemoveObjectToDestroy (see page 4362)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 4362)	Stores a pointer in the internal cache of pointers.
	SetMethodInfo (see page 4372)	Initializes the TInvContext (see page 4369) for a particular method on the invokeable interface.
	SetParamPointer (see page 4373)	Sets the value of a specified parameter.
	SetResultPointer (see page 4373)	This is SetResultPointer, a member of class TInvContext.

38.1.5.2 TInvContext Methods

The methods of the TInvContext class are listed here.

Public Methods

	Name	Description
	AllocServerData (see page 4371)	Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.
	GetParamPointer (see page 4371)	Returns a pointer to the value of a parameter on the called method.
	GetResultPointer (see page 4372)	Returns a pointer to the return value of the method call.
	SetMethodInfo (see page 4372)	Initializes the TInvContext (see page 4369) for a particular method on the invokeable interface.
	SetParamPointer (see page 4373)	Sets the value of a specified parameter.
	SetResultPointer (see page 4373)	This is SetResultPointer, a member of class TInvContext.

38.1.5.2.1 TInvContext.AllocServerData Method

Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.

Pascal

```
procedure AllocServerData(const MD: TIntfMethEntry);
```

C++

```
__fastcall AllocServerData(const TIntfMethEntry MD);
```

Description

AllocServerData suballocates the memory that the SetMethodInfo (see page 4372) method allocates so that it can represent the parameters and return value of a method. It looks up the types of each parameter and of the return type for the method, allocates memory for their values using the AllocData (see page 4361) method, and stores the pointers to that memory by calling the SetParamPointer (see page 4373) and SetResultPointer (see page 4373) methods.

MD is a description of the method, including its type, calling convention, and parameter signature.

Note: AllocServerData must come after a call to SetMethodInfo (see page 4372) so that the pool of memory is available. After calling AllocServerData, an application can call GetParamPointer (see page 4371) and GetResultPointer (see page 4372) to obtain pointers to the memory allocated for each parameter and for the return value of the method.

See Also

AllocData (see page 4361)

SetParamPointer (see page 4373)

setresultpointer (see page 4373)

SetMethodInfo (see page 4372)

38.1.5.2.2 TInvContext.GetParamPointer Method

Returns a pointer to the value of a parameter on the called method.

Pascal

```
function GetParamPointer(Param: Integer): Pointer;
```

C++

```
__fastcall void * GetParamPointer(int Param);
```

Description

GetParamPointer returns a pointer to the value of a specified parameter.

Param is the index of the desired parameter, where 0 is the first parameter, 1 is the second parameter, and so on.

This value is assigned by a call to the SetParamPointer (see page 4373) method.

See Also

SetParamPointer (see page 4373)

38.1.5.2.3 TInvContext.GetResultPointer Method

Returns a pointer to the return value of the method call.

Pascal

```
function GetResultPointer: Pointer;
```

C++

```
__fastcall void * GetResultPointer();
```

Description

GetResultPointer returns a pointer to the return value of the method call. This value is assigned by a call to the SetResultPointer (see page 4373) method.

See Also

setresultpointer (see page 4373)

38.1.5.2.4 TInvContext.SetMethodInfo Method

Initializes the TInvContext (see page 4369) for a particular method on the invokeable interface.

Pascal

```
procedure SetMethodInfo(const MD: TIntfMethEntry);
```

C++

```
__fastcall SetMethodInfo(const TIntfMethEntry MD);
```

Description

SetMethodInfo initializes the invocation context so that it represents a particular method. Objects that implement the IOPConvert interface call this method to initialize an invocation context when implementing their MsgToInvContext method.

MD is a description of the method, including its type, calling convention, and parameter signature.

Note: SetMethodInfo allocates the memory used internally to store pointers and raw data. An application must call this method before any calls to the AllocData (see page 4361), SetDataPointer (see page 4362), or SetParamPointer (see page 4373) methods.

See Also

IOPConvert (see page 6152)

MsgToInvContext (see page 6157)

AllocData (see page 4361)

SetDataPointer (see page 4362)

SetParamPointer (see page 4373)

38.1.5.2.5 TInvContext.SetParamPointer Method

Sets the value of a specified parameter.

Pascal

```
procedure SetParamPointer(Param: Integer; P: Pointer);
```

C++

```
__fastcall SetParamPointer(int Param, void * P);
```

Description

SetParamPointer associates a pointer with a specified parameter.

Param identifies the parameter whose value is to be set, where 0 specifies the first parameter, 1 specifies the second parameter, and so on.

P is a pointer to the value of the parameter. Memory to contain the value of P can be allocated using the AllocData (see page 4361) method.

See Also

GetParamPointer (see page 4371)

AllocData (see page 4361)

38.1.5.2.6 TInvContext.SetResultPointer Method

Pascal

```
procedure SetResultPointer(P: Pointer);
```

C++

```
__fastcall SetResultPointer(void * P);
```

Description

This is SetResultPointer, a member of class TInvContext.

38.1.6 TInvokableClass Class

TInvokableClass is the base class for the implementation classes of invokable interfaces.

Class Hierarchy

Pascal

```
TInvokableClass = class(TObject, IInterface);
```

C++

```
class TInvokableClass : public TObject, public IInterface;
```

File

InvokeRegistry

Description

Use TInvokableClass as a base class when writing the implementation class of an invokable interface. TInvokableClass has the following important attributes:

The invocation registry (InvRegistry (see page 4471)) knows how to create instances of TInvokableClass and (because it has a virtual constructor) its descendants. This allows the registry to supply an invoker in a Web Service application with an instance of the invokable class that can handle an incoming request.

TInvokableClass is an interfaced object that frees itself when the reference count on its interface drops to zero. Invoker components do not know when to free the implementation classes of the interfaces they call. Because TInvokableClass knows when to free itself, you do not need to supply your own lifetime management for this object.

TInvokableClass supports the ISOAPHeaders (see page 4464) interface so that your server can process the headers that accompany requests and add headers to outgoing responses.

To use a TInvokableClass descendant in a Web Service application, you must register it with the invocation registry. Use the RegisterInvokableClass method of the invocation registry to register your TInvokableClass descendant.

Note: If you do not use TInvokableClass as a base class when writing the implementation class for an invokable interface, you must supply the invocation registry with a method to retrieve instances of your class and you must handle the lifetime management of your class. If you are sending or receiving SOAP headers, you must use the global GetSOAPHeaders function to obtain an ISOAPHeaders (see page 4464) interface instead of obtaining one from the implementation class.

See Also

InvRegistry (see page 4471)

IInvokable

TInvokableClassRegistry (see page 4379)

THTTPSoapPascalInvoker

ISOAPHeaders (see page 4464)

Adding New Web Services

38.1.6.1 TInvokableClass Members

The following tables list the members exposed by TInvokableClass.

Protected Fields

	Name	Description
	FRefCount (see page 4375)	This is FRefCount, a member of class TInvokableClass.
	FSOAPHeaders (see page 4376)	This is FSOAPHeaders, a member of class TInvokableClass.

Protected Methods

	Name	Description
	_AddRef (see page 4376)	Increments the reference count for this interface.
	_Release (see page 4377)	Decrements the reference count for this interface.
	QueryInterface (see page 4378)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 4377)	Responds after the last constructor executes.
	BeforeDestruction (see page 4377)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	Create (see page 4375)	Creates and initializes a TInvokableClass (see page 4373) object.
	NewInstance (see page 4378)	Allocates memory for each instance of an invokable object.

Public Properties

	Name	Description
	RefCount (see page 4379)	Indicates the number of interface pointers currently dependent upon the object.

38.1.6.2 TInvokableClass.Create Constructor

Creates and initializes a TInvokableClass (see page 4373) object.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TInvokableClass();
```

Description

The TInvokableClass (see page 4373) constructor does nothing special. It exists primarily because

The TInvokableClass (see page 4373) constructor exists primarily because

The invocation registry knows how to call it.

It is virtual, so that the invocation registry can also instantiate descendant classes.

In addition, the constructor initializes the invokable class so that it can support the ISOAPHeaders (see page 4464) interface.

See Also

InvRegistry (see page 4471)

ISOAPHeaders (see page 4464)

38.1.6.3 TInvokableClass Fields

The fields of the TInvokableClass class are listed here.

Protected Fields

	Name	Description
	FRefCount (see page 4375)	This is FRefCount, a member of class TInvokableClass.
	FSOAPHeaders (see page 4376)	This is FSOAPHeaders, a member of class TInvokableClass.

38.1.6.3.1 TInvokableClass.FRefCount Field

Pascal

```
FRefCount: Integer;
```

C++

```
int FRefCount;
```

Description

This is FRefCount, a member of class TInvokableClass.

38.1.6.3.2 TInvokableClass.FSOAPHeaders Field

Pascal

```
FSOAPHeaders: TSOAPHeaders;
```

C++

```
TSOAPHeaders FSOAPHeaders;
```

Description

This is FSOAPHeaders, a member of class TInvokableClass.

38.1.6.4 TInvokableClass Methods

The methods of the TInvokableClass class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 4376)	Increments the reference count for this interface.
	_Release (see page 4377)	Decrements the reference count for this interface.
	QueryInterface (see page 4378)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 4377)	Responds after the last constructor executes.
	BeforeDestruction (see page 4377)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	NewInstance (see page 4378)	Allocates memory for each instance of an invokable object.

38.1.6.4.1 TInvokableClass._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release

38.1.6.4.2 TInvokableClass._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef

38.1.6.4.3 TInvokableClass.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of TInvokableClass (see page 4373) is allocated, RefCount (see page 4379) is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that RefCount (see page 4379) to re-establish the correct reference count after all constructors have executed.

See Also

BeforeDestruction (see page 4377)

NewInstance (see page 4378)

RefCount (see page 4379)

38.1.6.4.4 TInvokableClass.BeforeDestruction Method

Checks that the reference count is zero before allowing the interfaced object to be destroyed.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called automatically immediately before the interfaced object's destructor. It checks that the reference count is zero, and, if it is not, raises an exception.

See Also

RefCount (see page 4379)

System::TObject::Free

38.1.6.4.5 TInvokableClass.NewInstance Method

Allocates memory for each instance of an invokable object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

Constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TInvokableClass (see page 4373) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a RefCount (see page 4379) of zero. The AfterConstruction (see page 4377) method decrements the value of RefCount (see page 4379) after all constructors have executed.

See Also

RefCount (see page 4379)

System::TObject::NewInstance

System::TObject::InstanceSize

AfterConstruction (see page 4377)

38.1.6.4.6 TInvokableClass.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

Do not call the protected QueryInterface method directly. QueryInterface is called through the IInterface interface to obtain an

interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and QueryInterface returns S_OK. If the object does not support the interface, QueryInterface returns E_NOINTERFACE.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

_addref (see page 4376)

38.1.6.5 TInvokableClass Properties

The properties of the TInvokableClass class are listed here.

Public Properties

	Name	Description
	RefCount (see page 4379)	Indicates the number of interface pointers currently dependent upon the object.

38.1.6.5.1 TInvokableClass.RefCount Property

Indicates the number of interface pointers currently dependent upon the object.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount defines the lifetime of the object. When RefCount is zero, the object is destroyed.

RefCount is incremented by calls to the _AddRef (see page 4376) method and decremented by the _Release (see page 4377) method.

Note:

See Also

_addref (see page 4376)

_release (see page 4377)

38.1.7 TInvokableClassRegistry Class

TInvokableClassRegistry is the type of the invocation registry.

Class Hierarchy

Pascal

```
TInvokableClassRegistry = class(TInterfacedObject);
```

C++

```
class TInvokableClassRegistry : public TInterfacedObject;
```

File

InvokeRegistry

Description

TInvokableClassRegistry manages information about invokable interfaces and the classes that implement them. Invokable interfaces are used in applications that work with Web Services.

Server applications that implement Web Services need to register both the invokable interfaces and the implementation classes. The invoker component (such as THTTPSoapPascalInvoker or THTTPSoapCppInvoker) can then use the invocation registry to locate invokable interfaces and execute calls made to them. In addition, server applications should register the headers that are used with calls to the methods of the invokable interface and the exceptions that such calls can raise.

Client applications that call on Web Services register the invokable interfaces that they call. The remote interfaced object component (THTTPRio) can then use the invocation registry to generate a dynamic vtable for calling the invokable interfaces.

Do not create instances of TInvokableClassRegistry. Instead, use the global InvRegistry (see page 4471) function to access the invocation registry. Most applications need only call the RegisterInterface (see page 4404) and (if a server) the RegisterInvokableClass (see page 4405) methods. The other TInvokableClassRegistry methods are used by invoker components, remote interfaced object components, and the WSDL document importer. The allow these components to look up information about invokable interfaces and their implementation classes, access invokable interfaces, and so on.

See Also

InvRegistry (see page 4471)

IInvokable

THTTPSoapCppInvoker

THTTPSoapPascalInvoker

THTTPRio

38.1.7.1 TInvokableClassRegistry Members

The following tables list the members exposed by TInvokableClassRegistry.

Public Methods

	Name	Description
	Create (see page 4381)	Creates and initializes a TInvokableClassRegistry (see page 4379) object.
	Destroy (see page 4382)	Frees the memory associated with the TInvokableClassRegistry (see page 4379) object.
	GetActionURIOfIID (see page 4383)	Returns the SOAP Action header associated with the interface with a specified GUID.
	GetActionURIOfInfo (see page 4384)	Returns the SOAP Action header associated with the type info of a registered interface.
	GetClassFromIntfInfo (see page 4384)	Retrieves the implementation class that corresponds to the type info of an invokable interface.
	GetExceptionInfoForInterface (see page 4385)	Retrieves an array of exceptions that are registered as associated with an invokable interface.
	GetHeaderClass (see page 4385)	Returns the class reference for a header type, given its name and namespace.
	GetHeaderInfoForInterface (see page 4386)	Retrieves an array of headers that are registered as associated with an invokable interface.
	GetHeaderName (see page 4386)	Returns the name of a header that is identified by a specified class reference.
	GetHeaderNamespace (see page 4387)	This is the overview for the GetHeaderNamespace method overload.
	GetInfoForURI (see page 4388)	Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.

	GetInterfaceCount (see page 4388)	Returns the number of registered invokable interfaces.
	GetInterfaceExternalName (see page 4389)	This is the overview for the GetInterfaceExternalName method overload.
	GetInterfaceInfoFromName (see page 4390)	Returns information about a registered invokable interface given its name.
	GetInterfaceTypeInfo (see page 4390)	Returns the runtime type information (RTTI) of an invokable interface.
	GetIntfInvokeOptions (see page 4390)	This is the overview for the GetIntfInvokeOptions method overload.
	GetInvokableObjectFromClass (see page 4391)	Returns an instance of the implementation class for a registered invokable interface.
	GetMethExternalName (see page 4392)	Returns the name of the operation that corresponds to a method on an invokable interface.
	GetMethInternalName (see page 4392)	Returns the name of the method that corresponds to an operation in a WSDL document.
	GetNamespaceByGUID (see page 4393)	Returns a namespace that is uniquely associated with a registered interface.
	GetParamExternalName (see page 4393)	Returns the name of the part that corresponds to a parameter on a method of an invokable interface.
	GetParamInternalName (see page 4393)	Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.
	GetRegInterfaceEntry (see page 4394)	Returns the internal data that TInvokableClassRegistry (see page 4379) maintains for one of its registered interfaces.
	GetRequestHeaderInfoForInterface (see page 4394)	Retrieves an array of headers that are registered as associated with requests to an invokable interface.
	GetResponseHeaderInfoForInterface (see page 4395)	Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.
	GetReturnParamNames (see page 4396)	Returns the list of parameter names that map to return values of interface methods.
	GetUDDIInfo (see page 4396)	This is the overview for the GetUDDIInfo method overload.
	GetWSDLEncoding (see page 4397)	This is the overview for the GetWSDLEncoding method overload.
	HasRegInterfaceImpl (see page 4398)	Indicates whether there is a registered implementation class for an invokable interface.
	RegisterAllSOAPActions (see page 4398)	Supplies a set of SOAP Action headers to use for an already-registered interface.
	RegisterDefaultSOAPAction (see page 4399)	Supplies a SOAP Action header to use for an already-registered interface.
	RegisterException (see page 4399)	Registers the remotable exception classes that calls to an invokable interface can raise.
	RegisterExceptionMethod (see page 4400)	Registers methods of an invokable interface that can raise a registered exception class.
	RegisterExternalMethName (see page 4400)	Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.
	RegisterExternalParamName (see page 4401)	Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.
	RegisterHeaderClass (see page 4402)	This is the overview for the RegisterHeaderClass method overload.
	RegisterHeaderMethod (see page 4403)	Registers methods of an invokable interface that make use of a registered header.
	RegisterInterface (see page 4404)	Registers an invokable interface.
	RegisterInvokableClass (see page 4405)	This is the overview for the RegisterInvokableClass method overload.
	RegisterInvokeOptions (see page 4405)	This is the overview for the RegisterInvokeOptions method overload.
	RegisterReturnParamNames (see page 4406)	Registers a list of parameter names that map to return values of interface methods.
	RegisterUDDIInfo (see page 4407)	Registers the location of the UDDI description that can be used to import the location of a Web Service.
	UnRegisterInterface (see page 4407)	Unregisters an invokable interface.
	UnRegisterInvokableClass (see page 4408)	Unregisters the implementation class for an invokable interface.

38.1.7.2 TInvokableClassRegistry.Create Constructor

Creates and initializes a TInvokableClassRegistry (see page 4379) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TInvokableClassRegistry();
```

Description

Do not create your own instances of TInvokableClassRegistry (see page 4379). Instead, use the global InvRegistry (see page 4471) function, which provides access to a single global instance of TInvokableClassRegistry (see page 4379).

See Also

InvRegistry (see page 4471)

38.1.7.3 TInvokableClassRegistry.Destroy Destructor

Frees the memory associated with the TInvokableClassRegistry (see page 4379) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInvokableClassRegistry();
```

Description

Do not destroy the global TInvokableClassRegistry (see page 4379) instance that is available through the InvRegistry (see page 4471) function. This object is freed automatically when the application unloads the InvokeRegistry (see page 4347) unit.

See Also

InvRegistry (see page 4471)

38.1.7.4 TInvokableClassRegistry Methods

The methods of the TInvokableClassRegistry class are listed here.

Public Methods

	Name	Description
	GetActionURIOfIID (see page 4383)	Returns the SOAP Action header associated with the interface with a specified GUID.
	GetActionURIOfInfo (see page 4384)	Returns the SOAP Action header associated with the type info of a registered interface.
	GetClassFromIntfInfo (see page 4384)	Retrieves the implementation class that corresponds to the type info of an invokable interface.
	GetExceptionInfoForInterface (see page 4385)	Retrieves an array of exceptions that are registered as associated with an invokable interface.
	GetHeaderClass (see page 4385)	Returns the class reference for a header type, given its name and namespace.
	GetHeaderInfoForInterface (see page 4386)	Retrieves an array of headers that are registered as associated with an invokable interface.
	GetHeaderName (see page 4386)	Returns the name of a header that is identified by a specified class reference.
	GetHeaderNamespace (see page 4387)	This is the overview for the GetHeaderNamespace method overload.
	GetInfoForURI (see page 4388)	Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.
	GetInterfaceCount (see page 4388)	Returns the number of registered invokable interfaces.
	GetInterfaceExternalName (see page 4389)	This is the overview for the GetInterfaceExternalName method overload.
	GetInterfaceInfoFromName (see page 4390)	Returns information about a registered invokable interface given its name.
	GetInterfaceTypeInfo (see page 4390)	Returns the runtime type information (RTTI) of an invokable interface.
	GetIntfInvokeOptions (see page 4390)	This is the overview for the GetIntfInvokeOptions method overload.

	GetInvokableObjectFromClass (see page 4391)	Returns an instance of the implementation class for a registered invokable interface.
	GetMethExternalName (see page 4392)	Returns the name of the operation that corresponds to a method on an invokable interface.
	GetMethInternalName (see page 4392)	Returns the name of the method that corresponds to an operation in a WSDL document.
	GetNamespaceByGUID (see page 4393)	Returns a namespace that is uniquely associated with a registered interface.
	GetParamExternalName (see page 4393)	Returns the name of the part that corresponds to a parameter on a method of an invokable interface.
	GetParamInternalName (see page 4393)	Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.
	GetRegInterfaceEntry (see page 4394)	Returns the internal data that TInvokableClassRegistry (see page 4379) maintains for one of its registered interfaces.
	GetRequestHeaderInfoForInterface (see page 4394)	Retrieves an array of headers that are registered as associated with requests to an invokable interface.
	GetResponseHeaderInfoForInterface (see page 4395)	Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.
	GetReturnParamNames (see page 4396)	Returns the list of parameter names that map to return values of interface methods.
	GetUDDIInfo (see page 4396)	This is the overview for the GetUDDIInfo method overload.
	GetWSDLEncoding (see page 4397)	This is the overview for the GetWSDLEncoding method overload.
	HasRegInterfaceImpl (see page 4398)	Indicates whether there is a registered implementation class for an invokable interface.
	RegisterAllSOAPActions (see page 4398)	Supplies a set of SOAP Action headers to use for an already-registered interface.
	RegisterDefaultSOAPAction (see page 4399)	Supplies a SOAP Action header to use for an already-registered interface.
	RegisterException (see page 4399)	Registers the remotable exception classes that calls to an invokable interface can raise.
	RegisterExceptionMethod (see page 4400)	Registers methods of an invokable interface that can raise a registered exception class.
	RegisterExternalMethName (see page 4400)	Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.
	RegisterExternalParamName (see page 4401)	Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.
	RegisterHeaderClass (see page 4402)	This is the overview for the RegisterHeaderClass method overload.
	RegisterHeaderMethod (see page 4403)	Registers methods of an invokable interface that make use of a registered header.
	RegisterInterface (see page 4404)	Registers an invokable interface.
	RegisterInvokableClass (see page 4405)	This is the overview for the RegisterInvokableClass method overload.
	RegisterInvokeOptions (see page 4405)	This is the overview for the RegisterInvokeOptions method overload.
	RegisterReturnParamNames (see page 4406)	Registers a list of parameter names that map to return values of interface methods.
	RegisterUDDIInfo (see page 4407)	Registers the location of the UDDI description that can be used to import the location of a Web Service.
	UnRegisterInterface (see page 4407)	Unregisters an invokable interface.
	UnRegisterInvokableClass (see page 4408)	Unregisters the implementation class for an invokable interface.

38.1.7.4.1 TInvokableClassRegistry.GetActionURIOfIID Method

Returns the SOAP Action header associated with the interface with a specified GUID.

Pascal

```
function GetActionURIOfIID(const AGUID: TGUID): string;
```

C++

```
__fastcall UnicodeString GetActionURIOfIID(const TGUID AGUID);
```

Description

Typically, applications do not need to call GetActionURIOfIID. This method is called by remote interfaced objects (THTTPRIO) when generating the SOAP Action header of messages that call a method on a registered interface.

AGUID is the globally unique identifier (GUID) of the interface to call. The interface must be previously registered by a call to the RegisterInterface (see page 4404) method.

GetActionURIOfIID returns the string that should appear as the SOAP Action header on request messages to that interface.

See Also

GetActionURIOfInfo (see page 4384)

RegisterDefaultSOAPAction (see page 4399)

RegisterInterface (see page 4404)

THTTPRIO

38.1.7.4.2 TInvokableClassRegistry.GetActionURIOfInfo Method

Returns the SOAP Action header associated with the type info of a registered interface.

Pascal

```
function GetActionURIOfInfo(const IntfInfo: PTypeInfo; const MethodName: InvString;
MethodIndex: Integer): string;
```

C++

```
__fastcall UnicodeString GetActionURIOfInfo(const PTypeInfo IntfInfo, const InvString
MethodName, int MethodIndex);
```

Description

GetActionURIOfInfo looks up or generates the SOAP Action header that should be used with messages that call a specified method on an invokable interface.

IntfInfo (see page 4331) is the runtime type information (RTTI) that is compiled for the invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

MethodName is the name of the method to call on the specified interface.

MethodIndex indicates which method to use when the interface includes an overload of the method that MethodName specifies. 0 indicates the overload that is declared first, 1 the overload that is declared second, and so on.

GetActionURIOfInfo returns the string that should appear as the SOAP Action header on request messages to the specified method of the specified interface.

See Also

GetActionURIOfIID (see page 4383)

RegisterDefaultSOAPAction (see page 4399)

RegisterInterface (see page 4404)

THTTPRIO

38.1.7.4.3 TInvokableClassRegistry.GetClassFromIntfInfo Method

Retrieves the implementation class that corresponds to the type info of an invokable interface.

Pascal

```
procedure GetClassFromIntfInfo(Info: PTypeInfo; var AClass: TClass);
```

C++

```
__fastcall GetClassFromIntfInfo(PTypeInfo Info, TClass AClass);
```

Description

Typically, applications do not need to call GetClassFromIntfInfo. This method allows the invoker component (THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to look up the implementation class for an interface it is invoking.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

AClass returns a class reference for the class that implements the invokable interface identified by Info. It can be passed to the GetInvokableObjectFromClass (see page 4391) method to obtain an instance of the implementation class.

See Also

GetInvokableObjectFromClass (see page 4391)

RegisterInvokableClass (see page 4405)

RegisterInterface (see page 4404)

THTTPSoapPascalInvoker

THTTPSoapCppInvoker

38.1.7.4.4 TInvokableClassRegistry.GetExceptionInfoForInterface Method

Retrieves an array of exceptions that are registered as associated with an invokable interface.

Pascal

```
function GetExceptionInfoForInterface(Info: PTypeInfo): TExceptionItemArray;
```

C++

```
__fastcall TExceptionItemArray GetExceptionInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetExceptionInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the exceptions that an invokable interface can raise.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

GetExceptionInfoForInterface returns an array of IntfExceptionItem (see page 4476) values, each of which describes a remotable exception that was registered for the invokable interface by a call to the RegisterException (see page 4399) method.

See Also

RegisterException (see page 4399)

TWSDLHTMLPublish

RegisterInterface (see page 4404)

38.1.7.4.5 TInvokableClassRegistry.GetHeaderClass Method

Returns the class reference for a header type, given its name and namespace.

Pascal

```
function GetHeaderClass(Name: InvString; Namespace: InvString): TClass;
```

C++

```
__fastcall TClass GetHeaderClass(InvString Name, InvString Namespace);
```

Description

Typically, applications do not need to call GetHeaderClass. This method is used internally to look up the class of a header that is identified by name.

Name is the name of the header node.

Namespace is the URI of the namespace in which Name is defined.

GetHeaderClass returns the class reference for the TSOAPHeader (see page 4448) descendant that represents the identified header.

See Also

RegisterHeaderClass (see page 4402)

TSOAPHeader (see page 4448)

GetHeaderName (see page 4386)

GetHeaderNamespace (see page 4387)

38.1.7.4.6 TInvokableClassRegistry.GetHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with an invokable interface.

Pascal

```
function GetHeaderInfoForInterface(Info: PTypeInfo; MethodType: eHeaderMethodType = hmtAll):
THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetHeaderInfoForInterface(PTypeInfo Info, eHeaderMethodType
MethodType = hmtAll);
```

Description

Typically, applications do not need to call GetHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in messages about an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

MethodType indicates whether to retrieve information about the headers to use on request messages, the headers to use on response messages, or both.

GetHeaderInfoForInterface returns an array of IntfHeaderItem (see page 4477) values, each of which describes a header that was registered for the invokable interface by a call to the RegisterHeaderClass (see page 4402) method.

See Also

RegisterHeaderClass (see page 4402)

TWSDLHTMLPublish

RegisterInterface (see page 4404)

38.1.7.4.7 TInvokableClassRegistry.GetHeaderName Method

Returns the name of a header that is identified by a specified class reference.

Pascal

```
function GetHeaderName(Info: PTypeInfo; AClass: TClass): InvString;
```

C++

```
__fastcall InvString GetHeaderName(PTypeInfo Info, TClass AClass);
```

Description

GetHeaderName returns the name of a header given the class that represents it. This is the name supplied to a call to RegisterHeaderClass (see page 4402) when the header was registered. If the header was registered without supplying a name, this is the same name as the type name supplied for the header when it is registered with the remotable type registry.

Info is the runtime type information for the interface associated with the header.

AClass is a class reference to the TSOAPHeader (see page 4448) descendant that represents the header.

GetHeaderName returns the name of the header. This name is defined within the namespace that the GetHeaderNamespace (see page 4387) method returns.

See Also

RegisterHeaderClass (see page 4402)

TSOAPHeader (see page 4448)

GetHeaderNamespace (see page 4387)

GetHeaderClass (see page 4385)

38.1.7.4.8 GetHeaderNamespace Method

This is the overview for the GetHeaderNamespace method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetHeaderNamespace (PTypeInfo, TClass) (see page 4387)	Returns the namespace URI for a header that is identified by a specified class reference.
	TInvokableClassRegistry.GetHeaderNamespace (TClass) (see page 4387)	Returns the namespace URI for a header that is identified by a specified class reference.

38.1.7.4.8.1 TInvokableClassRegistry.GetHeaderNamespace Method (PTypeInfo, TClass)

Returns the namespace URI for a header that is identified by a specified class reference.

Pascal

```
function GetHeaderNamespace(Info: PTypeInfo; AClass: TClass): InvString; overload;
function GetHeaderNamespace(AClass: TClass): InvString; overload;
```

C++

```
__fastcall InvString GetHeaderNamespace(PTypeInfo Info, TClass AClass);
__fastcall InvString GetHeaderNamespace(TClass AClass);
```

Description

GetHeaderNamespace returns the namespace in which a header's name is defined. This is the namespace URI supplied to a call to RegisterHeaderClass (see page 4402) when the header was registered. If the header was registered without supplying a namespace URI, this is the same namespace as the one associated with the interface that uses the header.

Info is the runtime type information for the interface associated with the header.

AClass is a class reference to the TSOAPHeader (see page 4448) descendant that represents the header.

GetHeaderNamespace returns the namespace URI to use for resolving the name of the header.

See Also

RegisterHeaderClass (see page 4402)

TSOAPHeader (see page 4448)

GetHeaderName (see page 4386)

GetHeaderClass (see page 4385)

38.1.7.4.9 TInvokableClassRegistry.GetInfoForURI Method

Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.

Pascal

```
function GetInfoForURI(const PathURI: string; const ActionURI: string; var AClass: TClass; var
IntfInfo: PTypeInfo; var AMeth: string): Boolean;
```

C++

```
__fastcall Boolean GetInfoForURI(const UnicodeString PathURI, const UnicodeString ActionURI,
TClass AClass, PTypeInfo IntfInfo, UnicodeString AMeth);
```

Description

Typically, applications do not need to call GetInfoForURI. This method is called by the invoker component (THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to look up the target of a SOAP message that it is invoking.

PathURI is the path portion of the target URI on the SOAP message. When using a path to identify an invokable interface, the path portion must end in the name of the unit followed by the name of the interface. It is only used if ActionURI is an empty string.

ActionURI is the SOAP action header of the SOAP message. It identifies an interface based on the namespace in the SOAP action header. This namespace must match the namespace supplied to the RegisterInterface (see page 4404) method (or generated by that method if the caller did not supply a namespace). If ActionURI is an empty string, GetInfoForURI uses PathURI instead to identify the target interface.

AClass returns a class reference for the class that implements the invokable interface identified by PathURI or ActionURI.

IntfInfo (see page 4331) returns the runtime type information (RTTI) of the invokable interface identified by PathURI or ActionURI.

AMeth is not currently used.

See Also

DispatchSOAP

RegisterInterface (see page 4404)

38.1.7.4.10 TInvokableClassRegistry.GetInterfaceCount Method

Returns the number of registered invokable interfaces.

Pascal

```
function GetInterfaceCount: Integer;
```

C++

```
__fastcall int GetInterfaceCount();
```

Description

GetInterfaceCount returns the number of invokable interfaces that have been registered. Each invokable interface is registered by a call to the RegisterInterface (see page 4404) method.

See Also

RegisterInterface (see page 4404)

38.1.7.4.11 GetInterfaceExternalName Method

This is the overview for the GetInterfaceExternalName method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo) (see page 4389)	Returns the name of the port type that corresponds to an invokable interface.
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo, string) (see page 4389)	Returns the name of the port type that corresponds to an invokable interface.
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo, string, string) (see page 4389)	Returns the name of the port type that corresponds to an invokable interface.

38.1.7.4.11.1 TInvokableClassRegistry.GetInterfaceExternalName Method (PTypeInfo)

Returns the name of the port type that corresponds to an invokable interface.

Pascal

```
function GetInterfaceExternalName(Info: PTypeInfo; const Namespace: string; const
InternalIntfName: string): InvString; overload;
function GetInterfaceExternalName(Info: PTypeInfo; const Namespace: string): InvString;
overload;
function GetInterfaceExternalName(Info: PTypeInfo): InvString; overload;
```

C++

```
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info, const UnicodeString Namespace,
const UnicodeString InternalIntfName);
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info, const UnicodeString Namespace);
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info);
```

Description

GetInterfaceExternalName returns the name of the port type that corresponds to a specified invokable interface. This is the name given to the corresponding port type in a WSDL document. The association between the external name and the interface name is established by the RegisterInterface (see page 4404) method.

Info is the type information pointer for the invokable interface.

Namespace is the namespace assigned to the invokable interface when it was registered by a call to the RegisterInterface (see page 4404) method. If the interface was registered using the automatically-generated namespace, this parameter can be omitted or you can supply an empty string.

InternalIntfName is the name of the invokable interface.

See Also

RegisterInterface (see page 4404)

38.1.7.4.12 TInvokableClassRegistry.GetInterfaceInfoFromName Method

Returns information about a registered invokable interface given its name.

Pascal

```
procedure GetInterfaceInfoFromName(const UnitName: string; const IntfName: string; var Info:
PTypeInfo; var IID: TGUID);
```

C++

```
__fastcall GetInterfaceInfoFromName(const UnicodeString UnitName, const UnicodeString
IntfName, PTypeInfo Info, TGUID IID);
```

Description

GetInterfaceInfoFromName looks up a registered interface given its name and the name of the unit in which it is declared.

UnitName is the name of the unit that contains the interface definition.

IntfName is the name of the interface.

Info returns the type information that is compiled with the specified interface.

IID returns the globally unique identifier for the specified interface.

See Also

GetInterfaceTypeInfo (see page 4390)

GetInfoForURI (see page 4388)

38.1.7.4.13 TInvokableClassRegistry.GetInterfaceTypeInfo Method

Returns the runtime type information (RTTI) of an invokable interface.

Pascal

```
function GetInterfaceTypeInfo(const AGUID: TGUID): Pointer;
```

C++

```
__fastcall void * GetInterfaceTypeInfo(const TGUID AGUID);
```

Description

GetInterfaceTypeInfo returns a pointer to the RTTI that is compiled with an invokable interface.

AGUID is the globally unique identifier that specifies the desired interface.

See Also

GetInterfaceInfoFromName (see page 4390)

GetInfoForURI (see page 4388)

38.1.7.4.14 GetIntfInvokeOptions Method

This is the overview for the GetIntfInvokeOptions method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetIntfInvokeOptions (PTypeInfo) (see page 4391)	Retrieves any flags that customize the way calls to a registered interface are made.
	TInvokableClassRegistry.GetIntfInvokeOptions (TGUID) (see page 4391)	Retrieves any flags that customize the way calls to a registered interface are made.

38.1.7.4.14.1 TInvokableClassRegistry.GetIntfInvokeOptions Method (PTypeInfo)

Retrieves any flags that customize the way calls to a registered interface are made.

Pascal

```
function GetIntfInvokeOptions(const IntfInfo: PTypeInfo): TIntfInvokeOptions; overload;
function GetIntfInvokeOptions(const AGUID: TGUID): TIntfInvokeOptions; overload;
```

C++

```
__fastcall TIntfInvokeOptions GetIntfInvokeOptions(const PTypeInfo IntfInfo);
__fastcall TIntfInvokeOptions GetIntfInvokeOptions(const TGUID AGUID);
```

Description

GetIntfInvokeOptions retrieves any invocation options that have been registered for an interface using the RegisterInvokeOptions (see page 4406) method. Invocation options customize the way a registered interface is called, especially the way such calls are marshaled or unmarshaled. Remote interfaced objects (THTTPRIO) call this method to determine what options they need to set on the component that is the value of their Converter property.

AGUID identifies the registered interface using its GUID.

IntfInfo (see page 4331) identifies the registered interface using its runtime type information (RTTI). You can obtain the type information from an invokable interface by calling the TypeInfo function (delphi) or the __delphirtti operator (C++).

GetIntfInvokeOptions returns a set of flags that indicate how to make calls to the specified interface.

See Also

RegisterInvokeOptions (see page 4406)

Converter

Options (see page 6201)

RegisterInterface (see page 4404)

38.1.7.4.15 TInvokableClassRegistry.GetInvokableObjectFromClass Method

Returns an instance of the implementation class for a registered invokable interface.

Pascal

```
function GetInvokableObjectFromClass(AClass: TClass): TObject;
```

C++

```
__fastcall TObject * GetInvokableObjectFromClass(TClass AClass);
```

Description

Typically, applications do not need to call GetInvokableObjectFromClass. This method is called by the invoker component (THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to obtain an instance of the class that implements a registered invokable interface. When the invoker is finished with the object, it calls GetWSDLEncoding (see page 4397) to free the instance.

AClass identifies the class whose instance is desired. It must be registered by a call to RegisterInvokableClass (see page 4405). The invoker obtains this class reference by calling the GetInfoForURI (see page 4388) method.

See Also

GetInfoForURI (see page 4388)

RegisterInvokableClass (see page 4405)

GetWSDLEncoding (see page 4397)

38.1.7.4.16 TInvokableClassRegistry.GetMethExternalName Method

Returns the name of the operation that corresponds to a method on an invokable interface.

Pascal

```
function GetMethExternalName(Info: PTypeInfo; const MethodIntName: string): InvString;
```

C++

```
__fastcall InvString GetMethExternalName(PTypeInfo Info, const UnicodeString MethodIntName);
```

Description

GetMethExternalName returns the name of the operation that corresponds to a specified method. This is the name used to define the operation for that method in a WSDL document. The association between the external name and the method name is established by the RegisterExternalMethName (see page 4400) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodIntName is the name of the method on that invokable interface.

See Also

RegisterExternalMethName (see page 4400)

GetMethInternalName (see page 4392)

38.1.7.4.17 TInvokableClassRegistry.GetMethInternalName Method

Returns the name of the method that corresponds to an operation in a WSDL document.

Pascal

```
function GetMethInternalName(Info: PTypeInfo; const MethodExtName: InvString): string;
```

C++

```
__fastcall UnicodeString GetMethInternalName(PTypeInfo Info, const InvString MethodExtName);
```

Description

GetMethInternalName returns the name of a method on a registered invokable interface that corresponds to a specified operation. The operation name comes from a WSDL document that describes the invokable interface and its methods. The association between the operation name and the method name is established by the RegisterExternalMethName (see page 4400) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodExtName is the name of the operation for which you want the corresponding method name.

See Also

RegisterExternalMethName (see page 4400)

GetMethExternalName (see page 4392)

38.1.7.4.18 TInvokableClassRegistry.GetNamespaceByGUID Method

Returns a namespace that is uniquely associated with a registered interface.

Pascal

```
function GetNamespaceByGUID(const AGUID: TGUID): string;
```

C++

```
__fastcall UnicodeString GetNamespaceByGUID(const TGUID AGUID);
```

Description

GetNamespaceByGUID returns the namespace URI that identifies the specified interface. The namespace is originally associated with the interface when it is registered by a call to the RegisterInterface (see page 4404) method. This namespace can be supplied when the interface is registered, or automatically generated by the RegisterInterface (see page 4404) method.

AGUID is the globally unique identifier that identifies the interface whose namespace is desired..

See Also

GetInfoForURI (see page 4388)

RegisterInterface (see page 4404)

38.1.7.4.19 TInvokableClassRegistry.GetParamExternalName Method

Returns the name of the part that corresponds to a parameter on a method of an invokable interface.

Pascal

```
function GetParamExternalName(Info: PTypeInfo; const MethodName: string; const
InternalParamName: string): InvString;
```

C++

```
__fastcall InvString GetParamExternalName(PTypeInfo Info, const UnicodeString MethodName,
const UnicodeString InternalParamName);
```

Description

GetParamExternalName returns the name of the part that corresponds to a specified method parameter. This is the name used to define the part for that parameter in a WSDL document. The association between the external name and the parameter name is established by the RegisterExternalParamName (see page 4401) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodName is the name of the method on that invokable interface.

InternalParamName is the name of the parameter on the method of the invokable interface.

See Also

RegisterExternalParamName (see page 4401)

GetParamInternalName (see page 4393)

38.1.7.4.20 TInvokableClassRegistry.GetParamInternalName Method

Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.

Pascal

```
function GetParamInternalName(Info: PTypeInfo; const MethodName: string; const
ExternalParamName: InvString): string;
```

C++

```
__fastcall UnicodeString GetParamInternalName(PTypeInfo Info, const UnicodeString MethodName,
const InvString ExternalParamName);
```

Description

GetParamInternalName returns the name of the method parameter that corresponds to a specified part. This is the parameter of a method on an invokable interface that is registered to correspond to a part that is defined in a WSDL document. The association between the parameter name and the part name is established by the RegisterExternalParamName (see page 4401) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodName is the name of the method on that invokable interface.

ExternalParamName is the name of a part on one of the messages in the operation that corresponds to MethodName. GetParamInternalName returns the corresponding parameter name.

See Also

RegisterExternalParamName (see page 4401)

GetParamExternalName (see page 4393)

38.1.7.4.21 TInvokableClassRegistry.GetRegInterfaceEntry Method

Returns the internal data that TInvokableClassRegistry (see page 4379) maintains for one of its registered interfaces.

Pascal

```
function GetRegInterfaceEntry(Index: Integer): InvRegIntfEntry;
```

C++

```
__fastcall InvRegIntfEntry GetRegInterfaceEntry(int Index);
```

Description

TInvokableClassRegistry (see page 4379) maintains an InvRegIntfEntry (see page 4478) value for every registered invokable interface. GetRegInterfaceEntry provides indexed access to this information.

The InvRegIntfEntry (see page 4478) value for a registered interface is intended for internal use by the invocation registry and the components that use it.

See Also

GetInfoForURI (see page 4388)

GetInterfaceCount (see page 4388)

RegisterDefaultSOAPAction (see page 4399)

38.1.7.4.22 TInvokableClassRegistry.GetRequestHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with requests to an invokable interface.

Pascal

```
function GetRequestHeaderInfoForInterface(Info: PTypeInfo): THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetRequestHeaderInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetRequestHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in messages that invoke a method on an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

GetRequestHeaderInfoForInterface returns an array of IntfHeaderItem (see page 4477) values, each of which describes a header that was registered for request messages on the invokable interface by a call to the RegisterHeaderClass (see page 4402) method.

See Also

RegisterHeaderClass (see page 4402)

TWSDLHTMLPublish

RegisterInterface (see page 4404)

GetHeaderInfoForInterface (see page 4386)

38.1.7.4.23 TInvokableClassRegistry.GetResponseHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.

Pascal

```
function GetResponseHeaderInfoForInterface(Info: PTypeInfo): THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetResponseHeaderInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetResponseHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in response messages that return the results of executing a method on an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

GetResponseHeaderInfoForInterface returns an array of IntfHeaderItem (see page 4477) values, each of which describes a header that was registered for response messages on the invokable interface by a call to the RegisterHeaderClass (see page 4402) method.

See Also

RegisterHeaderClass (see page 4402)

TWSDLHTMLPublish

RegisterInterface (see page 4404)

GetHeaderInfoForInterface (see page 4386)

38.1.7.4.24 TInvokableClassRegistry.GetReturnParamNames Method

Returns the list of parameter names that map to return values of interface methods.

Pascal

```
function GetReturnParamNames(const IntfInfo: PTypeInfo): string;
```

C++

```
__fastcall UnicodeString GetReturnParamNames(const PTypeInfo IntfInfo);
```

Description

GetReturnParamNames returns the list of parameter names that have been registered as mapping to return values of methods on a specified interface. Any out messages in a port type with these names are identified as the return value of a method on the corresponding interface. If the application does not explicitly register output parameter names by a call to RegisterReturnParamNames (see page 4406), GetReturnParamNames returns the default list, 'Result;Return'.

IntfInfo (see page 4331) is the type information pointer for the invokable interface for which you want the registered parameter names.

GetReturnParamNames returns the list of parameter names as a semicolon delimited list.

See Also

RegisterReturnParamNames (see page 4406)

38.1.7.4.25 GetUDDIInfo Method

This is the overview for the GetUDDIInfo method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetUDDIInfo (PTypeInfo, string, string) (see page 4396)	Retrieves the description of the UDDI location that describes a registered interface.
	TInvokableClassRegistry.GetUDDIInfo (TGUID, string, string) (see page 4396)	Retrieves the description of the UDDI location that describes a registered interface.

38.1.7.4.25.1 TInvokableClassRegistry.GetUDDIInfo Method (PTypeInfo, string, string)

Retrieves the description of the UDDI location that describes a registered interface.

Pascal

```
function GetUDDIInfo(const IntfInfo: PTypeInfo; var Operator: string; var BindingKey: string):
Boolean; overload;
function GetUDDIInfo(const AGUID: TGUID; var Operator: string; var BindingKey: string):
Boolean; overload;
```

C++

```
__fastcall Boolean GetUDDIInfo(const PTypeInfo IntfInfo, UnicodeString Operator, UnicodeString
BindingKey);
__fastcall Boolean GetUDDIInfo(const TGUID AGUID, UnicodeString Operator, UnicodeString
BindingKey);
```

Description

Typically, applications do not need to call GetUDDIInfo. This method is used internally to provide fail-over support when a registered interface was imported using the UDDI browser. It retrieves the information that identifies the UDDI entry for a Web Service, so that the application can re-fetch the target URL for a registered interface. The UDDI information is previously registered by a call to the RegisterUDDIInfo (see page 4407) method.

IntfInfo (see page 4331) is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

AGUID is a GUID that identifies the registered interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

Operator returns the URL of the UDDI registry that was used to import the interface definition.

BindingKey is the unique identifier for the TModel that describes the Web Service to which the interface belongs.

GetUDDIInfo returns true if it successfully locates and returns values for the Operator and BindingKey parameters. It returns false if there is no registered UDDI information for the specified interface.

See Also

RegisterUDDIInfo (see page 4407)

RegisterInterface (see page 4404)

38.1.7.4.26 GetWSDLEncoding Method

This is the overview for the GetWSDLEncoding method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo) (see page 4397)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo, string) (see page 4397)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo, string, string) (see page 4397)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.

38.1.7.4.26.1 TInvokableClassRegistry.GetWSDLEncoding Method (PTypeInfo)

Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.

Pascal

```
function GetWSDLEncoding(Info: PTypeInfo; const Namespace: string; const InternalIntfName:
string): InvString; overload;
function GetWSDLEncoding(Info: PTypeInfo; const Namespace: string): InvString; overload;
function GetWSDLEncoding(Info: PTypeInfo): InvString; overload;
```

C++

```
__fastcall InvString GetWSDLEncoding(PTypeInfo Info, const UnicodeString Namespace, const
UnicodeString InternalIntfName);
__fastcall InvString GetWSDLEncoding(PTypeInfo Info, const UnicodeString Namespace);
__fastcall InvString GetWSDLEncoding(PTypeInfo Info);
```

Description

GetWSDLEncoding returns the encoding attribute, if any, that is registered with the interface by a call to the RegisterInterface (see page 4404) method. The encoding attribute is an attribute of the xml header of a WSDL document, and describes the character set that should be used with all the elements that the document defines.

Info points to the type information of the registered interface.

Namespace specifies the namespace that was registered for the interface by the RegisterInterface (see page 4404) method. It can be an empty string.

InternalIntfName is the name of the interface (which may differ from the port type to which it corresponds).

See Also

RegisterInterface (see page 4404)

38.1.7.4.27 TInvokableClassRegistry.HasRegInterfaceImpl Method

Indicates whether there is a registered implementation class for an invokable interface.

Pascal

```
function HasRegInterfaceImpl(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean HasRegInterfaceImpl(int Index);
```

Description

Typically, applications do not need to call HasRegInterfaceImpl. This method is used internally to determine whether an implementation class has been registered for an invokable interface. The RegisterInvokableClass (see page 4405) method registers these implementation classes.

Index identifies the invokable interface for which an implementation class is sought. This index is the same as the index used in the GetRegInterfaceEntry (see page 4394) method, and reflects the order in which interfaces are registered.

HasRegInterfaceImpl returns true if there is a registered implementation class for the specified interface, false otherwise.

See Also

GetInterfaceCount (see page 4388)

GetRegInterfaceEntry (see page 4394)

RegisterInterface (see page 4404)

RegisterInvokableClass (see page 4405)

38.1.7.4.28 TInvokableClassRegistry.RegisterAllSOAPActions Method

Supplies a set of SOAP Action headers to use for an already-registered interface.

Pascal

```
procedure RegisterAllSOAPActions(Info: PTypeInfo; const AllSOAPActions: InvString);
```

C++

```
__fastcall RegisterAllSOAPActions(PTypeInfo Info, const InvString AllSOAPActions);
```

Description

RegisterDefaultSOAPAction (see page 4399) associates a registered invokable interface with a set of SOAP Action headers.

A default SOAP Action header is generated (based on the namespace) when you register the interface by a call to the RegisterInterface (see page 4404) method. RegisterAllSOAPActions lets you supply or change the SOAP Action header to a set of headers after the interface is already registered.

Info is the runtime type information (RTTI) for the interface for which you are supplying a set of SOAP Action headers.

AllSOAPActions is a string that includes the SOAP headers to use for calls to the interface that Info identifies. The individual header strings are delimited by a character that is given by the first character in the string.

See Also

RegisterInterface (see page 4404)

RegisterDefaultSOAPAction (see page 4399)

38.1.7.4.29 TInvokableClassRegistry.RegisterDefaultSOAPAction Method

Supplies a SOAP Action header to use for an already-registered interface.

Pascal

```
procedure RegisterDefaultSOAPAction(Info: PTypeInfo; const DefSOAPAction: InvString);
```

C++

```
__fastcall RegisterDefaultSOAPAction(PTypeInfo Info, const InvString DefSOAPAction);
```

Description

RegisterDefaultSOAPAction associates a registered invokable interface with a SOAP Action header. A default SOAP Action header is generated (based on the namespace) when you register the interface by a call to the RegisterInterface (see page 4404) method. RegisterDefaultSOAPAction lets you supply or change the SOAP Action header after the interface is already registered.

Info is the runtime type information (RTTI) for the interface for which you are supplying a default SOAP Action header.

DefSOAPAction is the SOAP Action header to use for calls to the interface that Info identifies. If this is an empty string, the server automatically generates a SOAP Action header based on the namespace in which the corresponding port type is defined.

See Also

RegisterInterface (see page 4404)

THTTPRIO

RegisterAllSOAPActions (see page 4398)

38.1.7.4.30 TInvokableClassRegistry.RegisterException Method

Registers the remotable exception classes that calls to an invokable interface can raise.

Pascal

```
procedure RegisterException(Info: PTypeInfo; AClass: TClass);
```

C++

```
__fastcall RegisterException(PTypeInfo Info, TClass AClass);
```

Description

Use RegisterException to register a remotable exception class. When an exception class is registered, its definition is exported to WSDL documents so that it can be used to communicate exception information between clients and servers.

Info points to the type information of the invokable interface that can raise the exception. If more than one interface can raise the

exception, you must call this method once for each interface type.

AClass describes the exception class. This must be the class reference for an ERemotableException (see page 4350) descendant.

Note: After a call to RegisterException, the exception class is registered so that it is associated with all methods of the specified interface. If only a subset of the interface's methods raise the exception, you can indicate this by calling the RegisterExceptionMethod (see page 4400) method for each method that can raise the exception.

See Also

RegisterInterface (see page 4404)

ERemotableException (see page 4350)

RegisterExceptionMethod (see page 4400)

Creating Custom Exception Classes for Web Services

38.1.7.4.31 TInvokableClassRegistry.RegisterExceptionMethod Method

Registers methods of an invokable interface that can raise a registered exception class.

Pascal

```
procedure RegisterExceptionMethod(Info: PTypeInfo; AClass: TClass; const MethodName: string);
```

C++

```
__fastcall RegisterExceptionMethod(PTypeInfo Info, TClass AClass, const UnicodeString
MethodName);
```

Description

Call RegisterExceptionMethod for each method of an invokable interface that can raise a registered exception class. The exception class must be previously registered by a call to the RegisterException (see page 4399) method. If you do not call RegisterExceptionMethod, then the WSDL document for the interface treats the exception as associated with all methods of the interface. If you want to limit the use of the exception to a subset of the methods, call RegisterExceptionMethod for each method that can raise the exception. After the first method is registered this way, the exception becomes limited to that method alone (and to any method subsequently registered by another call to RegisterExceptionMethod).

Info points to the type information of the invokable interface that can raise the exception.

AClass describes the exception class. This must be the class reference used to register the exception previously by a call to the RegisterException (see page 4399) method.

MethodName is the name of a method that can raise the exception.

See Also

RegisterInterface (see page 4404)

ERemotableException (see page 4350)

RegisterException (see page 4399)

38.1.7.4.32 TInvokableClassRegistry.RegisterExternalMethName Method

Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.

Pascal

```
procedure RegisterExternalMethName(Info: PTypeInfo; const InternalName: string; const
ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalMethName(PTypeInfo Info, const UnicodeString InternalName, const
InvString ExternalName);
```

Description

Use RegisterExternalMethName to create an association between the name of an operation in a WSDL document and the corresponding method of the invokable interface that corresponds to that operation's port type. This allows the invokable interface to represent a port type that includes operations with names that are not legal in Object Pascal (for example, keywords).

Info points to the type information of the invokable interface.

InternalName is the name used for the method of the invokable interface that corresponds to the operation.

ExternalName is the name of the operation that is defined in the WSDL document.

See Also

RegisterExternalParamName (see page 4401)

RegisterInterface (see page 4404)

38.1.7.4.33 TInvokableClassRegistry.RegisterExternalParamName Method

Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.

Pascal

```
procedure RegisterExternalParamName(Info: PTypeInfo; const MethodName: string; const
InternalName: string; const ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalParamName(PTypeInfo Info, const UnicodeString MethodName, const
UnicodeString InternalName, const InvString ExternalName);
```

Description

Use RegisterExternalParamName to create an association between the name of a part in a WSDL document and the corresponding parameter on a method of an invokable interface. This allows the invokable interface to handle definitions that have names that are not legal in Object Pascal (for example, keywords).

Info points to the type information of the invokable interface.

MethodName is the name of the method on the invokable interface.

InternalName is the name used for the parameter of the specified method that corresponds to the part with the name specified by ExternalName.

ExternalName is the name of the part that is defined in the WSDL document.

See Also

RegisterExternalMethName (see page 4400)

RegisterInterface (see page 4404)

38.1.7.4.34 RegisterHeaderClass Method

This is the overview for the RegisterHeaderClass method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterHeaderClass (PTypeInfo, TClass, eHeaderMethodType, Boolean) (see page 4402)	Registers a class that represents a headers which accompanies calls to an invokable interface.
	TInvokableClassRegistry.RegisterHeaderClass (PTypeInfo, TClass, InvString, InvString, eHeaderMethodType, Boolean) (see page 4402)	Registers a class that represents a headers which accompanies calls to an invokable interface.

38.1.7.4.34.1 TInvokableClassRegistry.RegisterHeaderClass Method (PTypeInfo, TClass, InvString, InvString, eHeaderMethodType, Boolean)

Registers a class that represents a headers which accompanies calls to an invokable interface.

Pascal

```
procedure RegisterHeaderClass(Info: PTypeInfo; AClass: TClass; const HeaderName: InvString;
const HeaderNamespace: InvString; DefaultMethodType: eHeaderMethodType = hmtAll; Required:
Boolean = False); overload;
procedure RegisterHeaderClass(Info: PTypeInfo; AClass: TClass; DefaultMethodType:
eHeaderMethodType = hmtAll; Required: Boolean = False); overload;
```

C++

```
__fastcall RegisterHeaderClass(PTypeInfo Info, TClass AClass, const InvString HeaderName,
const InvString HeaderNamespace, eHeaderMethodType DefaultMethodType = hmtAll, Boolean
Required = False);
__fastcall RegisterHeaderClass(PTypeInfo Info, TClass AClass, eHeaderMethodType
DefaultMethodType = hmtAll, Boolean Required = False);
```

Description

Use RegisterHeaderClass to register a TSOAPHeader (see page 4448) descendant. Registering a TSOAPHeader (see page 4448) descendant serves two purposes:

On servers, when a TSOAPHeader (see page 4448) descendant is registered, its definition is exported to WSDL documents so that applications using that WSDL document can recognize and process the header.

On both servers and clients, registering associates the TSOAPHeader (see page 4448) descendant with a header name and the namespace in which that name is defined.

Info points to the type information of the invokable interface that the header should accompany. If the header should be included in messages invoking more than one interface or in the responses to more than one interface, you must call this method once for each interface type.

AClass describes the TSOAPHeader (see page 4448) descendant that represents the header.

HeaderName is the name of the header, as defined within the header's namespace. If HeaderName is not supplied, the header's name is assumed to be the same as the type name under which it is registered in the remotable type registry.

HeaderNamespace is the URI for the namespace in which the header's name is defined. If HeaderName and HeaderNamespace are not supplied, the header is assumed to be defined in the same namespace as the interface with which it is associated.

DefaultMethodType indicates whether the header is to be used on request messages, response messages, or both. This default

value is associated with the interface as a whole. It can be changed for individual methods of the interface by subsequent calls to the RegisterHeaderMethod (see page 4403) method.

Required indicates whether the header must be included in messages about the interface. When Required is true, the header must be included and understood by the recipient of the message. When Required is false, the header is optional. Note that if the header is required by some but not all messages, you must set this Required parameter to false, and then add the requirement to individual methods by subsequent calls to the RegisterHeaderMethod (see page 4403) method.

Note: After a call to RegisterHeaderClass, the header class is registered so that it is associated with all methods of the specified interface. If only a subset of the interface's methods use the header, or if the header should have a different method type for some of the methods on the interface, or if the header is required by a subset of the methods on the interface, you can indicate this by calling the RegisterHeaderMethod (see page 4403) method for each method that the header accompanies.

See Also

RegisterInterface (see page 4404)

TSOAPHeader (see page 4448)

RegisterHeaderMethod (see page 4403)

38.1.7.4.35 TInvokableClassRegistry.RegisterHeaderMethod Method

Registers methods of an invokable interface that make use of a registered header.

Pascal

```
procedure RegisterHeaderMethod(Info: PTypeInfo; AClass: TClass; const MethodName: string;
MethodType: eHeaderMethodType = hmtAll; Required: Boolean = False); overload;
```

C++

```
__fastcall RegisterHeaderMethod(PTypeInfo Info, TClass AClass, const UnicodeString MethodName,
eHeaderMethodType MethodType = hmtAll, Boolean Required = False);
```

Description

Call RegisterHeaderMethod for each method of an invokable interface that uses a registered header type. The header class must be previously registered by a call to the RegisterHeaderClass (see page 4402) method. If you do not call RegisterHeaderMethod, then the WSDL document for the interface treats the header as associated with all methods of the interface. Use RegisterHeaderMethod if you want to

Limit the use of the header to a subset of the methods.

Specify whether the header is used for request messages, response messages, or both in a way that varies from method to method.

Indicate that the header is required for some methods but not others.

Call RegisterHeaderMethod for each method that uses the header. After the first method is registered this way, the header becomes limited to that method alone (and to any method subsequently registered by another call to RegisterHeaderMethod).

Info points to the type information of the invokable interface that the header should accompany.

AClass describes the TSOAPHeader (see page 4448) descendant that represents the header. This class must be previously registered by a call to RegisterHeaderClass (see page 4402).

MethodName is the name of a method that uses the header.

MethodType indicates whether the header is to be used on request messages, response messages, or both.

Required indicates whether the header must be included in messages about the specified method. When Required is true, the

header must be included and understood by the recipient of the message. When Required is false, the header is optional.

See Also

RegisterInterface (see page 4404)

TSOAPHeader (see page 4448)

RegisterHeaderClass (see page 4402)

38.1.7.4.36 TInvokableClassRegistry.RegisterInterface Method

Registers an invokable interface.

Pascal

```
procedure RegisterInterface(Info: PTypeInfo; const Namespace: InvString = ''; const
WSDLEncoding: InvString = ''; const Doc: string = ''; const ExtName: InvString = '');
```

C++

```
__fastcall RegisterInterface(PTypeInfo Info, const InvString Namespace = '', const InvString
WSDLEncoding = '', const UnicodeString Doc = '', const InvString ExtName = '');
```

Description

Call RegisterInterface to register an invokable interface with the invocation registry. Typically, the call to RegisterInterface is added by the WSDL importer or Web Service wizard. In Delphi, this call goes in the initialization section of the unit that defines the invokable interface. In C++, it is called within a RegTypes method that uses the pragma startup directive.

Once the invokable interface is registered, client applications can statically bind to it using THTTPRio. On the server side, you must also register the implementation class (using the RegisterInvokableClass (see page 4405) method) before the server can respond to incoming SOAP requests.

PInfo is a pointer to the type information for the interface. You can obtain the type information from an invokable interface by calling the TypeInfo function (Delphi) or __delphirtti (C++).

Namespace specifies a namespace URI that uniquely identifies the interface. If Namespace is set to an empty string, RegisterInterface automatically generates a unique string to use as a namespace URI. Note that RegisterInterface generates a SOAP Action header to use based on the Namespace of the interface. You can change this SOAP Action header using the RegisterDefaultSoapAction method.

WSDLEncoding specifies a encoding attribute that appears on the xml header of any WSDL document that includes the definition of this interface (or rather, the port type to which it corresponds). The encoding attribute describes the character set to use in the WSDL document.

Doc is a descriptive string that describes the purpose or use of the interface. It is used if the application publishes a WSDL document that defines this interface.

ExtName is the name of the port type that corresponds to this interface in a WSDL document. This can be an empty string if the port type and interface have the same name. It is useful when a port type in a WSDL document has a name that is not a valid Object Pascal identifier (for example, a keyword).

DefSOAPAction is the SOAP Action header that should be used for messages that represent calls to the methods on this interface. If you do not supply a default SOAP Action header, your application uses an automatically-generated header that is based on the namespace of the interface.

See Also

RegisterInvokableClass (see page 4405)

IInvokable

THTTPRio

UnRegisterInterface (see page 4407)

RegisterDefaultSOAPAction (see page 4399)

38.1.7.4.37 RegisterInvokableClass Method

This is the overview for the RegisterInvokableClass method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterInvokableClass (TClass) (see page 4405)	Registers the implementation class for an invokable interface.
	TInvokableClassRegistry.RegisterInvokableClass (TClass, TCreateInstanceProc) (see page 4405)	Registers the implementation class for an invokable interface.

38.1.7.4.37.1 TInvokableClassRegistry.RegisterInvokableClass Method (TClass)

Registers the implementation class for an invokable interface.

Pascal

```
procedure RegisterInvokableClass(AClass: TClass; CreateProc: TCreateInstanceProc); overload;
procedure RegisterInvokableClass(AClass: TClass); overload;
```

C++

```
__fastcall RegisterInvokableClass(TClass AClass, TCreateInstanceProc CreateProc);
__fastcall RegisterInvokableClass(TClass AClass);
```

Description

RegisterInvokableClass registers the class that implements an already-registered invokable interface. It is only needed on server applications that implement Web Services. The interface itself is registered by the RegisterInterface (see page 4404) method.

The invocation registry must be able to instantiate the implementation class. If the class has a virtual constructor that takes no arguments, you do not need to supply a method to create the class instance. If the constructor takes any arguments, of if it is not virtual, you must supply a method to instantiate the class as the value of the CreateProc argument.

AClass is a class reference to the class being registered. In Delphi code, you need only supply the class name. In C++, you can obtain a class reference for a class using the __classid operator.

CreateProc is a procedure that returns an instance of AClass. If CreateProc is nil (Delphi) or NULL (C++), then AClass must be a descendant of TInvokableClass (see page 4373).

See Also

RegisterInterface (see page 4404)

UnRegisterInvokableClass (see page 4408)

TInvokableClass (see page 4373)

38.1.7.4.38 RegisterInvokeOptions Method

This is the overview for the RegisterInvokeOptions method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterInvokeOptions (PTypeInfo, TIntfInvokeOption) (see page 4406)	Registers flags that customize the way calls to a registered interface are made.
	TInvokableClassRegistry.RegisterInvokeOptions (PTypeInfo, TIntfInvokeOptions) (see page 4406)	Registers flags that customize the way calls to a registered interface are made.

38.1.7.4.38.1 TInvokableClassRegistry.RegisterInvokeOptions Method (PTypeInfo, TIntfInvokeOption)

Registers flags that customize the way calls to a registered interface are made.

Pascal

```
procedure RegisterInvokeOptions(Info: PTypeInfo; const InvokeOption: TIntfInvokeOption);
overload;
procedure RegisterInvokeOptions(Info: PTypeInfo; const InvokeOptions: TIntfInvokeOptions);
overload;
```

C++

```
__fastcall RegisterInvokeOptions(PTypeInfo Info, const TIntfInvokeOption InvokeOption);
__fastcall RegisterInvokeOptions(PTypeInfo Info, const TIntfInvokeOptions InvokeOptions);
```

Description

RegisterInvokeOptions associates the specified invocation flag or flags with a registered interface. The interface must be previously registered by a call to the RegisterInterface (see page 4404) method. Invocation flags customize the way a registered interface is called, especially the way such calls are marshaled or unmarshaled.

Info identifies the registered interface using its runtime type information (RTTI). You can obtain the type information from an invokable interface by calling the TypeInfo function (delphi) or __delphirtti (C++).

InvokeOption specifies a single invocation option to associate with the interface.

InvokeOptions specifies a set of invocation options to associate with the interface.

Note: RegisterInvokeOptions adds the specified option or options to the set of invocation flags already registered for the interface. It does not remove any previously registered options.

See Also

GetIntfInvokeOptions (see page 4391)

Converter

Options (see page 6201)

RegisterInterface (see page 4404)

38.1.7.4.39 TInvokableClassRegistry.RegisterReturnParamNames Method

Registers a list of parameter names that map to return values of interface methods.

Pascal

```
procedure RegisterReturnParamNames(Info: PTypeInfo; const RetParamNames: string);
```

C++

```
__fastcall RegisterReturnParamNames(PTypeInfo Info, const UnicodeString RetParamNames);
```

Description

RegisterReturnParamNames registers a list of parameter names that correspond to the return value of methods on a specified interface. Typically, this method is called by the WSDL importer. It is important that these names correspond exactly to those that the WSDL importer uses if the interface was imported from a WSDL document.

Info is the type information pointer for the invokable interface whose method return values map to the names in the list.

RetParamNames is a semicolon delimited list of parameter names that map to return values of methods on the specified interface. If RetParamNames is an empty string, the methods of the interface never have a return value (only output parameters). If no return parameter names are registered for an interface, the default list, 'Result;Return', is used.

See Also

GetReturnParamNames (see page 4396)

38.1.7.4.40 TInvokableClassRegistry.RegisterUDDIInfo Method

Registers the location of the UDDI description that can be used to import the location of a Web Service.

Pascal

```
procedure RegisterUDDIInfo(Info: PTypeInfo; const Operator: String; const BindingKey: string);
```

C++

```
__fastcall RegisterUDDIInfo(PTypeInfo Info, const UnicodeString Operator, const UnicodeString
BindingKey);
```

Description

Typically, applications do not need to call RegisterUDDIInfo. This method is called internally when a registered interface is imported using the UDDI browser. It associates the location of the imported UDDI entry with a registered interface so that the application can later provide fail-over support by re-fetching the location of the Web Service.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 4404) method.

Operator is the URL of the UDDI registry that was used to import the interface definition.

BindingKey is the unique identifier for the TModel that describes the Web Service to which the interface belongs.

See Also

GetUDDIInfo (see page 4396)

RegisterInterface (see page 4404)

38.1.7.4.41 TInvokableClassRegistry.UnRegisterInterface Method

Unregisters an invokable interface.

Pascal

```
procedure UnRegisterInterface(Info: PTypeInfo);
```

C++

```
__fastcall UnRegisterInterface(PTypeInfo Info);
```

Description

Call UnRegisterInterface to unregister an invokable interface from the invocation registry. In Delphi, this call goes in the finalization section of the unit that defines the invokable interface. In C++, it is called from a method that uses the pragma exit directive.

Once an invokable interface is unregistered, client applications can no longer call it and server applications can't respond to calls directed at the interface methods.

Info is a pointer to the runtime type information for the interface. You can obtain the type information from an invokable interface by calling the TypeInfo function in Delphi or __delphirtti in C++.

See Also

UnRegisterInvokableClass (see page 4408)

IInvokable

RegisterInterface (see page 4404)

38.1.7.4.42 TInvokableClassRegistry.UnRegisterInvokableClass Method

Unregisters the implementation class for an invokable interface.

Pascal

```
procedure UnRegisterInvokableClass(AClass: TClass);
```

C++

```
__fastcall UnRegisterInvokableClass(TClass AClass);
```

Description

UnRegisterInvokableClass unregisters an implementation class that was previously registered by a call to RegisterInvokableClass (see page 4405). Once a class is unregistered, the invocation registry can no longer create and use instances of the specified class to handle incoming method calls to its corresponding invokable interface.

AClass is a class reference to the class being unregistered.

See Also

UnRegisterInterface (see page 4407)

RegisterInvokableClass (see page 4405)

38.1.8 TRemotable Class

TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.

Class Hierarchy

Pascal

```
TRemotable = class;
```

C++

```
class TRemotable;
```

File

InvokeRegistry

Description

Use TRemotable as a base class when defining a class that can be used for parameters or return values on an invokable interface. TRemotable descendants can represent non-scalar types, or scalar types that do not correspond to Object Pascal types. TRemotable is compiled with runtime type information (RTTI) and has a virtual constructor that the remotable class registry recognizes and uses to supply class instances.

Note: To create a class that represents scalar values that do not correspond directly to an Object Pascal type, use TRemotableXS (see page 4432) as a base class instead. TRemotableXS (see page 4432) is a descendant of the TRemotable class.

Note: When writing a TRemotable descendant, you can control whether a property is encoded as an element node or as an attribute node by using the stored directive. Properties declared with a stored directive set to AS_ATTRIBUTE (see page 4497) are encoded as attribute nodes. Properties that are declared without a stored directive, or with a stored directive set to anything other than AS_ATTRIBUTE (see page 4497) are encoded as element nodes.

Both the client application and server application must register a TRemotable descendant with the remotable class registry before they can use it. To register a remotable class, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 4472) function returns.

On server applications, instances of TRemotable descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TRemotable descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TRemotable instances that it passes as input parameters, and for freeing any TRemotable instances it creates or that are returned as output parameters or method results.

See Also

RemClassRegistry (see page 4471)

RemTypeRegistry (see page 4472)

TRemotableTypeRegistry (see page 4414)

IInvokable

Using Remotable Objects

38.1.8.1 TRemotable Members

The following tables list the members exposed by TRemotable.

Protected Fields

	Name	Description
	FSerializationOptions (see page 4411)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 4410)	Creates and initializes a TRemotable (see page 4408) object.

	Destroy (see page 4410)	Frees the memory associated with the TRemotable (see page 4408) object.
	ObjectToSOAP (see page 4411)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 4412)	Initializes this TRemotable (see page 4408) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	DataContext (see page 4413)	Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.
	SerializationOptions (see page 4413)	Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.

38.1.8.2 TRemotable.Create Constructor

Creates and initializes a TRemotable (see page 4408) object.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TRemotable();
```

Description

The TRemotable (see page 4408) constructor does nothing special. It exists primarily because

The remotable type registry knows how to call it.

It is virtual, so that the remotable type registry can also instantiate descendant classes.

See Also

RemTypeRegistry (see page 4472)

38.1.8.3 TRemotable.Destroy Destructor

Frees the memory associated with the TRemotable (see page 4408) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRemotable();
```

Description

Use the Free method to free a TRemotable (see page 4408) instance. Free checks that the object is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TRemotable (see page 4408) instance (for example, one created as a return value or output parameter). By default, when a TRemotable (see page 4408) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext (see page 4413) property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TRemotable (see page 4408) instances that are passed in as a parameter belong to the data context.

On client applications, TRemotable (see page 4408) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

System::TObject::Free

DataContext (see page 4413)

38.1.8.4 TRemotable Fields

The fields of the TRemotable class are listed here.

Protected Fields

	Name	Description
	FSerializationOptions (see page 4411)	This is FSerializationOptions, a member of class TRemotable.

38.1.8.4.1 TRemotable.FSerializationOptions Field

Pascal

```
FSerializationOptions: TSerializationOptions;
```

C++

```
TSerializationOptions FSerializationOptions;
```

Description

This is FSerializationOptions, a member of class TRemotable.

38.1.8.5 TRemotable Methods

The methods of the TRemotable class are listed here.

Public Methods

	Name	Description
	ObjectToSOAP (see page 4411)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 4412)	Initializes this TRemotable (see page 4408) instance to represent the value of its SOAP representation.

38.1.8.5.1 TRemotable.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotable (see page 4408) descendant to its SOAP representation. In TRemotable (see page 4408), ObjectToSOAP simply calls the ObjInstanceToSOAP method of the converter that is supplied by the ObjConverter parameter.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP (see page 4463)

SOAPToObject (see page 4412)

38.1.8.5.2 TRemotable.SOAPToObject Method

Initializes this TRemotable (see page 4408) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); virtual;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific remotable type from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotable (see page 4408), SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP (see page 4411)

InitObjectFromSOAP (see page 4463)

38.1.8.6 TRemotable Properties

The properties of the TRemotable class are listed here.

Public Properties

	Name	Description
	DataContext (see page 4413)	Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.
R	SerializationOptions (see page 4413)	Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.

38.1.8.6.1 TRemotable.DataContext Property

Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.

Pascal

```
property DataContext: TDataContext;
```

C++

```
__property TDataContext DataContext;
```

Description

By default, when a Web Service application creates an instance of a remotable object, that object exists only for the time it takes to marshal the information it represents or to unmarshal that information into the TRemotable (see page 4408) instance and then make a method call that uses the instance. The object specified by DataContext handles the freeing of the TRemotable (see page 4408) instance.

There are times when a server may not want the TRemotable (see page 4408) instance to be so transient. For example, if the object contains state information, it may be more efficient to have a single instance that is used for every message call. If you change DataContext, the TRemotable (see page 4408) instance is removed from the data context, and is not freed when the data context goes away at the end of the method call.

Note: If DataContext is nil (Delphi) or NULL (C++), the application is responsible for freeing the TRemotable (see page 4408) instance. For example, TRemotable (see page 4408) objects created on client applications are created with a nil (Delphi) or NULL (C++) value for DataContext and must be freed by the client application.

See Also

Using Remotable Objects

38.1.8.6.2 TRemotable.SerializationOptions Property

Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.

Pascal

```
property SerializationOptions: TSerializationOptions;
```

C++

```
__property TSerializationOptions SerializationOptions;
```

Description

SerializationOptions is used internally when an instance of TRemotable (see page 4408) is converted to or from its SOAP encoding. The value of this property is a set of flags that configure the mapping between the published properties of the TRemotable (see page 4408) descendant and the corresponding SOAP encoding.

Typically, the value of this read-only property is set by the constructor of a TRemotable (see page 4408) descendant or by the WSDL importer when it registers a remotable class.

See Also

SerializeOptions (see page 4427)

RegisterSerializeOptions (see page 4424)

38.1.9 TRemotableTypeRegistry Class

TRemotableTypeRegistry is the type of the remotable type registry.

Class Hierarchy

Pascal

```
TRemotableTypeRegistry = class;
```

C++

```
class TRemotableTypeRegistry;
```

File

InvokeRegistry

Description

TRemotableTypeRegistry manages information about remotable classes and types that can be communicated between client applications and a Web Service provider. Remotable classes can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when the Web Service application is trying to execute a request. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable type registry recognizes and uses to supply class instances. Remotable types include dynamic arrays, enumerated types, and boolean types, which can be used in an invokable interface without resorting to a remotable class representation, but which must be registered because their values require special marshaling.

Both the client application and server application must register remotable classes and types with the remotable type registry before they can use them. Use the RegisterXSClass (see page 4425) method to register a remotable class. Use the RegiterXSInfo method to register a dynamic array type or enumerated type.

Do not create instances of TRemotableTypeRegistry. Instead, use the global RemTypeRegistry (see page 4472) function to access the remotable type registry. Most applications need only call the RegisterXSClass (see page 4425) or RegisterXSInfo (see page 4426) methods. The other TRemotableTypeRegistry methods are used internally to create or look up information about remotable classes and types.

See Also

RemTypeRegistry (see page 4472)

TRemotable (see page 4408)

IInvokable

ERemotableException (see page 4350)

TInvokableClassRegistry (see page 4379)

38.1.9.1 TRemotableTypeRegistry Members

The following tables list the members exposed by TRemotableTypeRegistry.

Protected Methods

	Name	Description
	Lock (see page 4422)	Prevents other threads from accessing the type registry.
	UnLock (see page 4428)	Allows other threads to access the type registry after it was locked.

Public Methods

	Name	Description
	ClassOptions (see page 4417)	Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.
	ClassToURI (see page 4418)	This is the overview for the ClassToURI method overload.
	Create (see page 4415)	Creates and initializes a TRemotableTypeRegistry (see page 4414) object.
	Destroy (see page 4416)	Frees the memory associated with the TRemotableTypeRegistry (see page 4414) object.
	GetExternalPropName (see page 4418)	Returns the name of the XML element that corresponds to a property of a registered remotable class.
	GetInternalPropName (see page 4419)	Returns the name of the property on a remotable class that corresponds to a specified XML element.
	GetURICount (see page 4419)	Returns the number of entries in the remotable type registry.
	GetURIMap (see page 4420)	This is the overview for the GetURIMap method overload.
	GetVarTypeFromXSD (see page 4421)	Returns the variant type that corresponds to a registered remotable type.
	GetXSDInfoForClass (see page 4421)	Returns type name and namespace URI that correspond to a remotable class.
	InfoToURI (see page 4421)	Returns the namespace URI and type name for a registered type.
	IsClassScalar (see page 4422)	Indicates whether a registered class represents a scalar type.
	RegisterExternalPropName (see page 4423)	Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.
	RegisterHolderClsMember (see page 4423)	Registers a remotable type whose type information must be extracted from a holder class.
	RegisterSerializeOptions (see page 4424)	This is the overview for the RegisterSerializeOptions method overload.
	RegisterXSClass (see page 4425)	Registers a remotable class.
	RegisterXSInfo (see page 4426)	Registers a remotable type.
	SerializeOptions (see page 4426)	This is the overview for the SerializeOptions method overload.
	TypeInfoToXSD (see page 4427)	Returns the type name and namespace URI registered with a remotable class or type.
	UnRegisterXSClass (see page 4428)	Unregisters a remotable class.
	UnRegisterXSInfo (see page 4429)	Unregisters a remotable type.
	URIToClass (see page 4429)	This is the overview for the URIToClass method overload.
	URIToInfo (see page 4430)	Returns the type information pointer for a registered remotable class or type.
	VariantToInfo (see page 4430)	Returns the TypeInfo pointer for the type of a specified Variant's data.
	XSDToTypeInfo (see page 4431)	Returns the type information pointer associated with a specified type name.

Public Properties

	Name	Description
	AutoRegisterNativeTypes (see page 4431)	Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

38.1.9.2 TRemotableTypeRegistry.Create Constructor

Creates and initializes a TRemotableTypeRegistry (see page 4414) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TRemotableTypeRegistry();
```

Description

Do not create your own instances of TRemotableTypeRegistry (see page 4414). Instead, use the global RemTypeRegistry (see page 4472) function, which provides access to a single global instance of TRemotableTypeRegistry (see page 4414).

See Also

RemTypeRegistry (see page 4472)

38.1.9.3 TRemotableTypeRegistry.Destroy Destructor

Frees the memory associated with the TRemotableTypeRegistry (see page 4414) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRemotableTypeRegistry();
```

Description

Do not destroy the global TRemotableTypeRegistry (see page 4414) instance that is available through the RemClassRegistry (see page 4471) function. This object is freed automatically when the application unloads the InvokeRegistry (see page 4347) unit.

See Also

RemClassRegistry (see page 4471)

38.1.9.4 TRemotableTypeRegistry Methods

The methods of the TRemotableTypeRegistry class are listed here.

Protected Methods

	Name	Description
	Lock (see page 4422)	Prevents other threads from accessing the type registry.
	UnLock (see page 4428)	Allows other threads to access the type registry after it was locked.

Public Methods

	Name	Description
	ClassOptions (see page 4417)	Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.
	ClassToURI (see page 4418)	This is the overview for the ClassToURI method overload.
	GetExternalPropName (see page 4418)	Returns the name of the XML element that corresponds to a property of a registered remotable class.
	GetInternalPropName (see page 4419)	Returns the name of the property on a remotable class that corresponds to a specified XML element.
	GetURICount (see page 4419)	Returns the number of entries in the remotable type registry.
	GetURIMap (see page 4420)	This is the overview for the GetURIMap method overload.
	GetVarTypeFromXSD (see page 4421)	Returns the variant type that corresponds to a registered remotable type.
	GetXSDInfoForClass (see page 4421)	Returns type name and namespace URI that correspond to a remotable class.
	InfoToURI (see page 4421)	Returns the namespace URI and type name for a registered type.
	IsClassScalar (see page 4422)	Indicates whether a registered class represents a scalar type.
	RegisterExternalPropName (see page 4423)	Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.
	RegisterHolderClsMember (see page 4423)	Registers a remotable type whose type information must be extracted from a holder class.

	RegisterSerializeOptions (see page 4424)	This is the overview for the RegisterSerializeOptions method overload.
	RegisterXSClass (see page 4425)	Registers a remotable class.
	RegisterXSInfo (see page 4426)	Registers a remotable type.
	SerializeOptions (see page 4426)	This is the overview for the SerializeOptions method overload.
	TypeInfoToXSD (see page 4427)	Returns the type name and namespace URI registered with a remotable class or type.
	UnRegisterXSClass (see page 4428)	Unregisters a remotable class.
	UnRegisterXSInfo (see page 4429)	Unregisters a remotable type.
	URIToClass (see page 4429)	This is the overview for the URIToClass method overload.
	URIToInfo (see page 4430)	Returns the type information pointer for a registered remotable class or type.
	VariantToInfo (see page 4430)	Returns the TypeInfo pointer for the type of a specified Variant's data.
	XSDToTypeInfo (see page 4431)	Returns the type information pointer associated with a specified type name.

38.1.9.4.1 TRemotableTypeRegistry.ClassOptions Method

Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.

Pascal

```
function ClassOptions(AClass: TClass): TObjMultiOptions;
```

C++

```
__fastcall TObjMultiOptions ClassOptions(TClass AClass);
```

Description

When an application converts a remotable class to its SOAP representation, it can convert multiple references to the same object in one of two ways. It can treat the object as if its properties have meaning, but not the reference itself. (That is, two references to the same object are equivalent values, but not the same thing.) Alternately, if the object reference itself has meaning, it can encode the fact that these are multiple references to the same entity in the SOAP packet.

Individual remotable classes can specify how they want this conversion to be handled, overriding any defaults that would otherwise be used. For example, THTTPRIO, THTTPSoapPascalInvoker, and THTTPSoapCppInvoker use TOPToSoapDomConvert to convert between remotable classes and their SOAP representation. TOPToSoapDomConvert has an Options property that specifies, among other things, whether to encode multiple references to the same object by value or as references. Individual remotable classes can supply an override to this default strategy when they are registered using the RegisterXSClass (see page 4425) method.

ClassOptions returns how to handle multiple references of a specified class.

AClass is a class reference for the registered class that you want to know how to convert.

See Also

RegisterXSClass (see page 4425)

Options (see page 6201)

IOPConvert (see page 6152)

THTTPSoapCppInvoker

THTTPSoapPascalInvoker

THTTPRIO

TOPToSoapDomConvert (see page 6167)

38.1.9.4.2 ClassToURI Method

This is the overview for the ClassToURI method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.ClassToURI (TClass, InvString, InvString) (see page 4418)	Returns the namespace URI associated with a specified remotable class.
	TRemotableTypeRegistry.ClassToURI (TClass, InvString, InvString, Boolean, Boolean) (see page 4418)	Returns the namespace URI associated with a specified remotable class.

38.1.9.4.2.1 TRemotableTypeRegistry.ClassToURI Method (TClass, InvString, InvString)

Returns the namespace URI associated with a specified remotable class.

Pascal

```
function ClassToURI(AClass: TClass; var URI: InvString; var Name: InvString; var IsScalar:
Boolean; tryToRegister: Boolean = True): Boolean; overload;
function ClassToURI(AClass: TClass; var URI: InvString; var Name: InvString): Boolean;
overload;
```

C++

```
__fastcall Boolean ClassToURI(TClass AClass, InvString URI, InvString Name, Boolean IsScalar,
Boolean tryToRegister = True);
__fastcall Boolean ClassToURI(TClass AClass, InvString URI, InvString Name);
```

Description

ClassToURI looks up a registered remotable class and retrieves its namespace URI and typename. The association of the class and its namespace URI and typename is established when you call RegisterXSClass (see page 4425) to register the remotable class.

AClass is a class reference for the remotable class you want to look up.

URI returns the namespace URI that is associated with the class.

Name returns the registered name of the type the class represents.

IsScalar indicates whether the class corresponds to a scalar type in a WSDL document.

tryToRegister indicates whether the remotable type registry should auto-register the class that AClass specifies if it is not already registered. This parameter is only used if the AutoRegisterNativeTypes (see page 4431) property is true.

See Also

RegisterXSClass (see page 4425)

URIToClass (see page 4429)

AutoRegisterNativeTypes (see page 4431)

38.1.9.4.3 TRemotableTypeRegistry.GetExternalPropName Method

Returns the name of the XML element that corresponds to a property of a registered remotable class.

Pascal

```
function GetExternalPropName(Info: PTypeInfo; InternalName: string): InvString;
```

C++

```
__fastcall InvString GetExternalPropName(PTypeInfo Info, UnicodeString InternalName);
```

Description

Use GetExternalPropName to obtain the name of the XML element that was registered as the XML schema analog of a property on a registered remotable class. The association between the element and the property must be previously registered by the RegisterExternalPropname method.

Info points to the type information of the remotable class that has the property.

InternalName is the name of the property from which you want to obtain the corresponding XML element name.

See Also

GetInternalPropName (see page 4419)

RegisterExternalPropName (see page 4423)

38.1.9.4.4 TRemotableTypeRegistry.GetInternalPropName Method

Returns the name of the property on a remotable class that corresponds to a specified XML element.

Pascal

```
function GetInternalPropName(Info: PTypeInfo; ExternalName: InvString): string;
```

C++

```
__fastcall UnicodeString GetInternalPropName(PTypeInfo Info, InvString ExternalName);
```

Description

Use GetInternalPropName to obtain the name of the property on a registered remotable class that corresponds to a specified XML element. The association between the element and the property must be previously registered by the RegisterExternalPropname method.

Info points to the type information of the remotable class that has the property.

ExternalName is the name of an XML element that is part of a complex type defined in an XML schema. The remotable class identified by Info must represent the complex type that includes ExternalName.

See Also

GetExternalPropName (see page 4418)

RegisterExternalPropName (see page 4423)

38.1.9.4.5 TRemotableTypeRegistry.GetURICount Method

Returns the number of entries in the remotable type registry.

Pascal

```
function GetURICount: Integer;
```

C++

```
__fastcall int GetURICount();
```

Description

GetURICount returns the number of entries in the remotable type registry. This value can be used as an upper bound for the index parameter of the GetURIMap (see page 4420) method.

Each entry in the remotable type registry represents a registered class or a registered type. The RegisterXSClass (see page 4425) method adds entries for classes, and the The RegisterXSInfo (see page 4426) method adds entries for types. Each entry associates the class or type with a unique namespace URI.

See Also

GetURIMap (see page 4420)

RegisterXSClass (see page 4425)

RegisterXSInfo (see page 4426)

38.1.9.4.6 GetURIMap Method

This is the overview for the GetURIMap method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.GetURIMap (Integer) (see page 4420)	Returns an entry from the remotable type registry.
	TRemotableTypeRegistry.GetURIMap (Integer, TRemRegEntry) (see page 4420)	Returns an entry from the remotable type registry.

38.1.9.4.6.1 TRemotableTypeRegistry.GetURIMap Method (Integer)

Returns an entry from the remotable type registry.

Pascal

```
function GetURIMap(Index: Integer): TRemRegEntry; overload;
function GetURIMap(Index: Integer; out RegEntry: TRemRegEntry): boolean; overload;
```

C++

```
__fastcall TRemRegEntry GetURIMap(int Index);
__fastcall boolean GetURIMap(int Index, TRemRegEntry RegEntry);
```

Description

GetURIMap returns an entry from the remotable type registry that is identified by its index. It is useful for operations that iterate over all the registered entries. Use the GetURICount (see page 4419) method to obtain an upper bound for the index.

Index is the index of the entry you want to see.

Using the first syntax, GetURIMap returns a TRemRegEntry (see page 4485) value that describes a registered class or type.

Using the second syntax, the RegEntry parameter returns the TRemRegEntry (see page 4485) at the specified index. Using this syntax, GetURIMap returns true if Index is a valid index, false if Index is out of range.

The TRemRegEntry (see page 4485) value that this method returns is used to represent both remotable classes, which are added using the RegisterXSClass (see page 4425) method, and remotable types, which are added using the RegisterXSInfo (see page 4426) method.

See Also

GetURICount (see page 4419)

RegisterXSClass (see page 4425)

38.1.9.4.7 TRemotableTypeRegistry.GetVarTypeFromXSD Method

Returns the variant type that corresponds to a registered remotable type.

Pascal

```
function GetVarTypeFromXSD(const URI: InvString; const TypeName: InvString): TVarType;
```

C++

```
__fastcall TVarType GetVarTypeFromXSD(const InvString URI, const InvString TypeName);
```

Description

GetVarTypeFromXSD returns the built-in Variant type that corresponds most naturally to a remotable type that was registered by a call to the RegisterXSInfo (see page 4426) method.

URI is the namespace URI associated with the type when it was registered.

TypeName is the type name associated with the type when it was registered.

See Also

VariantToInfo (see page 4430)

XSDToTypeInfo (see page 4431)

URIToInfo (see page 4430)

RegisterXSInfo (see page 4426)

38.1.9.4.8 TRemotableTypeRegistry.GetXSDInfoForClass Method

Returns type name and namespace URI that correspond to a remotable class.

Pascal

```
procedure GetXSDInfoForClass(Info: PTypeInfo; var URI: InvString; var TypeName: InvString);
```

C++

```
__fastcall GetXSDInfoForClass(PTypeInfo Info, InvString URI, InvString TypeName);
```

Description

GetXSDInfoForClass returns the type name under which a remotable class is registered, along with the namespace URI in which that type name is defined. The class must be previously registered by a call to the RegisterXSClass (see page 4425) method.

Info is the type info pointer for the registered remotable class.

URI returns the namespace URI that was associated with the class when it was registered.

TypeName returns the type name that was associated with the class when it was registered.

See Also

RegisterXSClass (see page 4425)

XSDToTypeInfo (see page 4431)

38.1.9.4.9 TRemotableTypeRegistry.InfoToURI Method

Returns the namespace URI and type name for a registered type.

Pascal

```
function InfoToURI(Info: PTypeInfo; var URI: InvString; var Name: InvString; var IsScalar:
Boolean; tryToRegister: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean InfoToURI(PTypeInfo Info, InvString URI, InvString Name, Boolean IsScalar,
Boolean tryToRegister = True);
```

Description

InfoToURI returns the namespace URI and type name that is associated with the specified type description. This association is established when you register a type by calling the RegisterXSInfo (see page 4426) method.

Info is a pointer to the runtime type information (RTTI) for the registered type.

URI returns the namespace URI that is associated with the type.

Name returns the registered name of the type.

IsScalar indicates whether the type corresponds to a scalar type in a WSDL document.

tryToRegister indicates whether the remotable type registry should auto-register the type that Info specifies if it is not already registered. This parameter is only used if the AutoRegisterNativeTypes (see page 4431) property is true.

See Also

RegisterXSInfo (see page 4426)

AutoRegisterNativeTypes (see page 4431)

38.1.9.4.10 TRemotableTypeRegistry.IsClassScalar Method

Indicates whether a registered class represents a scalar type.

Pascal

```
function IsClassScalar(AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean IsClassScalar(TClass AClass);
```

Description

IsClassScalar returns true if the specified class represents a scalar type. Classes that represent scalar types descend from the TRemotableXS (see page 4432) class, and can convert their value to or from a string.

AClass identifes the class to check. It must be previously registered by a call to RegisterXSClass (see page 4425).

See Also

RegisterXSClass (see page 4425)

TRemotableXS (see page 4432)

38.1.9.4.11 TRemotableTypeRegistry.Lock Method

Prevents other threads from accessing the type registry.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

Lock uses a critical section to prevent simultaneous use of the type registry by multiple threads in the same application. Once one thread has called Lock to obtain access to the type registry, no other thread can return from the Lock method until the first thread has called the UnLock (see page 4428) method to release its lock.

Lock is called automatically by the methods of TRemotableTypeRegistry (see page 4414).

See Also

UnLock (see page 4428)

38.1.9.4.12 TRemotableTypeRegistry.RegisterExternalPropName Method

Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.

Pascal

```
procedure RegisterExternalPropName(Info: PTypeInfo; const InternalName: string; const
ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalPropName(PTypeInfo Info, const UnicodeString InternalName, const
InvString ExternalName);
```

Description

Use RegisterExternalPropName to create an association between the name of an element in an XML schema complex type and the corresponding property of the remotable class that represents that type. This allows the remotable class to represent complex types that include elements with names that are not valid (for example, keywords).

Info points to the type information of the remotable class that corresponds to a complex type defined in an XML schema.

InternalName is the name used for the property of the remotable class that corresponds to the element.

ExternalName is the name of the element that is part of the complex type that the XML schema defines.

38.1.9.4.13 TRemotableTypeRegistry.RegisterHolderClsMember Method

Registers a remotable type whose type information must be extracted from a holder class.

Pascal

```
procedure RegisterHolderClsMember(ClsTypeInfo: PTypeInfo; const URI: InvString = ''; const
Name: InvString = ''; const ExtName: InvString = '');
```

C++

```
__fastcall RegisterHolderClsMember(PTypeInfo ClsTypeInfo, const InvString URI = '', const
InvString Name = '', const InvString ExtName = '');
```

Description

Use RegisterHolderClsMember to register a type that is used as the type of the only published property of a holder class. This method is used primarily for C++ applications, which can contain types that require a holder class so that the compiler can extract their runtime type information (RTTI). If the holder class contains more than one published property, use the RegisterXSInfo (see page 4426) method after extracting the type information from the property using the global GetClsMemberTypeInfo (see page 4334) function.

Info points to the type information of the holder class that has a single published member whose type you want to register.

URI is the namespace URI to associate with the type that is registered. This should uniquely identify the type. If you omit this parameter or supply an empty string, RegisterXSInfo (see page 4426) generates a namespace URI for you, based on the name of the type, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 4496) variable. Note that the namespace URI must be the same in both client and server applications. In Delphi, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 4496) must be the same in both applications. In C++, the client application must always explicitly supply a namespace.

Name is the name of the type as it appears in native code.

ExtName is the name of the type as it appears in encoded method calls and WSDL documents.

See Also

GetClsMemberTypeInfo

RegisterXSInfo (see page 4426)

38.1.9.4.14 RegisterSerializeOptions Method

This is the overview for the RegisterSerializeOptions method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.RegisterSerializeOptions (PTypeInfo, TSerializationOptions) (see page 4424)	Registers a set of serialization options that should be used with a registered remotable class or type.
	TRemotableTypeRegistry.RegisterSerializeOptions (TClass, TSerializationOptions) (see page 4424)	Registers a set of serialization options that should be used with a registered remotable class or type.

38.1.9.4.14.1 TRemotableTypeRegistry.RegisterSerializeOptions Method (PTypeInfo, TSerializationOptions)

Registers a set of serialization options that should be used with a registered remotable class or type.

Pascal

```
procedure RegisterSerializeOptions(Info: PTypeInfo; SerialOpt: TSerializationOptions);
overload;
procedure RegisterSerializeOptions(AClass: TClass; SerialOpt: TSerializationOptions); overload;
```

C++

```
__fastcall RegisterSerializeOptions(PTypeInfo Info, TSerializationOptions SerialOpt);
__fastcall RegisterSerializeOptions(TClass AClass, TSerializationOptions SerialOpt);
```

Description

RegisterSerializeOptions associates a registered remotable class or type with a set of flags that indicate how to encode or decode its value. Typically, this method is called by the WSDL importer.

AClass identifes the class for which you are supplying serialization options. It must be previously registered by a call to RegisterXSClass (see page 4425).

Info is the runtime type information for a type when you are supplying serialization options for a type rather than a class. It must be previously registered by a call to RegisterXSInfo (see page 4426).

SerialOpt is the set of flags that customize how the class is encoded or decoded.

See Also

SerializationOptions (see page 4413)

SerializeOptions (see page 4427)

RegisterXSClass (see page 4425)

RegisterXSInfo (see page 4426)

Defining and Using SOAP Headers

38.1.9.4.15 TRemotableTypeRegistry.RegisterXSClass Method

Registers a remotable class.

Pascal

```
procedure RegisterXSClass(AClass: TClass; const URI: InvString = ''; const Name: InvString =
'', const ExtName: InvString = ''; IsScalar: Boolean = False; MultiRefOpt: TObjMultiOptions =
ocDefault);
```

C++

```
__fastcall RegisterXSClass(TClass AClass, const InvString URI = '', const InvString Name = '',
const InvString ExtName = '', Boolean IsScalar = False, TObjMultiOptions MultiRefOpt =
ocDefault);
```

Description

Call RegisterXSClass to register a remotable class so that it can be communicated between client applications and a Web Service provider. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable type registry recognizes and uses to supply class instances.

When the remotable class is a TRemotable (see page 4408) descendant, it can be used to represent non-scalar types for parameters and return values on an invokable interface. When it is an ERemotableException (see page 4350) descendant, it allows the Web Service application to send clients a fault packet that includes all the published properties of the exception class.

AClass is the class reference for the remotable class. In Delphi, you specify this in code using the class name. In C++, you can obtain this class reference for a class using the __classid keyword.

URI is the namespace URI to associate with that class. This should uniquely identify the class. If you omit this parameter (or supply an empty string), RegisterXSClass generates a namespace URI for you, based on the name of the class, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 4496) variable. Note that the namespace URI must be the same in both client and server applications. Thus, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 4496) must be the same in both applications.

Name is the name of a type that the class represents. This is used for the mapping between the class and its definition in a WSDL document.

ExtName is the same as Name.

IsScalar indicates whether the class represents a scalar type. Remotable classes that represent scalar types must implement the XSToNative and NativeToXS methods.

MultiRefOpt indicates how multiple references to the same instance of the class should be handled when converting them to their SOAP representation. For more information, see the ClassOptions (see page 4417) method.

See Also

TRemotable (see page 4408)

ERemotableException (see page 4350)

AppNameSPacePrefix

UnRegisterXSClass (see page 4428)

ClassOptions (see page 4417)

Creating Custom Exception Classes for Web Services

Registering Nonscalar Types

38.1.9.4.16 TRemotableTypeRegistry.RegisterXSInfo Method

Registers a remotable type.

Pascal

```
procedure RegisterXSInfo(Info: PTypeInfo; const URI: InvString = ''; const Name: InvString =
'''; const ExtName: InvString = '');
```

C++

```
__fastcall RegisterXSInfo(PTypeInfo Info, const InvString URI = '', const InvString Name = '',
const InvString ExtName = '');
```

Description

Call RegisterXSInfo to register a type that requires special marshaling but does not need to be represented by a TRemotable (see page 4408) descendant before it can be marshaled for transport to or from a Web Services provider. Such types include dynamic arrays such as the dynamic array types defined in the Types unit (Delphi) or sysdyn.h (C++), enumerated types, and boolean types (Delphi) or types defined using a typedef statement (C++).

Info is the TypeInfo pointer for the type. It can be obtained using the global TypeInfo function (Delphi) or the __delphirtti keyword (C++). In C++, it is sometimes obtained from a global function such as GetClsMemberTypeInfo (see page 4334), which extracts the type information from a holder class.

URI is the namespace URI to associate with that type. This should uniquely identify the type. If you omit this parameter or supply an empty string, RegisterXSInfo generates a namespace URI for you, based on the name of the type, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 4496) variable. Note that the namespace URI must be the same in both client and server applications. In Delphi, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 4496) must be the same in both applications. In C++, the client application must always explicitly supply a namespace.

Name is the name of the type as it appears in native code.

ExtName is the name of the type as it appears in encoded method calls and WSDL documents.

See Also

AppNameSPacePrefix

UnRegisterXSInfo (see page 4429)

GetClsMemberTypeInfo

Registering Nonscalar Types

38.1.9.4.17 SerializeOptions Method

This is the overview for the SerializeOptions method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.SerializeOptions (PTypeInfo) (see page 4427)	Returns the serialization options for a registered remotable class or type.
	TRemotableTypeRegistry.SerializeOptions (TClass) (see page 4427)	Returns the serialization options for a registered remotable class or type.

38.1.9.4.17.1 TRemotableTypeRegistry.SerializeOptions Method (PTypeInfo)

Returns the serialization options for a registered remotable class or type.

Pascal

```
function SerializeOptions(Info: PTypeInfo): TSerializationOptions; overload;
function SerializeOptions(AClass: TClass): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializeOptions(PTypeInfo Info);
__fastcall TSerializationOptions SerializeOptions(TClass AClass);
```

Description

SerializeOptions is used internally to retrieve the flags that describe how a remotable class or type is encoded and decoded. These flags are associated with the remotable class or type by calling the RegisterSerializeOptions (see page 4424) method.

AClass is the registered class for which you want to retrieve serialization options.

Info is the runtime type information for the registered type for which you want to retrieve serialization options.

SerializeOptions returns the value of SerializationOptions (see page 4481) for the class that AClass specifies or the registered options for the type that Info specifies..

See Also

SerializationOptions (see page 4413)

RegisterSerializeOptions (see page 4424)

38.1.9.4.18 TRemotableTypeRegistry.TypeInfoToXSD Method

Returns the type name and namespace URI registered with a remotable class or type.

Pascal

```
function TypeInfoToXSD(Info: PTypeInfo; var URI: InvString; var TypeName: InvString): Boolean;
```

C++

```
__fastcall Boolean TypeInfoToXSD(PTypeInfo Info, InvString URI, InvString TypeName);
```

Description

TypInfoToXSD returns a type name and namespace URI from a type information pointer. The type information can describe either a remotable class that was previously registered by a call to RegisterXSClass (see page 4425), or a remotable type that was previously registered by a call to RegisterXSInfo (see page 4426).

Info is the type information pointer that describes the class or type.

URI returns the namespace URI that was associated with the class or type when it was registered.

TypeName returns the type name that was associated with the class or type when it was registered.

See Also

XSDToTypeInfo (see page 4431)

RegisterXSInfo (see page 4426)

RegisterXSClass (see page 4425)

InfoToURI (see page 4421)

38.1.9.4.19 TRemotableTypeRegistry.UnLock Method

Allows other threads to access the type registry after it was locked.

Pascal

```
procedure UnLock; virtual;
```

C++

```
virtual __fastcall UnLock();
```

Description

UnLock exits the critical section that the Lock (see page 4422) method entered. It reenables other threads access to the type registry after a call to the Lock (see page 4422) method.

Every call to the Lock (see page 4422) method must be paired by a call to UnLock when the locking thread is finished with the type registry. The methods of TRemotableTypeRegistry (see page 4414) that call the Lock (see page 4422) method also call the UnLock method when they are finished.

See Also

Lock (see page 4422)

38.1.9.4.20 TRemotableTypeRegistry.UnRegisterXSClass Method

Unregisters a remotable class.

Pascal

```
procedure UnRegisterXSClass(AClass: TClass);
```

C++

```
__fastcall UnRegisterXSClass(TClass AClass);
```

Description

Call UnRegisterXSClass to Unregister a remotable class that was previously registered by a call to the RegisterXSClass (see page 4425) method. In Delphi, this call is made in the finalization section of the unit that implements the remotable class. In C++, it is made from a method that uses the pragma exit directive in the unit that implements the remotable class.

AClass is the class reference for the remotable class. In Delphi code, you specify this using the class name. In C++, you can obtain this class reference for a class using the __classid keyword.

See Also

TRemotable (see page 4408)

ERemotableException (see page 4350)

RegisterXSClass (see page 4425)

38.1.9.4.21 TRemotableTypeRegistry.UnRegisterXSInfo Method

Unregisters a remotable type.

Pascal

```
procedure UnRegisterXSInfo(Info: PTypeInfo);
```

C++

```
__fastcall UnRegisterXSInfo(PTypeInfo Info);
```

Description

Call UnRegisterXSInfo to unregister a type that was previously registered (directly or indirectly) by a call to the RegisterXSInfo (see page 4426) method.

Info is the TypeInfo pointer for the type. It is the same as the Info parameter of the RegisterXSInfo (see page 4426) method.

See Also

RegisterXSInfo (see page 4426)

38.1.9.4.22 URIToClass Method

This is the overview for the URIToClass method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.URIToClass (InvString, InvString) (see page 4429)	Returns the class associated with a specified namespace URI.
	TRemotableTypeRegistry.URIToClass (InvString, InvString, Boolean) (see page 4429)	Returns the class associated with a specified namespace URI.

38.1.9.4.22.1 TRemotableTypeRegistry.URIToClass Method (InvString, InvString)

Returns the class associated with a specified namespace URI.

Pascal

```
function URIToClass(const URI: InvString; const Name: InvString; var IsScalar: Boolean):
TClass; overload;
function URIToClass(const URI: InvString; const Name: InvString): TClass; overload;
```

C++

```
__fastcall TClass URIToClass(const InvString URI, const InvString Name, Boolean IsScalar);
__fastcall TClass URIToClass(const InvString URI, const InvString Name);
```

Description

URIToClass looks up a registered remotable class given its namespace URI and typename.

URI is the namespace URI associated with the class. This association is established by the RegisterXSClass (see page 4425) method.

Name is the name of the type that the class represents. If the class does not have a corresponding type name, Name should be an empty string.

IsScalar returns true if the class represents a scalar type, false otherwise.

URIToClass returns the class reference for the class identified by URI (and Name).

See Also

RegisterXSClass (see page 4425)

ClassToURI (see page 4418)

38.1.9.4.23 TRemotableTypeRegistry.URIToInfo Method

Returns the type information pointer for a registered remotable class or type.

Pascal

```
function URIToInfo(const URI: InvString; const Name: InvString): PTypeInfo;
```

C++

```
__fastcall PTypeInfo URIToInfo(const InvString URI, const InvString Name);
```

Description

URIToInfo looks up a registered remotable class or type given its namespace URI and typename. It returns the type information pointer that describes the type or class.

URI is the namespace URI associated with the class or type. This association is established by the RegisterXSClass (see page 4425) or RegisterXSInfo (see page 4426) method.

Name is the type name that was associated with the class or type when it was registered.

Tip: To look up the type information for a namespace and type that may possibly represent a built-in XML schema type, use the XSDToTypeInfo (see page 4431) method instead.

See Also

RegisterXSClass (see page 4425)

InfoToURI (see page 4421)

RegisterXSInfo (see page 4426)

38.1.9.4.24 TRemotableTypeRegistry.VariantToInfo Method

Returns the TypeInfo pointer for the type of a specified Variant's data.

Pascal

```
function VariantToInfo(const V: Variant; TryAllSchema: Boolean): PTypeInfo;
```

C++

```
__fastcall PTypeInfo VariantToInfo(const Variant V, Boolean TryAllSchema);
```

Description

VariantToInfo returns the TypeInfo pointer of the scalar type that represents the value of a Variant. If the Variant's value is not a scalar type that can be marshaled without using a registered object or type, VariantToInfo raises an ETypeRegistryException (see page 4353) exception.

V is the Variant for which you want the corresponding TypeInfo pointer.

TryAllSchema specifies whether VariantToInfo should check for corresponding scalar type definitions in the XML schema associated with this application, or whether it should check all the possible sets of XML schema types that are defined in the xmlschema unit.

See Also

VarType

ETypeRegistryException (see page 4353)

38.1.9.4.25 TRemotableTypeRegistry.XSDToTypeInfo Method

Returns the type information pointer associated with a specified type name.

Pascal

```
function XSDToTypeInfo(const URI: InvString; const TypeName: InvString): PTypeInfo;
```

C++

```
__fastcall PTypeInfo XSDToTypeInfo(const InvString URI, const InvString TypeName);
```

Description

XSDToTypeInfo looks up a namespace URI and typename, and returns the type information pointer that describes the corresponding Object Pascal type or class. Unlike the URIToInfo (see page 4430) method, XSDToTypeInfo can look up built-in type correspondences that do not need to be registered.

URI is the namespace URI for the type.

TypeName is the type name to look up.

See Also

RegisterXSClass (see page 4425)

URIToInfo (see page 4430)

RegisterXSInfo (see page 4426)

38.1.9.5 TRemotableTypeRegistry Properties

The properties of the TRemotableTypeRegistry class are listed here.

Public Properties

	Name	Description
	AutoRegisterNativeTypes (see page 4431)	Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

38.1.9.5.1 TRemotableTypeRegistry.AutoRegisterNativeTypes Property

Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

Pascal

```
property AutoRegisterNativeTypes: Boolean;
```

C++

```
__property Boolean AutoRegisterNativeTypes;
```

Description

When AutoRegisterNativeTypes is true (the default), type definitions are automatically registered when the invokable interface or remotable class that uses them is registered. That is, when you register an invokable interface with the invocation registry or when you register a remotable class with the remotable type registry, the registry checks whether it makes use of types that are

not yet registered, and if so, registers them as well. Note that you can still provide your own registration (for example, to control the external name or namespace) by registering the type before the call that registers the interface or remotable class.

See Also

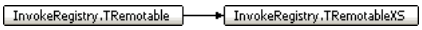
RegisterXSInfo (see page 4426)

RegisterXSClass (see page 4425)

38.1.10 TRemotableXS Class

TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.

Class Hierarchy



Pascal

```
TRemotableXS = class(TRemotable);
```

C++

```
class TRemotableXS : public TRemotable;
```

File

InvokeRegistry

Description

Use TRemotableXS as a base class when defining a class to represent scalar values from a WSDL document that have no corresponding Object Pascal type. When creating a TRemotableXS descendant, override the XSToNative (see page 4435) and NativeToXS (see page 4433) methods to indicate how the value of the class can be converted to and from the string representation that is used to encode the value for transport.

Both the client application and server application must register a TRemotableXS descendant with the remotable class registry before they can use it. To register a remotable class, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 4472) function returns.

On server applications, instances of TRemotableXS descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TRemotableXS descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable (see page 4408) descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TRemotableXS instances that it passes as input parameters, and for freeing any TRemotableXS instances it creates or that are returned as output parameters or method results.

See Also

RemTypeRegistry (see page 4472)

TRemotableTypeRegistry (see page 4414)

IInvokable

Using Remotable Objects

38.1.10.1 TRemotableXS Members

The following tables list the members exposed by TRemotableXS.

Protected Fields

	Name	Description
	FSerializationOptions (see page 4411)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 4410)	Creates and initializes a TRemotable (see page 4408) object.
	Destroy (see page 4410)	Frees the memory associated with the TRemotable (see page 4408) object.
	NativeToXS (see page 4433)	Returns a string that represents the value of a TRemotableXS (see page 4432) instance.
	ObjectToSOAP (see page 4434)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 4434)	Initializes this TRemotableXS (see page 4432) instance to represent the value of its SOAP representation.
	XSToNative (see page 4435)	Assigns the value of a string to the TRemotableXS (see page 4432) instance.

Public Properties

	Name	Description
	DataContext (see page 4413)	Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.
	SerializationOptions (see page 4413)	Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.

38.1.10.2 TRemotableXS Methods

The methods of the TRemotableXS class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 4433)	Returns a string that represents the value of a TRemotableXS (see page 4432) instance.
	ObjectToSOAP (see page 4434)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 4434)	Initializes this TRemotableXS (see page 4432) instance to represent the value of its SOAP representation.
	XSToNative (see page 4435)	Assigns the value of a string to the TRemotableXS (see page 4432) instance.

38.1.10.2.1 TRemotableXS.NativeToXS Method

Returns a string that represents the value of a TRemotableXS (see page 4432) instance.

Pascal

```
function NativeToXS: InvString; virtual; abstract;
```

C++

```
virtual __fastcall InvString NativeToXS() = 0;
```

Description

Override NativeToXS when implementing a TRemotableXS (see page 4432) descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS (see page 4432) instance. This string is used when marshaling the value of the TRemotableXS (see page 4432) descendant.

Classes that implement NativeToXS must also override the XSToNative (see page 4435) method so that it performs the converse transformation.

See Also

XSToNative (see page 4435)

Using Remotable Objects

38.1.10.2.2 TRemotableXS.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotableXS (see page 4432) descendant to its SOAP representation. In TRemotableXS (see page 4432), ObjectToSOAP uses the default conversion, replacing the text of the node with the value that the NativeToXS (see page 4433) method supplies.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

SOAPToObject (see page 4434)

NativeToXS (see page 4433)

38.1.10.2.3 TRemotableXS.SOAPToObject Method

Initializes this TRemotableXS (see page 4432) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
```

```
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a scalar value from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotableXS (see page 4432), SOAPToObject calls the XSToNative (see page 4435) method, passing in the text of the specified node.

See Also

ObjectToSOAP (see page 4434)

XSToNative (see page 4435)

38.1.10.2.4 TRemotableXS.XSToNative Method

Assigns the value of a string to the TRemotableXS (see page 4432) instance.

Pascal

```
procedure XSToNative(Data: InvString); virtual; abstract;
```

C++

```
virtual __fastcall XSToNative(InvString Data) = 0;
```

Description

Override XSToNative when implementing a TRemotableXS (see page 4432) descendant that represents a non-native scalar type. NativeToXS (see page 4433) interprets a string that represents the value of the TRemotableXS (see page 4432) instance and uses it to make any necessary changes so that the TRemotableXS (see page 4432) object reflects that value. It reverses the transformation that the NativeToXS (see page 4433) method performs.

Data is a string that represents the value of the TRemotableXS (see page 4432) instance.

See Also

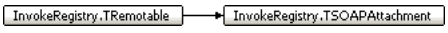
NativeToXS (see page 4433)

Using Remotable Objects

38.1.11 TSOAPAttachment Class

TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.

Class Hierarchy



Pascal

```
TSOAPAttachment = class(TRemotable);
```

C++

```
class TSOAPAttachment : public TRemotable;
```

File

InvokeRegistry

Description

Use TSOAPAttachment to receive or send an attachment as a parameter or return value of a method on an invokable interface. When a SOAP method call requires an attachment, it is marshaled as a MIME multipart form, where any attachments are included as additional parts.

Warning: Some Web Services (such as .Net) use DIME multipart forms rather than MIME multipart forms for attachments. TSOAPAttachment will not work with such services.

Note: Only use TSOAPAttachment and its descendants as the type for parameters or the return value of a method of an invokable interface. TSOAPAttachment can't be used as part of a more complex type, such as the type of a published property on a remotable object.

When sending an attachment, use the SourceString (see page 4448) property or the SetSourceFile (see page 4442) or SetSourceStream (see page 4443) method to supply the attachment data. Use the ContentType (see page 4445) property to indicate the type (encoding) of the attachment data.

When receiving an attachment, the attachment is automatically stored in a temporary file, whose name is given by the CacheFile (see page 4444) property. You can copy the contents of this temporary file using the SaveToFile (see page 4441) or SaveToStream (see page 4441) method. To determine the format of the data, read the ContentType (see page 4445) property. You can read any of the other headers for the part of the multipart form that contained the attachment using the Headers (see page 4446) property. By default, when the TSOAPAttachment object is freed, it deletes the temporary file that contains the attachment data that was received. You can use the CacheFilePersist (see page 4445) property to prevent this file from being deleted.

Note: If you use TSOAPAttachment as a base class for a customized attachment class, it is marshaled as a TSOAPAttachment value. That is, the application that receives the attachment sees it only as a TSOAPAttachment instance, not as an instance of the descendant class.

The application must register TSOAPAttachment with the remotable class registry before it can use this class. To register TSOAPAttachment or one of its descendants, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 4472) function returns. When you import a WSDL document, the call to register TSOAPAttachment is automatically added if any of the interfaces use attachments.

The caller of an invokable interface (client) is responsible for creating any TSOAPAttachment instances that it passes as input parameters, and for freeing any TSOAPAttachment instances it creates or that are returned as output parameters or method results.

In Web Service applications (servers), any TSOAPAttachment instances that represent parameters or return values are added to the data context, which frees them as necessary.

See Also

RemTypeRegistry (see page 4472)

TRemotableTypeRegistry (see page 4414)

IInvokable

Using Remotable Objects

38.1.11.1 TSOAPAttachment Members

The following tables list the members exposed by TSOAPAttachment.

Protected Fields

	Name	Description
	FSerializationOptions (see page 4411)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
V	GetSourceStream (see page 4439)	This is GetSourceStream, a member of class TSOAPAttachment.
V	InternalSetCacheFile (see page 4440)	This is InternalSetCacheFile, a member of class TSOAPAttachment.
	InternalSetSourceStream (see page 4440)	This is InternalSetSourceStream, a member of class TSOAPAttachment.
V	SetOwnership (see page 4442)	This is SetOwnership, a member of class TSOAPAttachment.
V	SetSourceString (see page 4443)	This is SetSourceString, a member of class TSOAPAttachment.

Public Methods

	Name	Description
V	Create (see page 4437)	Creates and initializes a TSOAPAttachment (see page 4435) object.
V	Destroy (see page 4438)	Frees the memory associated with the TSOAPAttachment (see page 4435) object.
	Init (see page 4439)	Initializes the properties of a TSOAPAttachment (see page 4435) instance.
V	ObjectToSOAP (see page 4440)	Converts this attachment object to its SOAP representation.
	SaveToFile (see page 4441)	Saves the contents of the attachment to a specified file.
	SaveToStream (see page 4441)	Saves the contents of the attachment to a specified file.
V	SetSourceFile (see page 4442)	Specifies a file from which TSOAPAttachment (see page 4435) can read the content of the attachment.
V	SetSourceStream (see page 4443)	Specifies a stream from which TSOAPAttachment (see page 4435) can read the content of the attachment.
V	SOAPToObject (see page 4443)	Initializes this TSOAPAttachment (see page 4435) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
R	CacheFile (see page 4444)	Identifies the temporary file where the attachment is saved.
	CacheFilePersist (see page 4445)	Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 4435) is freed.
	ContentType (see page 4445)	Specifies the MIME type of the attachment data.
	DataContext (see page 4413)	Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.
	Encoding (see page 4446)	Indicates the encoding (character set) of the body of the attachment.
R	Headers (see page 4446)	Indicates the values of the headers received with the attachment.
	Ownership (see page 4447)	Indicates whether the TSOAPAttachment (see page 4435) object should free its source stream.
R	SerializationOptions (see page 4413)	Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.
R	SourceStream (see page 4447)	Indicates a stream from which to read the content of an attachment that the application sends.
	SourceString (see page 4448)	Specifies a string that is sent as the value of the attachment.

38.1.11.2 TSOAPAttachment.Create Constructor

Creates and initializes a TSOAPAttachment (see page 4435) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TSOAPAttachment();
```

Description

Call Create to instantiate a TSOAPAttachment (see page 4435) object for an attachment you wish to send. When an application receives a SOAP attachment, an instance of TSOAPAttachment (see page 4435) is created automatically to represent that attachment.

38.1.11.3 TSOAPAttachment.Destroy Destructor

Frees the memory associated with the TSOAPAttachment (see page 4435) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPAttachment();
```

Description

Do not call Destroy to free a TSOAPAttachment (see page 4435) object. Instead, use the Free method, which checks for a nil reference before calling Destroy.

On server applications, there is typically no need to explicitly free a TSOAPAttachment (see page 4435) instance (for example, one created as a return value or output parameter). By default, when a TSOAPAttachment (see page 4435) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext (see page 4413) property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TSOAPAttachment (see page 4435) instances that are passed in as a parameter belong to the data context.

On client applications, TSOAPAttachment (see page 4435) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

DataContext (see page 4413)

System::TObject::Free

38.1.11.4 TSOAPAttachment Methods

The methods of the TSOAPAttachment class are listed here.

Protected Methods

	Name	Description
	GetSourceStream (see page 4439)	This is GetSourceStream, a member of class TSOAPAttachment.
	InternalSetCacheFile (see page 4440)	This is InternalSetCacheFile, a member of class TSOAPAttachment.
	InternalSetSourceStream (see page 4440)	This is InternalSetSourceStream, a member of class TSOAPAttachment.
	SetOwnership (see page 4442)	This is SetOwnership, a member of class TSOAPAttachment.
	SetSourceString (see page 4443)	This is SetSourceString, a member of class TSOAPAttachment.

Public Methods

	Name	Description
	Init (see page 4439)	Initializes the properties of a TSOAPAttachment (see page 4435) instance.
V	ObjectToSOAP (see page 4440)	Converts this attachment object to its SOAP representation.
	SaveToFile (see page 4441)	Saves the contents of the attachment to a specified file.
	SaveToStream (see page 4441)	Saves the contents of the attachment to a specified file.
V	SetSourceFile (see page 4442)	Specifies a file from which TSOAPAttachment (see page 4435) can read the content of the attachment.
V	SetSourceStream (see page 4443)	Specifies a stream from which TSOAPAttachment (see page 4435) can read the content of the attachment.
V	SOAPToObject (see page 4443)	Initializes this TSOAPAttachment (see page 4435) instance to represent the value of its SOAP representation.

38.1.11.4.1 TSOAPAttachment.GetSourceStream Method

Pascal

```
function GetSourceStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetSourceStream();
```

Description

This is GetSourceStream, a member of class TSOAPAttachment.

38.1.11.4.2 TSOAPAttachment.Init Method

Initializes the properties of a TSOAPAttachment (see page 4435) instance.

Pascal

```
procedure Init(ATempLocation: string; AHeaders: TStrings; AContentType: String; AEncoding:
InvString);
```

C++

```
__fastcall Init(UnicodeString ATempLocation, TStrings AHeaders, UnicodeString AContentType,
InvString AEncoding);
```

Description

Init sets the values of the CacheFile (see page 4444), Headers (see page 4446), ContentType (see page 4445), and Encoding (see page 4446) properties to the values supplied as parameters. It is used internally to initialize a TSOAPAttachment (see page 4435) instance to describe an attachment that the application received.

ATempLocation is the name of the temporary file where the attachment has been stored. It becomes the value of the CacheFile (see page 4444) property.

AHeaders lists the headers from the portion of the multipart form that represented this attachment. It becomes the value of the Headers (see page 4446) property.

AContentType is the content-type of the attachment. It becomes the value of the ContentType (see page 4445) property.

AEncoding is the character set encoding used for the attachment. It becomes the value of the Encoding (see page 4446) property.

See Also

CacheFile (see page 4444)

Headers (see page 4446)

ContentType (see page 4445)

Encoding (see page 4446)

38.1.11.4.3 TSOAPAttachment.InternalSetCacheFile Method

Pascal

```
procedure InternalSetCacheFile(const Value: string); virtual;
```

C++

```
virtual __fastcall InternalSetCacheFile(const UnicodeString Value);
```

Description

This is InternalSetCacheFile, a member of class TSOAPAttachment.

38.1.11.4.4 TSOAPAttachment.InternalSetSourceStream Method

Pascal

```
procedure InternalSetSourceStream(const Value: TStream; const Ownership: TStreamOwnership =
soReference);
```

C++

```
__fastcall InternalSetSourceStream(const TStream Value, const TStreamOwnership Ownership =
soReference);
```

Description

This is InternalSetSourceStream, a member of class TSOAPAttachment.

38.1.11.4.5 TSOAPAttachment.ObjectToSOAP Method

Converts this attachment object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjectToSOAP customizes the way an application converts a TSOAPAttachment (see page 4435) object to its SOAP representation. In TSOAPAttachment (see page 4435), ObjectToSOAP calls the inherited method, adding ocoDontSerializeProps and ocoDontPutTypeAttr to the ObjConvOpts flags so that the converter ignores the published properties and does not add type information to the generated node. It then generates a unique identifier for the attachment and calls the converter's AddAttachment method to properly generate the SOAP representation of the attachment.

RootNode is an ancestor of the generated node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated for the attachment.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP (see page 4463)

AddAttachment (see page 4462)

SOAPToObject (see page 4443)

38.1.11.4.6 TSOAPAttachment.SaveToFile Method

Saves the contents of the attachment to a specified file.

Pascal

```
procedure SaveToFile(AFileName: string);
```

C++

```
__fastcall SaveToFile(UnicodeString AFileName);
```

Description

Call SaveToFile to save the contents of the attachment to a specified file. This content can come from

The temporary file that the CacheFile (see page 4444) property indicates.

The stream that the SourceStream (see page 4447) property specifies.

The file that the SetSourceFile (see page 4442) method supplies.

The string that the SourceString (see page 4448) property specifies.

AFileName is the name of the file to which SaveToFile writes the content of the attachment.

See Also

CacheFile (see page 4444)

SourceStream (see page 4447)

SourceString (see page 4448)

SaveToStream (see page 4441)

38.1.11.4.7 TSOAPAttachment.SaveToStream Method

Saves the contents of the attachment to a specified file.

Pascal

```
procedure SaveToStream(AStream: TStream);
```

C++

```
__fastcall SaveToStream(TStream AStream);
```

Description

Call SaveToStream to write the contents of the attachment to a specified stream. This content can come from

The temporary file that the CacheFile (see page 4444) property indicates.

The stream that the SourceStream (see page 4447) property specifies.

The file that the SetSourceFile (see page 4442) method supplies.

The string that the SourceString (see page 4448) property specifies.

AStream is the stream to which SaveToStream writes the content of the attachment.

See Also

CacheFile (see page 4444)

SourceStream (see page 4447)

SourceString (see page 4448)

SaveToStream

SetSourceFile (see page 4442)

38.1.11.4.8 TSOAPAttachment.SetOwnership Method

Pascal

```
procedure SetOwnership(const Value: TStreamOwnership); virtual;
```

C++

```
virtual __fastcall SetOwnership(const TStreamOwnership Value);
```

Description

This is SetOwnership, a member of class TSOAPAttachment.

38.1.11.4.9 TSOAPAttachment.SetSourceFile Method

Specifies a file from which TSOAPAttachment (see page 4435) can read the content of the attachment.

Pascal

```
procedure SetSourceFile(const Value: string); virtual;
```

C++

```
virtual __fastcall SetSourceFile(const UnicodeString Value);
```

Description

Call SetSourceFile to supply the content of an attachment that the application sends from a file. SetSourceFile reads the content from the specified file, clears the SourceStream (see page 4447) and SourceString (see page 4448) properties, and sets the ContentType (see page 4445) property to 'Application/binary'. If another MIME type string better describes the data, reset the ContentType (see page 4445) property after calling SetSourceFile.

Value is the fully-qualified file name for the file whose content should be sent as an attachment.

See Also

ContentType (see page 4445)

SetSourceStream (see page 4443)

SourceString (see page 4448)

38.1.11.4.10 TSOAPAttachment.SetSourceStream Method

Specifies a stream from which TSOAPAttachment (see page 4435) can read the content of the attachment.

Pascal

```
procedure SetSourceStream(const Value: TStream; const Ownership: TStreamOwnership =
soReference); virtual;
```

C++

```
virtual __fastcall SetSourceStream(const TStream Value, const TStreamOwnership Ownership =
soReference);
```

Description

Call SetSourceStream to supply the content of an attachment that the application sends from a stream object. SetSourceStream assigns the specified stream as the value of the SourceStream (see page 4447) property, clears the FileName and SourceString (see page 4448) properties, and sets the ContentType (see page 4445) property to 'Application/binary'. If another MIME type string better describes the data, reset the ContentType (see page 4445) property after calling SetSourceStream.

Value is a stream from which TSOAPAttachment (see page 4435) can read the content of the attachment.

Ownership (see page 4447) indicates whether the TSOAPAttachment (see page 4435) object should free the stream (Value) when it no longer needs it. It becomes the value of the Ownership (see page 4447) property.

See Also

SourceStream (see page 4447)

ContentType (see page 4445)

Ownership (see page 4447)

SourceString (see page 4448)

SetSourceFile (see page 4442)

38.1.11.4.11 TSOAPAttachment.SetSourceString Method

Pascal

```
procedure SetSourceString(const Value: string); virtual;
```

C++

```
virtual __fastcall SetSourceString(const UnicodeString Value);
```

Description

This is SetSourceString, a member of class TSOAPAttachment.

38.1.11.4.12 TSOAPAttachment.SOAPToObject Method

Initializes this TSOAPAttachment (see page 4435) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. In TSOAPAttachment (see page 4435), this method calls the converter's FindAttachment method to locate information about the attachment, and then calls the Init (see page 4439) method to initialize this object from the located information.

RootNode is an ancestor node of the attachment node that can contain information about the attachment node.

Node is the node in the SOAP envelope that represents a reference to the attachment.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TSOAPAttachment (see page 4435), SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP (see page 4440)

FindAttachment (see page 4463)

Init (see page 4439)

38.1.11.5 TSOAPAttachment Properties

The properties of the TSOAPAttachment class are listed here.

Public Properties

	Name	Description
R	CacheFile (see page 4444)	Identifies the temporary file where the attachment is saved.
	CacheFilePersist (see page 4445)	Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 4435) is freed.
	ContentType (see page 4445)	Specifies the MIME type of the attachment data.
	Encoding (see page 4446)	Indicates the encoding (character set) of the body of the attachment.
R	Headers (see page 4446)	Indicates the values of the headers received with the attachment.
	Ownership (see page 4447)	Indicates whether the TSOAPAttachment (see page 4435) object should free its source stream.
R	SourceStream (see page 4447)	Indicates a stream from which to read the content of an attachment that the application sends.
	SourceString (see page 4448)	Specifies a string that is sent as the value of the attachment.

38.1.11.5.1 TSOAPAttachment.CacheFile Property

Identifies the temporary file where the attachment is saved.

Pascal

```
property CacheFile: string;
```

C++

```
__property UnicodeString CacheFile;
```

Description

CacheFile is set automatically when the application receives an attachment along with a SOAP message. This is the name of the

temporary file where the application stores the contents of the attachment.

To save this temporary file to another file of your choosing, use the SaveToFile (see page 4441) method. You can also save its contents to a stream, using the SaveToStream (see page 4441) method.

The ContentType (see page 4445) property indicates the format of the data in this temporary file.

By default, when the TSOAPAttachment (see page 4435) object is freed, it deletes the temporary file that CacheFile indicates. Use the CacheFilePersist (see page 4445) property to prevent this file from being deleted.

See Also

CacheFilePersist (see page 4445)

ContentType (see page 4445)

Data

SaveToFile (see page 4441)

SaveToStream (see page 4441)

38.1.11.5.2 TSOAPAttachment.CacheFilePersist Property

Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 4435) is freed.

Pascal

```
property CacheFilePersist: Boolean;
```

C++

```
__property Boolean CacheFilePersist;
```

Description

When CacheFilePersist is false (the default), the temporary file that the CacheFile (see page 4444) property identifies is automatically deleted when the TSOAPAttachment (see page 4435) instance is freed. You can save the information from the cache file using the SaveToFile (see page 4441) or SaveToStream (see page 4441) method before freeing the TSOAPAttachment (see page 4435) instance.

When CacheFilePersist is true, the temporary file is not deleted when the TSOAPAttachment (see page 4435) instance is freed.

See Also

CacheFile (see page 4444)

SaveToFile (see page 4441)

SaveToStream (see page 4441)

38.1.11.5.3 TSOAPAttachment.ContentType Property

Specifies the MIME type of the attachment data.

Pascal

```
property ContentType: string;
```

C++

```
__property UnicodeString ContentType;
```

Description

Use ContentType to determine the MIME type (format) used for the data in the attachment.

When receiving an attachment, the ContentType property is automatically set to reflect the value of the content-type header of the part of the multipart form that holds the attachment. You can read this property to determine how to interpret the contents of the file that the CacheFile (see page 4444) property indicates.

When sending an attachment, set ContentType to indicate the MIME type for the data you are sending. This property should not be set until after setting the SourceString (see page 4448) property calling the SetSourceFile (see page 4442) or SetSourceStream (see page 4443) method. That is because calling SetSourceStream (see page 4443) or SetSourceFile (see page 4442) automatically sets ContentType to 'Application/binary', and setting SourceString (see page 4448) automatically sets ContentType to 'Text/simple'. There is no need to reset ContentType if these values are correct.

See Also

CacheFile (see page 4444)

SourceString (see page 4448)

SetSourceFile (see page 4442)

SetSourceStream (see page 4443)

Data

38.1.11.5.4 TSOAPAttachment.Encoding Property

Indicates the encoding (character set) of the body of the attachment.

Pascal

```
property Encoding: InvString;
```

C++

```
__property InvString Encoding;
```

Description

Use Encoding to look up or set the encoding type of the attachment's content. When sending an attachment, set this property to indicate the encoding that was used for the source you supply.

If you supply the attachment data as a string by setting the SourceString (see page 4448) property, TSOAPAttachment (see page 4435) assumes an encoding of UTF-8 if you do not set the Encoding property. If you supply the attachment data by calling SetSourceFile (see page 4442) or SetSourceStream (see page 4443), you must set the Encoding property, as TSOAPAttachment (see page 4435) does not have a meaningful default encoding.

Note: Encoding indicates the encoding of the data that the SOAP attachment contains. The application does not automatically use this information to encode or decode that data.

See Also

ContentType (see page 4445)

38.1.11.5.5 TSOAPAttachment.Headers Property

Indicates the values of the headers received with the attachment.

Pascal

```
property Headers: TStrings;
```

C++

```
__property TStrings Headers;
```

Description

Read Headers to determine the headers that appeared in the part of the multipart form that contained the attachment. Headers is only intended for attachments that the application receives. When sending attachments, the application automatically generates the headers that accompany the attachment.

Headers is a string list where each string has the form HeaderName=Value. HeaderName is the name of the header, such as Content-Id, Content-Location, Content-Size, and so on. Value is the string that is the header value. Note that if you are only interested in the content type, you can use the ContentType (see page 4445) property instead.

Note: Although the MIME specification states that the Content-Length header is optional, TSoapAttachment requires this header and raises an exception if it is not present.

See Also

ContentType (see page 4445)

CacheFile (see page 4444)

38.1.11.5.6 TSOAPAttachment.Ownership Property

Indicates whether the TSOAPAttachment (see page 4435) object should free its source stream.

Pascal

```
property Ownership: TStreamOwnership;
```

C++

```
__property TStreamOwnership Ownership;
```

Description

Ownership indicates whether TSOAPAttachment (see page 4435) needs to free the stream that supplied the content of the attachment when the application supplied this data using the SetSourceStream (see page 4443) method.

See Also

SourceStream (see page 4447)

SetSourceStream (see page 4443)

38.1.11.5.7 TSOAPAttachment.SourceStream Property

Indicates a stream from which to read the content of an attachment that the application sends.

Pascal

```
property SourceStream: TStream;
```

C++

```
__property TStream SourceStream;
```

Description

Read SourceStream to access the content of an attachment that the application is sending. When SourceStream is not nil (Delphi) or NULL (C++), the content of the attachment is read from this stream and the SourceString (see page 4448) property is not set. Depending on the value of the Ownership (see page 4447) property, SourceStream can be owned by the TSOAPAttachment (see page 4435) object (in which case TSOAPAttachment (see page 4435) frees the stream object when

it is no longer needed) or can exist as an independent object that TSOAPAttachment (see page 4435) does not free.

SourceStream is a read-only property. To specify a stream from which TSOAPAttachment (see page 4435) reads its data, use the SetSourceStream (see page 4443) method.

See Also

SourceString (see page 4448)

SetSourceStream (see page 4443)

Ownership (see page 4447)

ContentType (see page 4445)

38.1.11.5.8 TSOAPAttachment.SourceString Property

Specifies a string that is sent as the value of the attachment.

Pascal

```
property SourceString: string;
```

C++

```
__property UnicodeString SourceString;
```

Description

Use SourceString to supply the content of an attachment as a string.

Setting the SourceString property causes TSOAPAttachment (see page 4435) to clear the SourceStream (see page 4447) property and to set the ContentType (see page 4445) property to 'Text/simple'. If another MIME type string better describes the data, reset the ContentType (see page 4445) property after setting SourceString.

See Also

Data

SourceStream (see page 4447)

SetSourceStream (see page 4443)

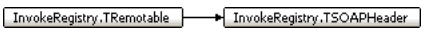
ContentType (see page 4445)

SetSourceFile (see page 4442)

38.1.12 TSOAPHeader Class

TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.

Class Hierarchy



Pascal

```
TSOAPHeader = class(TRemotable);
```

C++

```
class TSOAPHeader : public TRemotable;
```

File

InvokeRegistry

Description

TSOAPHeader is the base class for classes that represent SOAP headers. Each type of header has an associated TSOAPHeader descendant, which is registered with the remotable type registry. Register a TSOAPHeader descendant like any other remotable type, using the name of the header as the type name. If you import a WSDL document, the code to define and register the TSOAPHeader descendant is generated automtically.

You can also register the TSOAPHeader descendant with the invocation registry. The invocation registry's RegisterHeaderClass method associates the header class and the invokable interfaces with which it should be included. Registering the header with the invocation registry serves two purposes:

When a server publishes WSDL information about the Web Service, information about the header type is included in the resulting WSDL document.

On both servers and clients, the invocation registry associates the header class with a header name and a namespace in which that name is defined. This is important if the namespace for the header type is different than the namespace for the invokable interface that uses the header.

A SOAP header is a special remotable type that represents a node in the header section of a SOAP envelope. As with any other remotable type, the published properties of the TSOAPHeader descendant correspond to the child nodes and attributes of the corresponding header node. TSOAPHeader introduces two properties for common attributes of SOAP headers: MustUnderstand (see page 4452) and Actor (see page 4451). Typically, SOAP headers represent information that is associated with an entire invokable interface (port type) rather than just the method that is the focus of a single message.

When an application receives a SOAP message (that is, when a server receives a request to execute a method on an invokable interface or when a client receives a response after sending such a request), the headers on that message are automatically converted to the corresponding TSOAPHeader descendants that are registered with the remotable type registry. You can access these TSOAPHeader descendants using the ISOAPHeaders (see page 4464) interface, which can be obtained from the implementation class of the invokable interface or from the invokable interface of a remote interfaced object.

When an application sends a SOAP message (either a client sending a request or a server sending a response), supply the headers that should appear on that outgoing message by creating TSOAPHeader descendants and calling the Send method of an ISOAPHeaders (see page 4464) interface.

See Also

ISOAPHeaders (see page 4464)

TRemotableTypeRegistry (see page 4414)

TInvokableClassRegistry (see page 4379)

38.1.12.1 TSOAPHeader Members

The following tables list the members exposed by TSOAPHeader.

Protected Fields

	Name	Description
	FSerializationOptions (see page 4411)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 4410)	Creates and initializes a TRemotable (see page 4408) object.

	Destroy (see page 4410)	Frees the memory associated with the TRemotable (see page 4408) object.
	ObjectToSOAP (see page 4450)	Converts this TSOAPHeader (see page 4448) object to its SOAP representation.
	SOAPToObject (see page 4451)	Initializes this TSOAPHeader (see page 4448) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	Actor (see page 4451)	Identifies the agent to which the header is directed.
	DataContext (see page 4413)	Specifies the object that controls the lifetime of a TRemotable (see page 4408) instance.
	MustUnderstand (see page 4452)	Indicates whether an application is required to recognize and handle the header on received messages.
	SerializationOptions (see page 4413)	Indicates how the value of the TRemotable (see page 4408) descendant encodes its value.

38.1.12.2 TSOAPHeader Methods

The methods of the TSOAPHeader class are listed here.

Public Methods

	Name	Description
	ObjectToSOAP (see page 4450)	Converts this TSOAPHeader (see page 4448) object to its SOAP representation.
	SOAPToObject (see page 4451)	Initializes this TSOAPHeader (see page 4448) instance to represent the value of its SOAP representation.

38.1.12.2.1 TSOAPHeader.ObjectToSOAP Method

Converts this TSOAPHeader (see page 4448) object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TSOAPHeader (see page 4448) descendant to its SOAP representation. In TSOAPHeader (see page 4448), ObjectToSOAP uses the default conversion (supplied by the ObjConverter parameter), adding a MustUnderstand (see page 4452) attribute whose value reflects the value of the MustUnderstand (see page 4452) property.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

MustUnderstand (see page 4452)

ObjInstanceToSOAP (see page 4463)

SOAPToObject (see page 4451)

38.1.12.2.2 TSOAPHeader.SOAPToObject Method

Initializes this TSOAPHeader (see page 4448) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific type of header from its XML representation.

RootNode is an ancestor node of the header node that can contain information about the header node.

Node is the node in the SOAP envelope that represents the header.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

See Also

ObjectToSOAP (see page 4450)

InitObjectFromSOAP (see page 4463)

MustUnderstand (see page 4452)

38.1.12.3 TSOAPHeader Properties

The properties of the TSOAPHeader class are listed here.

Public Properties

	Name	Description
	Actor (see page 4451)	Identifies the agent to which the header is directed.
	MustUnderstand (see page 4452)	Indicates whether an application is required to recognize and handle the header on received messages.

38.1.12.3.1 TSOAPHeader.Actor Property

Identifies the agent to which the header is directed.

Pascal

```
property Actor: InvString;
```

C++

```
__property InvString Actor;
```

Description

Actor is the URI to which the header is directed. Set Actor to indicate the ultimate target of a header you are sending. Read Actor to determine whether a header the application receives is supposed to be handled by the application rather than merely passed on to the next processor.

Note: When the Actor property of a received header identifies this application and the application raises an exception concerning that header (such as a MustUnderstand (see page 4452) fault for a header that it does not know how to handle), the value of Actor should become the value of the FaultActor property on the ERemotableException (see page 4350) object.

See Also

MustUnderstand (see page 4452)

FaultActor (see page 4352)

38.1.12.3.2 TSOAPHeader.MustUnderstand Property

Indicates whether an application is required to recognize and handle the header on received messages.

Pascal

```
property MustUnderstand: Boolean;
```

C++

```
__property Boolean MustUnderstand;
```

Description

When MustUnderstand is false, applications can ignore this header if they do not recognize it and know how to handle it.

When MustUnderstand is true, applications that receive messages with this header but that do not recognize or handle it must throw a MustUnderstand fault.

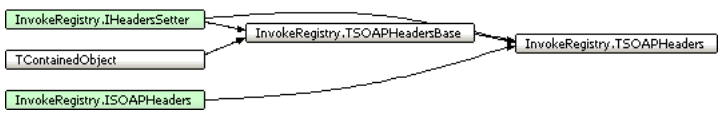
See Also

Actor (see page 4451)

38.1.13 TSOAPHeaders Class

TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 4464) interface.

Class Hierarchy



Pascal

```
TSOAPHeaders = class(TSOAPHeadersBase, ISOAPHeaders, IHeadersSetter);
```

C++

```
class TSOAPHeaders : public TSOAPHeadersBase, public ISOAPHeaders, public IHeadersSetter;
```

File

InvokeRegistry

Description

TSOAPHeaders implements the ISOAPHeaders (see page 4464) interface for a class that needs to provide access to the headers of SOAP messages that the application sends and receives Typically, an application does not work directly with instances of TSOAPHeaders. Rather, it uses the ISOAPHeaders (see page 4464) interface that this object implements for another class, such as TRIO or TInvokableClass (see page 4373).

See Also

ISOAPHeaders (see page 4464)

TSOAPHeader (see page 4448)

TRIO (see page 7408)

TInvokableClass (see page 4373)

38.1.13.1 TSOAPHeaders Members

The following tables list the members exposed by TSOAPHeaders.

Protected Fields

	Name	Description
	FHeadersInbound (see page 4458)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 4458)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

Protected Methods

	Name	Description
	SetHeadersInOut (see page 4458)	Sets the lists of incoming and outgoing headers.

Public Methods

	Name	Description
	Get (see page 4465)	This is the overview for the Get method overload.
	Get (see page 4454)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 4455)	Indicates whether the TSOAPHeaders (see page 4452) object frees headers that are added using the Send (see page 4455) method
	Send (see page 4455)	Adds a header to an outgoing message.
	SendAt (see page 4455)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 4456)	Indicates the number of headers that have been added to an outgoing message.
	SetHeadersInOut (see page 4461)	This is SetHeadersInOut, a member of class IHeadersSetter.
	SetOwnsSentHeaders (see page 4456)	Sets a flag that indicates who should free outgoing header objects.

Public Properties

	Name	Description
	OwnsSentHeaders (see page 4468)	Indicates who is responsible for freeing header objects that are included in outbound messages.

38.1.13.2 TSOAPHeaders Methods

The methods of the TSOAPHeaders class are listed here.

Public Methods

	Name	Description
	Get (see page 4454)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 4455)	Indicates whether the TSOAPHeaders (see page 4452) object frees headers that are added using the Send (see page 4455) method
	Send (see page 4455)	Adds a header to an outgoing message.
	SendAt (see page 4455)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 4456)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 4456)	Sets a flag that indicates who should free outgoing header objects.

38.1.13.2.1 Get Method

This is the overview for the Get method overload.

Overload List

	Name	Description
	TSOAPHeaders.Get (InvString, InvString) (see page 4454)	Retrieves a header from an incoming message.
	TSOAPHeaders.Get (TClass) (see page 4454)	Retrieves a header from an incoming message.
	TSOAPHeaders.Get (TClass, TSOAPHeader) (see page 4454)	Retrieves a header from an incoming message.

38.1.13.2.1.1 TSOAPHeaders.Get Method (InvString, InvString)

Retrieves a header from an incoming message.

Pascal

```
procedure Get(Cls: TClass; out Hdr: TSOAPHeader); overload;
function Get(Cls: TClass): TSOAPHeader; overload;
function Get(const Name: InvString; const URI: InvString): TSOAPHeader; overload;
```

C++

```
__fastcall Get(TClass Cls, TSOAPHeader Hdr);
__fastcall TSOAPHeader Get(TClass Cls);
__fastcall TSOAPHeader Get(const InvString Name, const InvString URI);
```

Description

Call Get to access the header object that is automatically created to represent one of the headers on an incoming messages. On the server, this is a header on the request message. On the client, this is a header on the response message.

Cls is the class reference that indicates the type of header class you want to retrieve. Each header class corresponds to a type of header, where the type name under which the class is registered corresponds to the name of the header node.

Hdr returns a TSOAPHeader (see page 4448) descendant that represents the header of the incoming message. If this syntax is not used, Get returns the header instance.

Name is the name of the header (the node name for the header node) that you want to retrieve.

URI is the URI for the namespace in which Name is defined.

See Also

Send (see page 4455)

DataContext (see page 4413)

38.1.13.2.2 TSOAPHeaders.GetOwnsSentHeaders Method

Indicates whether the TSOAPHeaders (see page 4452) object frees headers that are added using the Send (see page 4455) method

Pascal

```
function GetOwnsSentHeaders: Boolean;
```

C++

```
__fastcall Boolean GetOwnsSentHeaders();
```

Description

GetOwnsSentHeaders returns true if the TSOAPHeaders (see page 4452) instance frees any header objects added using the Send (see page 4455) method. It returns false if the caller is responsible for freeing these header objects.

See Also

Send (see page 4455)

38.1.13.2.3 TSOAPHeaders.Send Method

Adds a header to an outgoing message.

Pascal

```
procedure Send(const Hdr: TSOAPHeader);
```

C++

```
__fastcall Send(const TSOAPHeader Hdr);
```

Description

Call Send to add a header to an outgoing message. On the client, call Send before executing a method on an invokable interface. On the server, call Send before returning from the method that the client invoked.

Hdr is the header object instance that represents the header you want to add to the outgoing message.

Note: The SetOwnsSentHeaders (see page 4456) method controls whether you must free the object that is supplied as the Hdr parameter after sending the message.

See Also

SendAt (see page 4455)

SendCount (see page 4456)

SetOwnsSentHeaders (see page 4456)

38.1.13.2.4 TSOAPHeaders.SendAt Method

Retrieves a header from an outgoing message, given its position.

Pascal

```
function SendAt(Index: Integer): TSOAPHeader;
```

C++

```
__fastcall TSOAPHeader SendAt(int Index);
```

Description

Call SendAt to retrieve the header object for a header you have already added to an outgoing message using the Send (see page 4455) method.

Index is the position of the header you want to retrieve, where 0 specifies the first header, 1 specifies the second header, and so on. The SendCount (see page 4456) property provides an upper bound for this index.

SendAt returns the header at the specified position.

See Also

Send (see page 4455)

SendCount (see page 4456)

38.1.13.2.5 TSOAPHeaders.SendCount Method

Indicates the number of headers that have been added to an outgoing message.

Pascal

```
function SendCount: Integer;
```

C++

```
__fastcall int SendCount();
```

Description

Call SendCount to determine the number of headers that have been added to an outgoing message. These headers are added by previous calls to the Send (see page 4455) method.

See Also

Send (see page 4455)

SendAt (see page 4455)

38.1.13.2.6 TSOAPHeaders.SetOwnsSentHeaders Method

Sets a flag that indicates who should free outgoing header objects.

Pascal

```
procedure SetOwnsSentHeaders(Flag: Boolean);
```

C++

```
__fastcall SetOwnsSentHeaders(Boolean Flag);
```

Description

Call SetOwnsSentHeaders to specify who should free header objects that are added using the Send (see page 4455) method.

Flag indicates whether this TSOAPHeaders (see page 4452) frees the outgoing headers. If Flag is true, TSOAPHeaders (see page 4452) frees the headers that are added by calling Send (see page 4455). If Flag is false, the caller is responsible for freeing these header objects.

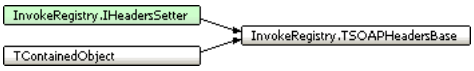
See Also

Send (see page 4455)

38.1.14 TSOAPHeadersBase Class

TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

Class Hierarchy



Pascal

```
TSOAPHeadersBase = class(TContainedObject, IHeadersSetter);
```

C++

```
class TSOAPHeadersBase : public TContainedObject, public IHeadersSetter;
```

File

InvokeRegistry

Description

TSOAPHeadersBase provides support for maintaining two lists of TSOAPHeader (see page 4448) descendants. One list is for the headers that an application sends and the other is for the headers that are it receives with an incoming message.

TSOAPHeadersBase is a descendant of TContainedObject. As such, it is designed to be used as a helper object in supporting an interface for working with SOAP headers. TSOAPHeadersBase is intended only as a base class. For working with SOAP headers, use a TSOAPHeadersBase descendant such as TSOAPHeaders (see page 4452).

See Also

TSOAPHeaders (see page 4452)

TSOAPHeader (see page 4448)

38.1.14.1 TSOAPHeadersBase Members

The following tables list the members exposed by TSOAPHeadersBase.

Protected Fields

	Name	Description
	FHeadersInbound (see page 4458)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 4458)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

Protected Methods

	Name	Description
	SetHeadersInOut (see page 4458)	Sets the lists of incoming and outgoing headers.

38.1.14.2 TSOAPHeadersBase Fields

The fields of the TSOAPHeadersBase class are listed here.

Protected Fields

	Name	Description
	FHeadersInbound (see page 4458)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 4458)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

38.1.14.2.1 TSOAPHeadersBase.FHeadersInbound Field

Pascal

```
FHeadersInbound: THeaderList;
```

C++

```
THeaderList FHeadersInbound;
```

Description

This is FHeadersInbound, a member of class TSOAPHeadersBase.

38.1.14.2.2 TSOAPHeadersBase.FHeadersOutBound Field

Pascal

```
FHeadersOutBound: THeaderList;
```

C++

```
THeaderList FHeadersOutBound;
```

Description

This is FHeadersOutBound, a member of class TSOAPHeadersBase.

38.1.14.3 TSOAPHeadersBase Methods

The methods of the TSOAPHeadersBase class are listed here.

Protected Methods

	Name	Description
	SetHeadersInOut (see page 4458)	Sets the lists of incoming and outgoing headers.

38.1.14.3.1 TSOAPHeadersBase.SetHeadersInOut Method

Sets the lists of incoming and outgoing headers.

Pascal

```
procedure SetHeadersInOut(var InHdrs: THeaderList; var OutHdrs: THeaderList);
```

C++

```
__fastcall SetHeadersInOut(THeaderList InHdrs, THeaderList OutHdrs);
```

Description

Applications can't call the protected SetHeadersInOut method. It is used by descendant classes to initialize the lists of incoming and outgoing header objects.

InHdrs is a list of TSoapHeader descendants that represent the headers received in a SOAP message.

OutHdrs is a list of TSoapHeader descendants that represent the headers sent in a SOAP message.

See Also

TSoapHeader

38.2 Interfaces

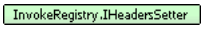
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHeadersSetter (see page 4460)	This is class InvokeRegistry.IHeadersSetter.
	IObjConverter (see page 4461)	IObjConverter is the interface for converting between a remotable class and its SOAP representation.
	ISOAPHeaders (see page 4464)	ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

38.2.1 IHeadersSetter Interface

Class Hierarchy



Pascal

```
IHeadersSetter = interface;
```

C++

```
__interface IHeadersSetter;
```

File

InvokeRegistry

Description

This is class InvokeRegistry.IHeadersSetter.

38.2.1.1 IHeadersSetter Members

The following tables list the members exposed by IHeadersSetter.

Public Methods

	Name	Description
	SetHeadersInOut (see page 4461)	This is SetHeadersInOut, a member of class IHeadersSetter.

38.2.1.2 IHeadersSetter Methods

The methods of the IHeadersSetter class are listed here.

Public Methods

	Name	Description
	SetHeadersInOut (see page 4461)	This is SetHeadersInOut, a member of class IHeadersSetter.

38.2.1.2.1 IHeadersSetter.SetHeadersInOut Method

Pascal

```
procedure SetHeadersInOut(var InHdrs: THeaderList; var OutHdrs: THeaderList);
```

C++

```
__fastcall SetHeadersInOut(THeaderList InHdrs, THeaderList OutHdrs);
```

Description

This is SetHeadersInOut, a member of class IHeadersSetter.

38.2.2 IObjConverter Interface

IObjConverter is the interface for converting between a remotable class and its SOAP representation.

Class Hierarchy

Pascal

```
IObjConverter = interface;
```

C++

```
__interface IObjConverter;
```

File

InvokeRegistry

Description

IObjConverter defines methods that aid in the conversion between remotable classes (classes that descend from TRemotable (see page 4408)) and their representation in the messages that pass between clients and Web Service providers. That is, this interface provides methods that are used when converting between a remotable class, whether it represents a complex type in an invokable interface, an attachment, or a SOAP header, and its SOAP representation. The TSOAPDomConv class implements this interface.

Note: C++ method declarations that use IObjConverter use the _di_IObjConverter type instead. This type is a DelphiInterface wrapper around the IObjConverter interface:

Note: typedef System::DelphiInterface< IObjConverter > _di_IObjConverter;

See Also

TSOAPDomCOnv

TRemotable (see page 4408)

TSoapHeader

TSoapAttachment

38.2.2.1 IObjConverter Members

The following tables list the members exposed by IObjConverter.

Public Methods

	Name	Description
	AddAttachment (see page 4462)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	FindAttachment (see page 4463)	Returns a TSOAPAttachment (see page 4435) instance that represent the attachment with a specified ID.
	InitObjectFromSOAP (see page 4463)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	ObjInstanceToSOAP (see page 4463)	Returns the SOAP representation of a remotable object.

38.2.2.2 IObjConverter Methods

The methods of the IObjConverter class are listed here.

Public Methods

	Name	Description
	AddAttachment (see page 4462)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	FindAttachment (see page 4463)	Returns a TSOAPAttachment (see page 4435) instance that represent the attachment with a specified ID.
	InitObjectFromSOAP (see page 4463)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	ObjInstanceToSOAP (see page 4463)	Returns the SOAP representation of a remotable object.

38.2.2.2.1 IObjConverter.AddAttachment Method

Stores information about a SOAP attachment so that it is included in the final message that is created.

Pascal

```
procedure AddAttachment(Attachment: TSOAPAttachment; const AContentId: string);
```

C++

```
__fastcall AddAttachment(TSOAPAttachment Attachment, const UnicodeString AContentId);
```

Description

AddAttachment stores a SOAP attachment so that it can be used to generate part of the multipart form that includes the SOAP envelope as another of its parts.

Attachment is the attachment that should be added to the multipart form.

AContentId is a string that uniquely identifies the attachment. It is added as a header to the part of the form that contains the attachment so that XML in the SOAP envelope can refer to the attachment. When interpreting the encoded SOAP envelope, this value is passed to the FindAttachment (see page 4463) method so that the information in the attachment part can be used to recreate the TSOAPAttachment (see page 4435) instance.

See Also

FindAttachment (see page 4463)

38.2.2.2.2 IObjConverter.FindAttachment Method

Returns a TSOAPAttachment (see page 4435) instance that represent the attachment with a specified ID.

Pascal

```
function FindAttachment(const AContentId: string): TSOAPAttachment;
```

C++

```
__fastcall TSOAPAttachment FindAttachment(const UnicodeString AContentId);
```

Description

FindAttachment locates the attachment that is identified by a specified content ID (HREF).

AContentId is the unique identifier for the attachment that was supplied to the AddAttachment (see page 4462) method when the attachment was first added to the multipart form.

FindAttachment returns a TSOAPAttachment (see page 4435) instance that represents the attachment.

See Also

AddAttachment (see page 4462)

38.2.2.2.3 IObjConverter.InitObjectFromSOAP Method

Initializes an instance of a remotable object to reflect the values in its SOAP representation.

Pascal

```
procedure InitObjectFromSOAP(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall InitObjectFromSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

InitObjectFromSOAP assigns values to the published properties of a remotable object that represents a node from the SOAP envelope.

Instance is the remotable object to initialize from its SOAP representation.

RootNode is an ancestor node of the node for the remotable object that can contain information needed for initializing the remotable object.

Node is the node in the SOAP envelope that represents the remotable object.

See Also

ObjInstanceToSOAP (see page 4463)

38.2.2.2.4 IObjConverter.ObjInstanceToSOAP Method

Returns the SOAP representation of a remotable object.

Pascal

```
function ObjInstanceToSOAP(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
NodeName: InvString; const NodeNamespace: InvString; ObjConvOpts: TObjectConvertOptions; out
RefID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode ObjInstanceToSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode
```

```
ParentNode, const InvString NodeName, const InvString NodeNamespace, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjInstanceToSOAP adds a node to represent the value of a remotable object.

Instance is the remotable object to encode as its SOAP representation.

RootNode is an ancestor of the node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjInstanceToSOAP generates.

NodeName is the name of the node that is generated.

NodeNamespace is the URI for the namespace in which NodeName is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjInstanceToSOAP returns the XML for the newly generated node.

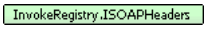
See Also

InitObjectFromSOAP (see page 4463)

38.2.3 ISOAPHeaders Interface

ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

Class Hierarchy



Pascal

```
ISOAPHeaders = interface;
```

C++

```
__interface ISOAPHeaders;
```

File

InvokeRegistry

Description

ISOAPHeaders is the interface for accessing the headers of a SOAP envelope. It defines methods for adding headers that are sent with an outgoing message and for retrieving the headers that were included in an incoming message.

On the server, the implementation class for in invokable interface supports this interface. You can obtain a reference to the ISOAPHeaders interface from the implementation class (a TInvokableClass (see page 4373) descendant). You can then use this interface to retrieve the headers that were included in an incoming request or to add headers to an outgoing response message. If your implementation class does not descend from TInvokableClass (see page 4373), you can obtain a reference to this interface using the global GetSOAPHeaders function.

For client-side header support, TRIO implements the ISOAPHeaders interface. However, you can't obtain an ISOAPHeaders interface directly from a remote interfaced object, because when you try to obtain an interface from TRIO or one of its descendants, it assumes that you are trying to obtain an invokable interface for a server object and generates an in-memory vtable. Instead, you must access the ISOAPHeaders interface from the invokable interface that the remote interface object supports.

Note: C++ method declarations that use ISOAPHeaders use the _di_ISOAPHeaders type instead. This type is a DelphiInterface wrapper around the ISOAPHeaders interface:

Note: typedef System::DelphiInterface< ISOAPHeaders > _di_ISOAPHeaders;

See Also

TRIO (see page 7408)

TInvokableClass (see page 4373)

TSOAPHeaders (see page 4452)

GetSOAPHeaders

Defining and Using SOAP Headers

Processing Headers in Client Applications

38.2.3.1 ISOAPHeaders Members

The following tables list the members exposed by ISOAPHeaders.

Public Methods

	Name	Description
	Get (see page 4465)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 4466)	Returns the value of the OwnsSentHeaders (see page 4468) property.
	Send (see page 4467)	Adds a header to an outgoing message.
	SendAt (see page 4467)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 4468)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 4468)	Sets the value of the OwnsSentHeaders (see page 4468) property.

Public Properties

	Name	Description
	OwnsSentHeaders (see page 4468)	Indicates who is responsible for freeing header objects that are included in outbound messages.

38.2.3.2 ISOAPHeaders Methods

The methods of the ISOAPHeaders class are listed here.

Public Methods

	Name	Description
	Get (see page 4465)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 4466)	Returns the value of the OwnsSentHeaders (see page 4468) property.
	Send (see page 4467)	Adds a header to an outgoing message.
	SendAt (see page 4467)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 4468)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 4468)	Sets the value of the OwnsSentHeaders (see page 4468) property.

38.2.3.2.1 Get Method

This is the overview for the Get method overload.

Overload List

	Name	Description
	ISOAPHeaders.Get (InvString, InvString) (see page 4466)	Retrieves a header from an incoming message.
	ISOAPHeaders.Get (TClass) (see page 4466)	Retrieves a header from an incoming message.
	ISOAPHeaders.Get (TClass, TSOAPHeader) (see page 4466)	Retrieves a header from an incoming message.

38.2.3.2.1.1 ISOAPHeaders.Get Method (InvString, InvString)

Retrieves a header from an incoming message.

Pascal

```
procedure Get(Cls: TClass; out Hdr: TSOAPHeader); overload;
function Get(Cls: TClass): TSOAPHeader; overload;
function Get(const Name: InvString; const URI: InvString): TSOAPHeader; overload;
```

C++

```
__fastcall Get(TClass Cls, TSOAPHeader Hdr);
__fastcall TSOAPHeader Get(TClass Cls);
__fastcall TSOAPHeader Get(const InvString Name, const InvString URI);
```

Description

Call Get to access the header object that is automatically created to represent one of the headers on an incoming messages. On the server, this is a header on the request message. On the client, this is a header on the response message.

Cls is the class reference that indicates the type of header class you want to retrieve. Each header class corresponds to a type of header, where the type name under which the class is registered corresponds to the name of the header node.

Hdr returns a TSOAPHeader (see page 4448) descendant that represents the header of the incoming message. If this syntax is not used, Get returns the header instance.

Name is the name of the header (the node name for the header node) that you want to retrieve.

URI is the URI for the namespace in which Name is defined.

See Also

Send (see page 4467)

DataContext (see page 4413)

38.2.3.2.2 ISOAPHeaders.GetOwnsSentHeaders Method

Returns the value of the OwnsSentHeaders (see page 4468) property.

Pascal

```
function GetOwnsSentHeaders: Boolean;
```

C++

```
__fastcall Boolean GetOwnsSentHeaders();
```

Description

GetOwnsSentHeaders is the read implementation for the OwnsSentHeaders (see page 4468) property.

See Also

OwnsSentHeaders (see page 4468)

38.2.3.2.3 ISOAPHeaders.Send Method

Adds a header to an outgoing message.

Pascal

```
procedure Send(const Hdr: TSOAPHeader);
```

C++

```
__fastcall Send(const TSOAPHeader Hdr);
```

Description

Call Send to add a header to an outgoing message. On the client, call Send before executing a method on an invokable interface. On the server, call Send before returning from the method that the client invoked.

Hdr is the header object instance that represents the header you want to add to the outgoing message.

Note: The OwnsSentHeaders (see page 4468) property controls whether you must free the object that is supplied as the Hdr parameter after sending the message.

See Also

SendAt (see page 4467)

SendCount (see page 4468)

OwnsSentHeaders (see page 4468)

38.2.3.2.4 ISOAPHeaders.SendAt Method

Retrieves a header from an outgoing message, given its position.

Pascal

```
function SendAt(Index: Integer): TSOAPHeader;
```

C++

```
__fastcall TSOAPHeader SendAt(int Index);
```

Description

Call SendAt to retrieve the header object for a header you have already added to an outgoing message using the Send (see page 4467) method.

Index is the position of the header you want to retrieve, where 0 specifies the first header, 1 specifies the second header, and so on. The SendCount (see page 4468) property provides an upper bound for this index.

SendAt returns the header at the specified position.

See Also

Send (see page 4467)

SendCount (see page 4468)

38.2.3.2.5 ISOAPHeaders.SendCount Method

Indicates the number of headers that have been added to an outgoing message.

Pascal

```
function SendCount: Integer;
```

C++

```
__fastcall int SendCount();
```

Description

Call SendCount to determine the number of headers that have been added to an outgoing message. These headers are added by previous calls to the Send (see page 4467) method.

See Also

Send (see page 4467)

SendAt (see page 4467)

38.2.3.2.6 ISOAPHeaders.SetOwnsSentHeaders Method

Sets the value of the OwnsSentHeaders (see page 4468) property.

Pascal

```
procedure SetOwnsSentHeaders(Flag: Boolean);
```

C++

```
__fastcall SetOwnsSentHeaders(Boolean Flag);
```

Description

SetOwnsSentHeaders is the write implementation for the OwnsSentHeaders (see page 4468) property.

Flag is the new value of the OwnsSentHeaders (see page 4468) property.

See Also

OwnsSentHeaders (see page 4468)

38.2.3.3 ISOAPHeaders Properties

The properties of the ISOAPHeaders class are listed here.

Public Properties

	Name	Description
	OwnsSentHeaders (see page 4468)	Indicates who is responsible for freeing header objects that are included in outbound messages.

38.2.3.3.1 ISOAPHeaders.OwnsSentHeaders Property

Indicates who is responsible for freeing header objects that are included in outbound messages.

Pascal

```
property OwnsSentHeaders: Boolean;
```

C++

```
__property Boolean OwnsSentHeaders;
```

Description

Use OwnsSentHeaders to indicate whether the header object instances you create and add to messages by calling the Send (see page 4467) or SendAt (see page 4467) method should be freed by the object that implements this interface. When OwnsSentHeaders is true, the object implementing ISOAPHeaders (see page 4464) frees the header objects after sending them. When OwnsSentHeaders is false, the caller is responsible for freeing the header objects.

Note: For inbound headers (headers retrieved from incoming messages), the object that implements always frees them unless you call the Get (see page 4466) method. After a call to Get (see page 4466), the caller is responsible for freeing the header, except on the server, where the header object is freed by the header object's DataContext.

See Also

Send (see page 4467)

SendAt (see page 4467)

DataContext (see page 4413)

38.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetRemotableDataContext (see page 4470)	Returns the TDataContext (see page 4354) object that provides temporary storage during a call to an invokable interface.
	InvRegistry (see page 4471)	Provides access to the invocation registry.
	RemClassRegistry (see page 4471)	Provides access to the remotable class registry.
	RemTypeRegistry (see page 4472)	Provides access to the remotable type registry.
	SetRemotableDataContext (see page 4473)	Sets the data context that provides temporary storage during a call to an invokable interface.
	SubstituteStrings (see page 4474)	This is function InvokeRegistry.SubstituteStrings.

38.3.1 InvokeRegistry.GetRemotableDataContext Function

Returns the TDataContext (see page 4354) object that provides temporary storage during a call to an invokable interface.

Pascal

```
function GetRemotableDataContext: Pointer;
```

C++

```
void * GetRemotableDataContext();
```

File

InvokeRegistry

Description

GetRemotableDataContext returns a pointer to the data context object that stores all the temporary objects and values needed for implementing a call to an invokable interface. When a client application calls an invokable interface on a Web Service application, the Web Service application automatically creates a data context for the call. Any objects or variables that are needed for marshaling the interface call are automatically added to the data context for the duration of the call. In addition, any remotable objects (TRemotable (see page 4408) descendants) that are created while there is a remotable data context available are automatically added to the data context. When the interface call is complete, the data context is freed, and it automatically frees the stored objects and variables.

See Also

SetRemotableDataContext (see page 4473)

TRemotable (see page 4408)

TDataContext (see page 4354)

38.3.2 InvokeRegistry.InvRegistry Function

Provides access to the invocation registry.

Pascal

```
function InvRegistry: TInvokableClassRegistry;
```

C++

```
TInvokableClassRegistry InvRegistry();
```

File

InvokeRegistry

Description

Call InvRegistry to access the invocation registry. The returned object has two methods, RegisterInterface and RegisterInvokableClass, that you can use for registering invokable interfaces (interfaces compiled with RTTI) and the classes that implement them.

Server applications that implement Web Services need to register both the invokable interfaces and the implementation classes. The invoker component can then use the invocation registry to locate invokable interfaces and identify classes that implement them.

Client applications that call on Web Services register the invokable interfaces that they call. The remote interfaced object component can then use the invocation registry to generate a dynamic method table for calling the invokable interfaces.

Note: Both client and server applications must register invokable interfaces using the same namespace. You can either supply this namespace explicitly when you register an interface, or you can use the automatically generated namespace. If you use the automatically generated interface, the interface must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 4496) variable.

See Also

IInvokable

THTTPSoapPascalInvoker

THTTPRio

AppNamespacePrefix

RemTypeRegistry (see page 4472)

38.3.3 InvokeRegistry.RemClassRegistry Function

Provides access to the remotable class registry.

Pascal

```
function RemClassRegistry: TRemotableClassRegistry;
```

C++

```
TRemotableClassRegistry RemClassRegistry();
```

File

InvokeRegistry

Description

RemClassRegistry returns the object that implements the remotable class registry. Use its RegisterXSClass method to register a remotable class in an application that publishes or uses Web Services.

Remotable classes are classes that can be transmitted between the client of a Web Services application and the server that implements the service. They can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when trying to execute a request to a Web Service application. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable class registry recognizes and uses to obtain class instances.

Both the client application and server application must register remotable classes before they can use them. They must register them to use the same namespace. You can either supply this namespace explicitly when you register the class, or you can use the automatically generated namespace. If you use the automatically generated interface, the class must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 4496) variable.

Note: The remotable class registry is the same thing as the remotable type registry. You can use RemClassRegistry interchangeably with the RemTypeRegistry (see page 4472) function.

See Also

TRemotable (see page 4408)

ERemotableException (see page 4350)

AppNameSpacePrefix (see page 4496)

InvRegistry (see page 4471)

RemTypeRegistry (see page 4472)

38.3.4 InvokeRegistry.RemTypeRegistry Function

Provides access to the remotable type registry.

Pascal

```
function RemTypeRegistry: TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry RemTypeRegistry();
```

File

InvokeRegistry

Description

Call RemTypeRegistry to access to the remotable type registry. Use the RegisterXSClass method of the returned object to register a remotable class in an application that publishes or uses Web Services. Use its RegisterXSInfo method to register a remotable type.

Remotable classes are classes that can be transmitted between the client of a Web Services application and the server that implements the service. They can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when trying to execute a request to a Web Service application. All remotable classes are compiled with

runtime type information (RTTI) and have a virtual constructor that the remotable class registry recognizes and uses to obtain class instances.

Remotable types include dynamic arrays, enumerated types, and boolean types, which can be used in an invokable interface without resorting to a remotable class representation, but which must be registered because their values require special marshaling.

Both the client application and server application must register remotable classes and types before they can use them. They must register them to use the same namespace. You can either supply this namespace explicitly when you register the class or type, or you can use the automatically generated namespace. If you use the automatically generated interface, the class must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 4496) variable.

Note: The remotable type registry is the same thing as the remotable class registry. You can use RemTypeRegistry interchangeably with the RemClassRegistry (see page 4471) function.

See Also

TRemotable (see page 4408)

ERemotableException (see page 4350)

AppNameSpacePrefix (see page 4496)

InvRegistry (see page 4471)

38.3.5 InvokeRegistry.SetRemotableDataContext Function

Sets the data context that provides temporary storage during a call to an invokable interface.

Pascal

```
procedure SetRemotableDataContext(Value: Pointer);
```

C++

```
SetRemotableDataContext(void * Value);
```

File

InvokeRegistry

Description

SetRemotableDataContext assigns a data context for a remote call to an invokable interface. Once a data context is assigned, it is available to store the temporary objects and values needed for implementing a call to an invokable interface. If the application creates a TRemotable (see page 4408) descendant while there is a remotable data context, that remotable object is automatically added to the data context and freed when the application frees the data context.

Note: The invoker component in a Web Services application automatically calls SetRemotableDataContext to set up a data context for incoming SOAP requests. Thus, there is typically no need to call this method in server applications.

Value is a pointer to a TDataContext (see page 4354) descendant that temporarily stores information for remote interface calls.

See Also

GetRemotableDataContext (see page 4470)

TRemotable (see page 4408)

TDataContext (see page 4354)

38.3.6 InvokeRegistry.SubstituteStrings Function

Pascal

```
function SubstituteStrings(const InputString: WideString; const SubString: WideString; const
Replacement: WideString): WideString;
```

C++

```
BSTR SubstituteStrings(const BSTR InputString, const BSTR SubString, const BSTR Replacement);
```

File

InvokeRegistry

Description

This is function InvokeRegistry.SubstituteStrings.

38.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	eHeaderMethodType (see page 4475)	eHeaderMethodType and THeaderMethodTypeArray (see page 4490) indicate the types of messages that should include a given header type.
	ObjectConvertOptions (see page 4480)	ObjectConvertOptions and TObjectConvertOptions (see page 4491) are flags that customize how to convert a remotable object to or from its SOAP representation.
	SerializationOptions (see page 4481)	SerializationOptions and TSerializationOptions (see page 4494) are flags that customize how to convert a remotable object to or from its SOAP representation.
	TIntfInvokeOption (see page 4483)	TIntfInvokeOption and TIntfInvokeOptions (see page 4490) represent flags that customize how to handle calls to an invokable interface.
	TObjMultiOptions (see page 4484)	TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.

Records

	Name	Description
	ExtNameMapItem (see page 4476)	This is record InvokeRegistry.ExtNameMapItem.
	IntfExceptionItem (see page 4476)	IntfExceptionItem and TExceptionItemArray (see page 4489) describe the association between a remotable exception and an interface that can raise it.
	IntfHeaderItem (see page 4477)	IntfHeaderItem and THeaderItemArray (see page 4489) describe the association between a header and the SOAP messages that include it.
	InvRegClassEntry (see page 4478)	This is record InvokeRegistry.InvRegClassEntry.
	InvRegIntfEntry (see page 4478)	InvRegIntfEntry stores information about an invokable interface.
	MethParamNameMapItem (see page 4480)	This is record InvokeRegistry.MethParamNameMapItem.
	TDynToClear (see page 4482)	This is record InvokeRegistry.TDynToClear.
	TRemHeaderEntry (see page 4485)	This is record InvokeRegistry.TRemHeaderEntry.
	TRemRegEntry (see page 4485)	TRemRegEntry stores information for the remotable type registry.

38.4.1 InvokeRegistry.eHeaderMethodType Enumeration

eHeaderMethodType and THeaderMethodTypeArray (see page 4490) indicate the types of messages that should include a given header type.

Pascal

```
eHeaderMethodType = (
  hmtAll,
  hmtRequest,
  hmtResponse
);
```

C++

```
enum eHeaderMethodType {
  hmtAll,
  hmtRequest,
  hmtResponse
};
```

File

InvokeRegistry

Description

THeaderMethodTypeArray (see page 4490) is a dynamic array of eHeaderMethodType values. The following table describes the possible eHeaderMethodType values:

Value	Meaning
hmtAll	The header should be included in all messages pertaining to the invokable interface.
hmtRequest	The header should be included in request messages in which the client attempts to invoke the invokable interface.
hmtResponse	The header should be included in response message that the server returns after executing a method on the invokable interface.

38.4.2 InvokeRegistry.ExtNameMapItem Record

Pascal

```
ExtNameMapItem = record
  Name: string;
  ExtName: InvString;
end;
```

C++

```
struct ExtNameMapItem {
  UnicodeString Name;
  InvString ExtName;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.ExtNameMapItem.

38.4.3 InvokeRegistry.IntfExceptionItem Record

IntfExceptionItem and TExceptionItemArray (see page 4489) describe the association between a remotable exception and an interface that can raise it.

Pascal

```
IntfExceptionItem = record
  ClassType: TClass;
  MethodNames: string;
end;
```

C++

```
struct IntfExceptionItem {
  TClass ClassType;
  UnicodeString MethodNames;
```

```
};
```

File

InvokeRegistry

Description

IntfExceptionItem describes a single remotable exception type. TExceptionItemArray (see page 4489) is an array of IntfExceptionItem descriptors. Typically, this array describes the set of exceptions associated with an invokable interface.

ClassType is the class reference for the remotable exception class.

MethodNames is a comma-delimited list of method names. If MethodNames is an empty string, the exception is associated with all the methods on an interface. If MethodNames is nonempty, the exception is associated only with those methods of the interface that the MethodNames field lists.

38.4.4 InvokeRegistry.IntfHeaderItem Record

IntfHeaderItem and THeaderItemArray (see page 4489) describe the association between a header and the SOAP messages that include it.

Pascal

```
IntfHeaderItem = record
  Info: PTypeInfo;
  ClassType: TClass;
  Namespace: InvString;
  Name: InvString;
  DefaultMethodType: eHeaderMethodType;
  DefaultRequired: Boolean;
  MethodNames: string;
  MethodTypes: THeaderMethodTypeArray;
  HeaderRequired: TRequiredArray;
end;
```

C++

```
struct IntfHeaderItem {
  PTypeInfo Info;
  TClass ClassType;
  InvString Namespace;
  InvString Name;
  eHeaderMethodType DefaultMethodType;
  Boolean DefaultRequired;
  UnicodeString MethodNames;
  THeaderMethodTypeArray MethodTypes;
  TRequiredArray HeaderRequired;
};
```

File

InvokeRegistry

Description

IntfHeaderItem describes a single header type. THeaderItemArray (see page 4489) is an array of IntfHeaderItem descriptors. Typically, this array describes the set of headers associated with an invokable interface.

Info is a pointer to the type description for the interface that the header is associated with.

ClassType is the class reference for the header class (a TSOAPHeader (see page 4448) descendant).

NameSpace is the namespace in which the header's name is defined.

Name is the name of the header node.

DefaultMethodType indicates whether the header is included in request messages, response messages, or both. This value is only used if the MethodNames field is an empty string. If MethodNames is nonempty, use the MethodTypes field instead.

DefaultRequired indicates whether the header must appear in messages about the interface that Info describes. If DefaultRequired is false, the header is optional. This value is only used if the MethodNames field is an empty string. If MethodNames is nonempty, use the HeaderRequired field instead.

MethodNames is a comma-delimited list of method names. If empty, the header is associated with all methods on the interface. If MethodNames includes one or more names, the header is only associated with those methods of the interface that are contained in the list.

MethodTypes is a dynamic array of values that indicate whether the header is included in request messages, response messages, or both. Each value is associated with a method listed in the MethodNames field in the same position. If MethodNames is an empty string, MethodTypes is not used.

HeaderRequired is a dynamic array of values that indicate whether the header is required or optional. Each value is associated with a method listed in the MethodNames field in the same position. If MethodNames is an empty string, HeaderRequired is not used.

38.4.5 InvokeRegistry.InvRegClassEntry Record

Pascal

```
InvRegClassEntry = record
  ClassType: TClass;
  Proc: TCreateInstanceProc;
  URI: string;
end;
```

C++

```
struct InvRegClassEntry {
  TClass ClassType;
  TCreateInstanceProc Proc;
  UnicodeString URI;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.InvRegClassEntry.

38.4.6 InvokeRegistry.InvRegIntfEntry Record

InvRegIntfEntry stores information about an invokable interface.

Pascal

```
InvRegIntfEntry = record
  Name: string;
  ExtName: InvString;
  UnitName: string;
  GUID: TGUID;
```

```
    Info: PTypeInfo;
    DefImpl: TClass;
    Namespace: InvString;
    WSDLEncoding: InvString;
    Documentation: string;
    SOAPAction: string;
    ReturnParamNames: string;
    InvokeOptions: TIntfInvokeOptions;
    MethNameMap: array of ExtNameMapItem;
    MethParamNameMap: array of MethParamNameMapItem;
    IntfHeaders: array of IntfHeaderItem;
    IntfExceptions: array of IntfExceptionItem;
    UDDIOperator: String;
    UDDIBindingKey: String;
  end;
```

C++

```
  struct InvRegIntfEntry {
    UnicodeString Name;
    InvString ExtName;
    UnicodeString UnitName;
    TGUID GUID;
    PTypeInfo Info;
    TClass DefImpl;
    InvString Namespace;
    InvString WSDLEncoding;
    UnicodeString Documentation;
    UnicodeString SOAPAction;
    UnicodeString ReturnParamNames;
    TIntfInvokeOptions InvokeOptions;
    array of ExtNameMapItem MethNameMap;
    array of MethParamNameMapItem MethParamNameMap;
    array of IntfHeaderItem IntfHeaders;
    array of IntfExceptionItem IntfExceptions;
    UnicodeString UDDIOperator;
    UnicodeString UDDIBindingKey;
  };
```

File

InvokeRegistry

Description

InvRegIntfEntry is used by the invocation registry to store information about a registered invokable interface.

Name is the name of the registered invokable interface.

ExtName is the name of the corresponding port type that appears in the WSDL document describing this Web Service. ExtName can be an empty string if the interface and port type have the same name.

UnitName is the name of the unit in which the interface is defined.

GUID is the globally unique identifier for the interface.

Info points to the runtime type information (RTTI) compiled with the interface.

DefImpl is the default implementation class for the interface.

Namespace is the namespace that is uniquely associated with the interface.

WSDLEncoding is the encoding attribute, if any, of the xml header on a WSDL document that defines the port type to which the invokable interface corresponds.

Documentation is a brief description that documents the interface.

DefaultSOAPAction is the SOAP Action header (or set of SOAP Action headers) to use on messages directed to this interface. If

this is an empty string, the application generates a SOAP Action header that is based on the Namespace field. Assign SOAP Action headers using the RegisterDefaultSOAPAction or RegisterAllSOAPActions method. Note that if you have more than one interface with the same namespace URI, you must register them with different SOAP Action headers so that the server can distinguish them.

ReturnParamNames is a semicolon-delimited list of parameter names that map to the return value of a function.

InvokeOptions are the options for the interface that have been registered using the RegisterInvokeOptions method. These options customize the way the interface is called.

MethNameMap is an array of entries that describe the relationship between methods on the invokable interface and the operation names to which they correspond (if different).

MethParamNameMap is an array of entries that describe the relationship between parameters on a method of the invokable interface and the part names to which they correspond (if different).

IntfHeaders is an array of entries that describe the SOAP headers that should accompany requests directed at the invokable interface.

IntfExceptions is an array of entries that describe the exceptions that the interface can raise.

UDDIOperator is the URL of the UDDI registry, if any, that was used to import the interface definition.

UDDIBindingKey is the unique identifier for the TModel from UDDIOperator that describes the Web Service to which the interface belongs.

38.4.7 InvokeRegistry.MethParamNameMapItem Record

Pascal

```
MethParamNameMapItem = record
  MethName: string;
  ParamNameMap: array of ExtNameMapItem;
end;
```

C++

```
struct MethParamNameMapItem {
  UnicodeString MethName;
  array of ExtNameMapItem ParamNameMap;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.MethParamNameMapItem.

38.4.8 InvokeRegistry.ObjectConvertOptions Enumeration

ObjectConvertOptions and TObjectConvertOptions (see page 4491) are flags that customize how to convert a remotable object to or from its SOAP representation.

Pascal

```
ObjectConvertOptions = (
```

```
  ocoDontPrefixNode,
  ocoDontSerializeProps,
  ocoDontPutTypeAttr
);
```

C++

```
enum ObjectConvertOptions {
  ocoDontPrefixNode,
  ocoDontSerializeProps,
  ocoDontPutTypeAttr
};
```

File

InvokeRegistry

Description

TObjectConvertOptions (see page 4491) is a set of ObjectConvertOptions values. ObjectConvertOptions values are flags that can be supplied when customizing the conversion between an object and its SOAP representation.

Value	Meaning
ocoDontPrefixNode	When set, the SOAP representation does not include a namespace prefix on node names.
ocoDontSerializeProps	When set, the SOAP representation does not include any representation for the published properties of the remotable object.
ocoDontPutTypeAttr	When set, nodes do not include an xsi:type attribute that indicates their type.

38.4.9 InvokeRegistry.SerializationOptions Enumeration

SerializationOptions and TSerializationOptions (see page 4494) are flags that customize how to convert a remotable object to or from its SOAP representation.

Pascal

```
SerializationOptions = (
  xoHolderClass,
  xoAttributeOnLastMember,
  xoInlineArrays,
  xoLiteralParam,
  xoSimpleTypeWrapper,
  xoOption6,
  xoOption7,
  xoOption8,
  xoOption9,
  xoOptionA,
  xoOptionB,
  xoOptionC,
  xoOptionD,
  xoOptionE,
  xoOptionF,
  xoOptionG,
  xoOptionH,
  xoOptionI
);
```

C++

```
enum SerializationOptions {
  xoHolderClass,
  xoAttributeOnLastMember,
```

```
    xoInlineArrays,
    xoLiteralParam,
    xoSimpleTypeWrapper,
    xoOption6,
    xoOption7,
    xoOption8,
    xoOption9,
    xoOptionA,
    xoOptionB,
    xoOptionC,
    xoOptionD,
    xoOptionE,
    xoOptionF,
    xoOptionG,
    xoOptionH,
    xoOptionI
  };
```

File

InvokeRegistry

Description

TSerializationOptions (see page 4494) is a set of SerializationOptions values. SerializationOptions values are flags that can be registered with an invokable interface to indicate how it should be called. The following table lists the possible values.

Value	Meaning
xoHolderClass	The remotable object corresponds to a "holder" class. That is, the SOAP representation does not include a node for the class itself, just for its members. This is used when a type that would otherwise not require a remotable class uses a feature only available on remotable classes (such as attributes).
xoAttributeOnLastMember	Attributes of the remotable class are stored on the last member of the class that maps to an element node rather than on the node that corresponds to the class itself. This is necessary if a holder class has attributes.
xoInlineArrays	If the remotable class contains class members that are array types, the encoding of those members omits the type name of the array types. This results in more efficient encoding of holder classes that represent array types.
xoLiteralParam	The remotable class represents one of the two top-level nodes (the input or output value) of a document-literal encoding. The class is treated as a holder class if it is used as the direct input or output parameter of a function.
xoSimpleTypeWrapper	The remotable class is a wrapper for a simple type such as a string or integer. The value that the class represents is represented as a single published property of the remotable class.
xoOption5 - xoOptionI	The remaining options are reserved for future use.

38.4.10 InvokeRegistry.TDynToClear Record

Pascal

```
  TDynToClear = record
    P: Pointer;
    Info: PTypeInfo;
  end;
```

C++

```
  struct TDynToClear {
    void * P;
```

```
  PTypeInfo Info;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.TDynToClear.

38.4.11 InvokeRegistry.TIntfInvokeOption Enumeration

TIntfInvokeOption and TIntfInvokeOptions (see page 4490) represent flags that customize how to handle calls to an invokable interface.

Pascal

```
TIntfInvokeOption = (
  ioDefault,
  ioDocument,
  ioLiteral,
  ioHasDefaultSOAPAction,
  ioHasReturnParamNames,
  ioHasNamespace,
  ioIsAppServerSOAP,
  ioHasUDDIInfo,
  ioHasAllSOAPActions
);
```

C++

```
enum TIntfInvokeOption {
  ioDefault,
  ioDocument,
  ioLiteral,
  ioHasDefaultSOAPAction,
  ioHasReturnParamNames,
  ioHasNamespace,
  ioIsAppServerSOAP,
  ioHasUDDIInfo,
  ioHasAllSOAPActions
};
```

File

InvokeRegistry

Description

TIntfInvokeOptions (see page 4490) is a set of TIntfInvokeOption values. TIntfInvokeOption values are flags that can be registered with an invokable interface to indicate how it should be called. The following table lists the possible values.

Value	Meaning
ioDefault	Calls to the interface use the default options unless overridden by other options. (RPC-based SOAP encoding and HTTP as a transport protocol).
ioDocument	The Web Services server uses document-style encoding.
ioLiteral	The Web Services server uses document-literal style and the types for input and output values have not been unwound to create a method call.
ioHasDefaultSOAPAction	A default SOAP action header has been registered for use with the interface.

ioHasReturnParamNames	A set of parameter names have been registered for use with the interface that are reserved for the return value of function calls. If this option is not set, the strings 'Return' and 'Result' are used.
ioHasNamespace	A namespace was explicitly provided when the interface was registered, rather than using an automatically-generated namespace.
ioIsAppServerSOAP	The interface descends from IAppServerSOAP. Such interfaces are used in multi-tier database applications to interact with the application server.
ioHasUDDIInfo	UDDI information has been registered for the interface. This option is set when the Web Service description is imported using the UDDI browser.
ioHasAllSOAPActions	The SOAPAction headers could not be collapsed into a single string. When representing the SOAP action headers, a single string is created where the first character represents a delimiter that marks the boundary between the various SOAP action headers.

38.4.12 InvokeRegistry.TObjMultiOptions Enumeration

TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.

Pascal

```
TObjMultiOptions = (
  ocDefault,
  ocMultiRef,
  ocNoMultiRef
);
```

C++

```
enum TObjMultiOptions {
  ocDefault,
  ocMultiRef,
  ocNoMultiRef
};
```

File

InvokeRegistry

Description

TObjMultiOptions indicates a class-specific override to the default strategy for converting multiple references to the same object instance into the corresponding SOAP representation. It indicates whether a remotable object should be treated by value (like a structure) or by reference (where object identity is meaningful).

Value	Meaning
ocDefault	Use the default strategy for handling multiple references.
ocMultiRef	Always indicate in the SOAP packet when multiple references refer to the same instance of the specified class.
ocNoMultiRef	Never indicate in the SOAP packet when multiple references refer to the same instance of the specified class. That is, always treat it "by value".

See Also

TRemotableTypeRegistry (see page 4414)

38.4.13 InvokeRegistry.TRemHeaderEntry Record

Pascal

```
TRemHeaderEntry = record
  Name: InvString;
  ExtName: InvString;
  URI: InvString;
end;
```

C++

```
struct TRemHeaderEntry {
  InvString Name;
  InvString ExtName;
  InvString URI;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.TRemHeaderEntry.

38.4.14 InvokeRegistry.TRemRegEntry Record

TRemRegEntry stores information for the remotable type registry.

Pascal

```
TRemRegEntry = record
  ClassType: TClass;
  Info: PTypeInfo;
  URI: InvString;
  Name: InvString;
  ExtName: InvString;
  IsScalar: Boolean;
  MultiRefOpt: TObjMultiOptions;
  SerializationOpt: TSerializationOptions;
  PropNameMap: array of ExtNameMapItem;
end;
```

C++

```
struct TRemRegEntry {
  TClass ClassType;
  PTypeInfo Info;
  InvString URI;
  InvString Name;
  InvString ExtName;
  Boolean IsScalar;
  TObjMultiOptions MultiRefOpt;
  TSerializationOptions SerializationOpt;
  array of ExtNameMapItem PropNameMap;
};
```

File

InvokeRegistry

Description

TRemRegEntry is the type that the remotable type registry uses to store information about a registered type. The TRemRegEntry type is used to represent both remotable classes and remotable types. It contains the following members:

ClassType is the class reference for the class. It is not used for entries that represent types that are not classes.

Info is the type info pointer for a registered class or type.

URI is the namespace URI that the registry associates with the class or type.

Name is the type name of the class or type, as it appears in encoded method calls or WSDL documents.

ExtName is the same as Name.

IsScalar indicates whether the entry describes a class or type that is marshaled as a scalar type.

MultiRefOpt indicates how to marshal multiple instances of a registered class. It is not used for entries that represent types.

SerializationOpt indicates how to convert between a registered class and its encoded representation.

PropNameMap is an array of entries that describe the relationship between properties on the invokable interface and the external names to which they correspond (if different).

38.5 Types

The following table lists types in this documentation.

Types

Name	Description
ERemotableExceptionClass (see page 4487)	This is type InvokeRegistry.ERemotableExceptionClass.
InvString (see page 4488)	This is type InvokeRegistry.InvString.
InvStringDynArray (see page 4488)	This is type InvokeRegistry.InvStringDynArray.
PTRemotable (see page 4488)	This is type InvokeRegistry.PTRemotable.
TCreateInstanceProc (see page 4489)	TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.
TExceptionItemArray (see page 4489)	This is type InvokeRegistry.TExceptionItemArray.
THeaderItemArray (see page 4489)	This is type InvokeRegistry.THeaderItemArray.
THeaderMethodTypeArray (see page 4490)	This is type InvokeRegistry.THeaderMethodTypeArray.
TIntfInvokeOptions (see page 4490)	This is type InvokeRegistry.TIntfInvokeOptions.
TInvokableClassClass (see page 4490)	This is type InvokeRegistry.TInvokableClassClass.
TInvRegIntfEntryArray (see page 4491)	This is type InvokeRegistry.TInvRegIntfEntryArray.
TObjectConvertOptions (see page 4491)	This is type InvokeRegistry.TObjectConvertOptions.
TPascalRemotableClassRegistry (see page 4491)	TPascalRemotableClassRegistry is the type of the remotable class registry.
TPascalRemotableTypeRegistry (see page 4492)	TPascalRemotableTypeRegistry is the type of the remotable type registry.
TRemotableClass (see page 4492)	This is type InvokeRegistry.TRemotableClass.
TRemotableClassRegistry (see page 4492)	This is type InvokeRegistry.TRemotableClassRegistry.
TRemotableXSClass (see page 4493)	This is type InvokeRegistry.TRemotableXSClass.
TRemRegEntryArray (see page 4493)	This is type InvokeRegistry.TRemRegEntryArray.
TRequiredArray (see page 4493)	TRequiredArray is a dynamic array of boolean values.
TSerializationOptions (see page 4494)	This is type InvokeRegistry.TSerializationOptions.
TSOAPAttachmentClass (see page 4494)	This is type InvokeRegistry.TSOAPAttachmentClass.
TSOAPHeaderClass (see page 4494)	This is type InvokeRegistry.TSOAPHeaderClass.

38.5.1 InvokeRegistry.ERemotableExceptionClass Type

Pascal

```
ERemotableExceptionClass = class of ERemotableException;
```

C++

```
ERemotableException ERemotableExceptionClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.ERemotableExceptionClass.

38.5.2 InvokeRegistry.InvString Type

Pascal

```
InvString = xmldom.DOMString;
```

C++

```
xmldom.DOMString InvString;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.InvString.

38.5.3 InvokeRegistry.InvStringDynArray Type

Pascal

```
InvStringDynArray = array of InvString;
```

C++

```
array of InvString InvStringDynArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.InvStringDynArray.

38.5.4 InvokeRegistry.PTRemotable Type

Pascal

```
PTRemotable = ^TRemotable;
```

C++

```
^TRemotable PTRemotable;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.PTRemotable.

38.5.5 InvokeRegistry.TCreateInstanceProc Type

TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.

Pascal

```
TCreateInstanceProc = procedure (out obj: TObject);
```

C++

```
(out obj: TObject) ( TCreateInstanceProc)();
```

File

InvokeRegistry

Description

TCreateInstanceProc is the type for procedures that return an instance of an invokable class on behalf of the invocation registry. When the class that implements an invokable interface is not a descendant of TInvokableClass (see page 4373), or when you want to control the instantiation of the implementation class (for example to use a single shared instance for all incoming requests), supply a procedure of this type when registering the implementation class.

obj returns an instance of the invokable class, which can handle the current incoming request.

38.5.6 InvokeRegistry.TExceptionItemArray Type

Pascal

```
TExceptionItemArray = array of IntfExceptionItem;
```

C++

```
array of IntfExceptionItem TExceptionItemArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TExceptionItemArray.

38.5.7 InvokeRegistry.THeaderItemArray Type

Pascal

```
THeaderItemArray = array of IntfHeaderItem;
```

C++

```
array of IntfHeaderItem THeaderItemArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.THeaderItemArray.

38.5.8 InvokeRegistry.THeaderMethodTypeArray Type

Pascal

```
THeaderMethodTypeArray = array of eHeaderMethodType;
```

C++

```
array of eHeaderMethodType THeaderMethodTypeArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.THeaderMethodTypeArray.

38.5.9 InvokeRegistry.TIntfInvokeOptions Type

Pascal

```
TIntfInvokeOptions = set of TIntfInvokeOption;
```

C++

```
TIntfInvokeOption TIntfInvokeOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TIntfInvokeOptions.

38.5.10 InvokeRegistry.TInvokableClassClass Type

Pascal

```
TInvokableClassClass = class of TInvokableClass;
```

C++

```
TInvokableClass TInvokableClassClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TInvokableClassClass.

38.5.11 InvokeRegistry.TInvRegIntfEntryArray Type

Pascal

```
TInvRegIntfEntryArray = array of InvRegIntfEntry;
```

C++

```
array of InvRegIntfEntry TInvRegIntfEntryArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TInvRegIntfEntryArray.

38.5.12 InvokeRegistry.TObjectConvertOptions Type

Pascal

```
TObjectConvertOptions = set of ObjectConvertOptions;
```

C++

```
ObjectConvertOptions TObjectConvertOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TObjectConvertOptions.

38.5.13 InvokeRegistry.TPascalRemotableClassRegistry Type

TPascalRemotableClassRegistry is the type of the remotable class registry.

Pascal

```
TPascalRemotableClassRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TPascalRemotableClassRegistry;
```

File

InvokeRegistry

Description

TPascalRemotableClassRegistry is a synonym for TRemotableTypeRegistry (see page 4414). It is defined for compatibility with Delphi 6.

See Also

RemClassRegistry (see page 4471)

TRemotable (see page 4408)

38.5.14 InvokeRegistry.TPascalRemotableTypeRegistry Type

TPascalRemotableTypeRegistry is the type of the remotable type registry.

Pascal

```
TPascalRemotableTypeRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TPascalRemotableTypeRegistry;
```

File

InvokeRegistry

Description

TPascalRemotableTypeRegistry is a synonym for TRemotableTypeRegistry (see page 4414). It is defined for compatibility with Delphi 6.

See Also

RemTypeRegistry (see page 4472)

TRemotable (see page 4408)

38.5.15 InvokeRegistry.TRemotableClass Type

Pascal

```
TRemotableClass = class of TRemotable;
```

C++

```
TRemotable TRemotableClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableClass.

38.5.16 InvokeRegistry.TRemotableClassRegistry Type

Pascal

```
TRemotableClassRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TRemotableClassRegistry;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableClassRegistry.

38.5.17 InvokeRegistry.TRemotableXSClass Type

Pascal

```
TRemotableXSClass = class of TRemotableXS;
```

C++

```
TRemotableXS TRemotableXSClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableXSClass.

38.5.18 InvokeRegistry.TRemRegEntryArray Type

Pascal

```
TRemRegEntryArray = array of TRemRegEntry;
```

C++

```
array of TRemRegEntry TRemRegEntryArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemRegEntryArray.

38.5.19 InvokeRegistry.TRequiredArray Type

TRequiredArray is a dynamic array of boolean values.

Pascal

```
TRequiredArray = array of Boolean;
```

C++

```
array of Boolean TRequiredArray;
```

File

InvokeRegistry

Description

TRequiredArray is a dynamic array of boolean values. Each value indicates whether a corresponding item is required (true) or optional (false).

38.5.20 InvokeRegistry.TSerializationOptions Type

Pascal

```
TSerializationOptions = set of SerializationOptions;
```

C++

```
SerializationOptions TSerializationOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSerializationOptions.

38.5.21 InvokeRegistry.TSOAPAttachmentClass Type

Pascal

```
TSOAPAttachmentClass = class of TSOAPAttachment;
```

C++

```
TSOAPAttachment TSOAPAttachmentClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSOAPAttachmentClass.

38.5.22 InvokeRegistry.TSOAPHeaderClass Type

Pascal

```
TSOAPHeaderClass = class of TSOAPHeader;
```

C++

```
TSOAPHeader TSOAPHeaderClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSOAPHeaderClass.

38.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AppNameSpacePrefix (see page 4496)	Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

38.6.1 InvokeRegistry.AppNameSpacePrefix Variable

Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

Pascal

```
AppNameSpacePrefix: string;
```

C++

```
UnicodeString AppNameSpacePrefix;
```

File

InvokeRegistry

Description

Set AppNameSpacePrefix to a unique string that the invocation registry and remotable type registry use when generating namespaces. These generated namespaces must be unique strings that identify invokable interfaces and remotable classes in a Web Service application. They are generated by the RegisterInterface method of the invocation registry and the RegisterXSClass and RegisterXSInfo methods of the remotable class registry if the application does not explicitly supply a namespace.

The invocation registry and remotable type registry generate namespaces using the value of AppNameSpacePrefix, the name of the unit in which the interface or class is defined, and the name of the class, type, or interface. If your unit and interface or class names are at all generic, it is a good idea to supply a unique value to AppNameSpacePrefix to guard against the possibility of another interface, type, or class with the same name that is registered in another unit with the same name.

Both client and server applications must register invokable interfaces and remotable types and classes using the same namespace. If you rely on automatically-generated namespaces, be sure to use the same value of AppNameSpacePrefix in both the client and the server applications.

See Also

InvRegistry (see page 4471)

TInvokableClassRegistry (see page 4379)

TRemotableTypeRegistry (see page 4414)

RemTypeRegistry (see page 4472)

38.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AS_ATTRIBUTE (see page 4497)	Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.
XMLBase64Types (see page 4497)	This is constant InvokeRegistry.XMLBase64Types.
XMLSchemaInstNamepspaces (see page 4498)	This is constant InvokeRegistry.XMLSchemaInstNamepspaces.
XMLSchemaNamepspaces (see page 4498)	This is constant InvokeRegistry.XMLSchemaNamepspaces.

38.7.1 InvokeRegistry.AS_ATTRIBUTE Constant

Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.

Pascal

```
AS_ATTRIBUTE = False;
```

C++

```
AS_ATTRIBUTE = False;
```

File

InvokeRegistry

Description

Assign AS_ATTRIBUTE as the value of the stored directive on a property declaration in a TRemotable (see page 4408) descendant to indicate that the property is encoded as an attribute node rather than as an element node. If a property declaration does not include a stored directive with a value of AS_ATTRIBUTE, it is encoded as an element node instead.

Note: Note:AS_ATTRIBUTE must be used directly. If you set the stored directive to a function that returns AS_ATTRIBUTE (false), the property is still encoded as an element node.

See Also

TRemotable (see page 4408)

TRemotableXS (see page 4432)

38.7.2 InvokeRegistry.XMLBase64Types Constant

Pascal

```
XMLBase64Types: array[0..1] of InvString = ('base64Binary', 'bin.base64');
```

C++

```
array[0..1] of InvString XMLBase64Types = ('base64Binary', 'bin.base64');
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLBase64Types.

38.7.3 InvokeRegistry.XMLSchemaInstNamepspaces Constant

Pascal

```
XMLSchemaInstNamepspaces: array[0..2] of InvString = (SXMLSchemaInstURI_1999,
SXMLSchemaInstURI_2000_10, SXMLSchemaInstURI);
```

C++

```
array[0..2] of InvString XMLSchemaInstNamepspaces = (SXMLSchemaInstURI_1999,
SXMLSchemaInstURI_2000_10, SXMLSchemaInstURI);
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLSchemaInstNamepspaces.

38.7.4 InvokeRegistry.XMLSchemaNamepspaces Constant

Pascal

```
XMLSchemaNamepspaces: array[0..2] of InvString = (SXMLSchemaURI_1999, SXMLSchemaURI_2000_10,
SXMLSchemaURI_2001);
```

C++

```
array[0..2] of InvString XMLSchemaNamepspaces = (SXMLSchemaURI_1999, SXMLSchemaURI_2000_10,
SXMLSchemaURI_2001);
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLSchemaNamepspaces.

39 ISAPIApp Namespace

This is namespace ISAPIApp.

Topics

Name	Description
Classes (see page 4500)	The following table lists classes in this documentation.
Functions (see page 4505)	The following table lists functions in this documentation.
Variables (see page 4507)	The following table lists variables in this documentation.

Classes

	Name	Description
	TISAPIApplication (see page 4500)	TISAPIApplication encapsulates an ISAPI or NSAPI server application.
	TISAPIFactory (see page 4503)	

Functions

	Name	Description
	GetExtensionVersion (see page 4505)	Returns the name and version number of an ISAPI or NSAPI application.
	HttpExtensionProc (see page 4505)	Receives an HTTP request message for an ISAPI or NSAPI application.
	TerminateExtension (see page 4506)	Indicates whether an ISAPI or NSAPI application is ready to be shut down.

Variables

Name	Description
DispatchThread (see page 4507)	This is variable ISAPIApp.DispatchThread.

39.1 Classes

The following table lists classes in this documentation.

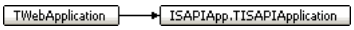
Classes

	Name	Description
	TISAPIApplication (see page 4500)	TISAPIApplication encapsulates an ISAPI or NSAPI server application.
	TISAPIFactory (see page 4503)	

39.1.1 TISAPIApplication Class

TISAPIApplication encapsulates an ISAPI or NSAPI server application.

Class Hierarchy



Pascal

```
TISAPIApplication = class(TWebApplication);
```

C++

```
class TISAPIApplication : public TWebApplication;
```

File

ISAPIApp

Description

TISAPIApplication provides the fundamental behavior of an ISAPI or NSAPI server application. The HTTP server calls the methods of TISAPIApplication when it receives an HTTP request message. TISAPIApplication creates a thread to manage the request message, and a creates a new instance of the Web module for that thread, or reuses an old instance that was deactivated after a previous response was sent.

TISAPIApplication creates TISAPIRequest (see page 4510) and TISAPIResponse (see page 4518) objects to represent the request message and any response that should be sent in return, passes these to the dispatcher so that the response can be filled in, and sends the response if it was not already sent by a TWebActionItem (see page 3335).

Each ISAPI server project automatically declares an Application variable of type TISAPIApplication as the instance of the application. This declaration overrides the Application variable declared in the Forms (see page 1921) unit if the ISAPIApp (see page 4499) unit is declared last in the uses list of the project file. Every ISAPI server application must include the ISAPIApp (see page 4499) unit.

To work with an Apache server, use the TApacheApplication object. To work with a CGI server, use the TCGIApplication object instead.

See Also

TCGIApplication

TISAPIRequest (see page 4510)

TISAPIResponse (see page 4518)

TWebActionItem (see page 3335)

TWebDispatcher (see page 3349)

39.1.1.1 TISAPIApplication Members

The following tables list the members exposed by TISAPIApplication.

Public Methods

	Name	Description
	Create (see page 4501)	Creates an instance of TWebApplication.
	Destroy (see page 4501)	Destroys an instance of TWebApplication.
	GetExtensionVersion (see page 4502)	Returns the application name and version number.
	HttpExtensionProc (see page 4502)	Responds to an HTTP request message.
	TerminateExtension (see page 4503)	Returns true.

39.1.1.2 TISAPIApplication.Create Constructor

Creates an instance of TWebApplication.

Pascal

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

C++

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

Description

Do not call Create. The single Web application object named Application is instantiated automatically when the CGIApp unit, the ISAPIApp unit, or the ApacheApp unit is included in the uses list of the project.

Create allocates the helper objects used by TWebApplication.

See Also

Create

39.1.1.3 TISAPIApplication.Destroy Destructor

Destroys an instance of TWebApplication.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TISAPIApplication();
```

Description

Never call the Destroy or Free method. The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web modules created by the application.

See Also

System::TObject::Free

Run

39.1.1.4 TISAPIApplication Methods

The methods of the TISAPIApplication class are listed here.

Public Methods

	Name	Description
	GetExtensionVersion (see page 4502)	Returns the application name and version number.
	HttpExtensionProc (see page 4502)	Responds to an HTTP request message.
	TerminateExtension (see page 4503)	Returns true.

39.1.1.4.1 TISAPIApplication.GetExtensionVersion Method

Returns the application name and version number.

Pascal

```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL;
```

C++

```
__fastcall BOOL GetExtensionVersion(THSE_VERSION_INFO Ver);
```

Description

The HTTP server calls GetExtensionVersion to obtain the application name and version number. GetExtensionVersion returns the version number for the HTTP functions in the runtime library, and the Title property of the ISAPI application object.

See Also

Title

39.1.1.4.2 TISAPIApplication.HttpExtensionProc Method

Responds to an HTTP request message.

Pascal

```
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
```

C++

```
__fastcall DWORD HttpExtensionProc(TEXTENSION_CONTROL_BLOCK ECB);
```

Description

The HTTP server calls HttpExtensionProc when it receives an HTTP request message. HttpExtensionProc creates the TISAPIRequest (see page 4510) and TISAPIResponse (see page 4518) objects for the HTTP request message, and calls the HandleRequest message so that the response can be filled out and sent. Finally, HttpExtensionProc frees the request and response objects.

If the response is filled out and sent, HttpExtensionProc returns HSE_STATUS_SUCCESS. Otherwise it returns HSE_STATUS_ERROR.

See Also

TISAPIRequest (see page 4510)

TISAPIResponse (see page 4518)

HandleRequest

TWebDispatcher (see page 3349)

39.1.1.4.3 TISAPIApplication.TerminateExtension Method

Returns true.

Pascal

```
function TerminateExtension(dwFlags: DWORD): BOOL;
```

C++

```
__fastcall BOOL TerminateExtension(DWORD dwFlags);
```

Description

The HTTP server calls TerminateExtension when it wants to shut down the ISAPI application. TerminateExtension always returns true, allowing the server to shut down.

39.1.2 TISAPIFactory Class

Class Hierarchy

Pascal

```
TISAPIFactory = class(TObject);
```

C++

```
class TISAPIFactory : public TObject;
```

File

ISAPIApp

See Also

TCGIApplication

39.1.2.1 TISAPIFactory Members

The following tables list the members exposed by TISAPIFactory.

Protected Methods

	Name	Description
	NewRequest (see page 4504)	This is NewRequest, a member of class TISAPIFactory.
	NewResponse (see page 4504)	This is NewResponse, a member of class TISAPIFactory.

Public Methods

	Name	Description
	Create (see page 4503)	Creates an instance of Create. {Placeholder—Update for BCB}

39.1.2.2 TISAPIFactory.Create Constructor

Creates an instance of Create. {Placeholder—Update for BCB}

Pascal

```
constructor Create;
```

C++

```
__fastcall TISAPIFactory();
```

Description

Do not create instances of Create. Instead, use the new keyword, passing any arguments for the Create's constructor.

See Also

System::TObject::Destroy

new

39.1.2.3 TISAPIFactory Methods

The methods of the TISAPIFactory class are listed here.

Protected Methods

	Name	Description
	NewRequest (see page 4504)	This is NewRequest, a member of class TISAPIFactory.
	NewResponse (see page 4504)	This is NewResponse, a member of class TISAPIFactory.

39.1.2.3.1 TISAPIFactory.NewRequest Method

Pascal

```
function NewRequest(var AECB: TEXTENSION_CONTROL_BLOCK): TISAPIRequest; virtual;
```

C++

```
virtual __fastcall TISAPIRequest NewRequest(TEXTENSION_CONTROL_BLOCK AECB);
```

Description

This is NewRequest, a member of class TISAPIFactory.

39.1.2.3.2 TISAPIFactory.NewResponse Method

Pascal

```
function NewResponse(ISAPIRequest: TISAPIRequest): TISAPIResponse; virtual;
```

C++

```
virtual __fastcall TISAPIResponse NewResponse(TISAPIRequest ISAPIRequest);
```

Description

This is NewResponse, a member of class TISAPIFactory.

39.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetExtensionVersion (see page 4505)	Returns the name and version number of an ISAPI or NSAPI application.
	HttpExtensionProc (see page 4505)	Receives an HTTP request message for an ISAPI or NSAPI application.
	TerminateExtension (see page 4506)	Indicates whether an ISAPI or NSAPI application is ready to be shut down.

39.2.1 ISAPIApp.GetExtensionVersion Function

Returns the name and version number of an ISAPI or NSAPI application.

Pascal

```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
```

C++

```
__stdcall BOOL GetExtensionVersion(THSE_VERSION_INFO Ver);
```

File

ISAPIApp

Description

Do not call GetExtensionVersion. ISAPI and NSAPI DLLs export GetExtension to provide an entry point that is called by ISAPI or NSAPI Web servers. This allows the Web server to obtain version information about the web server application.

See Also

HttpExtensionProc (see page 4505)

TerminateExtension (see page 4506)

TIsapiapplication

39.2.2 ISAPIApp.HttpExtensionProc Function

Receives an HTTP request message for an ISAPI or NSAPI application.

Pascal

```
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
```

C++

```
__stdcall DWORD HttpExtensionProc(TEXTENSION_CONTROL_BLOCK ECB);
```

File

ISAPIApp

Description

Do not call HttpExtensionProc. ISAPI and NSAPI DLLs export HttpExtensionProc to provide an entry point that is called by ISAPI or NSAPI Web servers when they receive HTTP request messages.

See Also

GetExtensionVersion (see page 4505)

TerminateExtension (see page 4506)

39.2.3 ISAPIApp.TerminateExtension Function

Indicates whether an ISAPI or NSAPI application is ready to be shut down.

Pascal

```
function TerminateExtension(dwFlags: DWORD): BOOL; stdcall;
```

C++

```
__stdcall BOOL TerminateExtension(DWORD dwFlags);
```

File

ISAPIApp

Description

Do not call TerminateExtension. TerminateExtension is exported by ISAPI and NSAPI DLLs. It provides an entry point that is called by ISAPI or NSAPI Web servers, when they are ready to shut down the ISAPI or NSAPI application, to check whether it would cause problems to do so.

See Also

HttpExtensionProc (see page 4505)

39.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DispatchThread (see page 4507)	This is variable ISAPIApp.DispatchThread.

39.3.1 ISAPIApp.DispatchThread Variable

Pascal

```
DispatchThread: function (AECB: PEXTENSION_CONTROL_BLOCK): Boolean of object;
```

C++

```
function (AECB: PEXTENSION_CONTROL_BLOCK): Boolean of object DispatchThread;
```

File

ISAPIApp

Description

This is variable ISAPIApp.DispatchThread.

40 IsapiHTTP Namespace

This is namespace IsapiHTTP.

Topics

Name	Description
Classes (see page 4510)	The following table lists classes in this documentation.

Classes

	Name	Description
	TISAPIRequest (see page 4510)	TISAPIRequest represents client request information that is passed to a server application DLL.
	TISAPIResponse (see page 4518)	TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

40.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TISAPIRequest (see page 4510)	TISAPIRequest represents client request information that is passed to a server application DLL.
	TISAPIResponse (see page 4518)	TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

40.1.1 TISAPIRequest Class

TISAPIRequest represents client request information that is passed to a server application DLL.

Class Hierarchy

Pascal

```
TISAPIRequest = class(TWebRequest);
```

C++

```
class TISAPIRequest : public TWebRequest;
```

File

IsapiHTTP

Description

Use TISAPIRequest to interpret a client request to a server application DLL.

ISAPI application objects create TISAPIRequest objects to capture client requests. The properties of the TISAPIRequest objects are obtained from the Web server. The ISAPI application uses TISAPIRequest objects to create corresponding TISAPIResponse (see page 4518) objects. The TWebDispatcher (see page 3349) for the application then invokes the TWebActionItem (see page 3335) that is associated with the path portion of the URL of the TISAPIRequest object.

See Also

TISAPIApplication (see page 4500)

TISAPIResponse (see page 4518)

TWebActionItem (see page 3335)

TWebDispatcher (see page 3349)

Accessing Client Request Information

40.1.1.1 TISAPIRequest Members

The following tables list the members exposed by TISAPIRequest.

Protected Methods

	Name	Description
	GetDateVariable (see page 4513)	Extracts a date value from the extension control block.
	GetIntegerVariable (see page 4514)	Extracts an integer value from the extension control block.
	GetInternalPathInfo (see page 3366)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 3366)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 4514)	Extracts a string value from the extension control block.
	UpdateMethodType (see page 3369)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	Create (see page 4512)	Creates an instance of TISAPIRequest (see page 4510) to represent the request message information contained in the AECB parameter.
	Destroy (see page 3361)	Destroys an instance of TWebRequest (see page 3358).
	ExtractContentFields (see page 3363)	Parses the Content (see page 3373) property into its constituent fields.
	ExtractCookieFields (see page 3363)	Parses the Cookie (see page 3377) property into its constituent fields.
	ExtractFields (see page 3363)	This is the overview for the ExtractFields method overload.
	ExtractQueryFields (see page 3364)	Parses the Query (see page 3384) property into its constituent fields.
	GetFieldByName (see page 4513)	Returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 4515)	Reads Count bytes from the HTTP request into Buffer.
	ReadString (see page 4515)	Reads Count bytes from the HTTP request into a string.
	ReadUnicodeString (see page 3368)	This is ReadUnicodeString, a member of class TWebRequest.
	TranslateURI (see page 4516)	Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.
	WriteClient (see page 4516)	Writes Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 4516)	Writes the header contents back to the HTTP client.
	WriteString (see page 4517)	Writes the string AString back to the HTTP client.

Public Properties

	Name	Description
	Accept (see page 3371)	Reports the value of the Accept header of the HTTP request message.
	Authorization (see page 3372)	Reports the value of the Authorization header of the HTTP request message.
	CacheControl (see page 3372)	Reports the value of the Cache-Control header of the HTTP request message.
	Connection (see page 3373)	Reports the value of the Connection header of the HTTP request message.
	Content (see page 3373)	Represents the unparsed contents of the HTTP request message.
	ContentEncoding (see page 3374)	Reports the value of the Content (see page 3373)-Encoding header of the HTTP request message.
	ContentFields (see page 3375)	Reports the parsed content of a POST request message.
	ContentLength (see page 3375)	Reports the value of the Content (see page 3373)-Length header of the HTTP request message.
	ContentParser (see page 3376)	Refers to the object's content parser object.
	ContentType (see page 3376)	Reports the value of the Content (see page 3373)-Type header of the HTTP request message.
	ContentVersion (see page 3376)	Reports the value of the Content (see page 3373)-Version header of the HTTP request message.
	Cookie (see page 3377)	Reports the value of the Cookie header of the HTTP request message.
	CookieFields (see page 3377)	Represents the parsed contents of the Cookie (see page 3377) header of an HTTP request message.
	Date (see page 3378)	Reports the value of the Date header of the HTTP request message.
	DerivedFrom (see page 3378)	Reports the value of the Derived-From (see page 3379) header of the HTTP request message.
	ECB (see page 4518)	Indicates the extension control block structure used by the server application DLL to communicate with the Web server.
	Expires (see page 3379)	Reports the value of the Expires header of the HTTP request message.
	Files (see page 3379)	Provides access to the files uploaded in the HTTP request.
	From (see page 3379)	Reports the value of the From header of the HTTP request message.

	Host (see page 3380)	Reports the value of the Host header of the HTTP request message.
	IfModifiedSince (see page 3380)	Reports the value of the If-Modified-Since header of the HTTP request message.
	InternalPathInfo (see page 3381)	Returns the path info portion of the URL (see page 3388).
	InternalScriptName (see page 3381)	Returns the ScriptName (see page 3386) part of the URL (see page 3388).
	Method (see page 3382)	This is Method, a member of class TWebRequest.
	MethodType (see page 3382)	This is MethodType, a member of class TWebRequest.
	PathInfo (see page 3382)	Reports the value of the path information (if any) of the URL (see page 3388) specified in the HTTP request message.
	PathTranslated (see page 3383)	Represents a translation of the PathInfo (see page 3382) property to a fully qualified path on the Web server.
	ProtocolVersion (see page 3383)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
	Query (see page 3384)	Indicates the value of the query information (if any) of the URL (see page 3388) specified in the HTTP request message.
	QueryFields (see page 3384)	Represents the parsed contents of the Query (see page 3384) in the URL (see page 3388) of the HTTP request message.
	RawContent (see page 3385)	This is RawContent, a member of class TWebRequest.
	Referer (see page 3385)	Reports the value of the Referer header of the HTTP request message.
	RemoteAddr (see page 3386)	Indicates the remote IP address of the client associated with the HTTP request message.
	RemoteHost (see page 3386)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
	ScriptName (see page 3386)	Indicates the value of the script information (if any) of the URL (see page 3388) specified in the HTTP request message.
	ServerPort (see page 3387)	Indicates the port number of the port on the Web server that receives the HTTP request message.
	Title (see page 3387)	Reports the value of the Title header of the HTTP request message.
	URL (see page 3388)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
	UserAgent (see page 3388)	Reports the value of the User-Agent header of the HTTP request message.

40.1.1.2 TISAPIRequest.Create Constructor

Creates an instance of TISAPIRequest (see page 4510) to represent the request message information contained in the AECB parameter.

Pascal

```
constructor Create(AECB: PEXTENSION_CONTROL_BLOCK);
```

C++

```
__fastcall TISAPIRequest(PEXTENSION_CONTROL_BLOCK AECB);
```

Description

Applications don't need to call Create to instantiate an ISAPI request object. TISAPIApplication creates a TISAPIRequest (see page 4510) object whenever it receives a client request.

See Also

ECB (see page 4518)

TISAPIApplication (see page 4500)

40.1.1.3 TISAPIRequest Methods

The methods of the TISAPIRequest class are listed here.

Protected Methods

	Name	Description
	GetDateVariable (see page 4513)	Extracts a date value from the extension control block.
	GetIntegerVariable (see page 4514)	Extracts an integer value from the extension control block.
	GetStringVariable (see page 4514)	Extracts a string value from the extension control block.

Public Methods

	Name	Description
	GetFieldByName (see page 4513)	Returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 4515)	Reads Count bytes from the HTTP request into Buffer.
	ReadString (see page 4515)	Reads Count bytes from the HTTP request into a string.
	TranslateURI (see page 4516)	Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.
	WriteClient (see page 4516)	Writes Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 4516)	Writes the header contents back to the HTTP client.
	WriteString (see page 4517)	Writes the string AString back to the HTTP client.

40.1.1.3.1 TISAPIRequest.GetDateVariable Method

Extracts a date value from the extension control block.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

GetDateVariable reads date-time values from the extension control block specified by the ECB (see page 4518) property. The ISAPI request object uses GetDateVariable to access the properties that represent dates from the HTTP request headers.

See Also

Date (see page 3378)

ECB (see page 4518)

Expires (see page 3379)

IfModifiedSince (see page 3380)

40.1.1.3.2 TISAPIRequest.GetFieldByName Method

Returns the value of the HTTP header variable specified by the Name parameter.

Pascal

```
function GetFieldByName(const Name: AnsiString): AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name);
```

Description

Use GetFieldByName to read the value of an HTTP header variable that is not surfaced as one of the properties of TISAPIRequest (see page 4510). For information on possible HTTP request header variables, see the current Internet standards documents.

See Also

ECB (see page 4518)

GetStringVariable (see page 4514)

40.1.1.3.3 TISAPIRequest.GetIntegerVariable Method

Extracts an integer value from the extension control block.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

GetIntegerVariable reads integer values from the extension control block specified by the ECB (see page 4518) property. The ISAPI request object uses GetIntegerVariable to access the properties that represent integers from the HTTP request headers.

See Also

ContentLength (see page 3375)

ECB (see page 4518)

ServerPort (see page 3387)

40.1.1.3.4 TISAPIRequest.GetStringVariable Method

Extracts a string value from the extension control block.

Pascal

```
function GetStringVariable(Index: Integer): AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

GetStringVariable reads string values from the extension control block specified by the ECB (see page 4518) property. The ISAPI request object uses GetStringVariable to access the properties that represent strings from the HTTP request headers.

See Also

Accept (see page 3371)

CacheControl (see page 3372)

Connection (see page 3373)

Content (see page 3373)

ContentEncoding (see page 3374)

ContentType (see page 3376)

ContentVersion (see page 3376)

Cookie (see page 3377)

DerivedFrom (see page 3378)

ECB (see page 4518)

From (see page 3379)

Host (see page 3380)

TISAPIRequest (see page 4510)

PathInfo (see page 3382)

PathTranslated (see page 3383)

ProtocolVersion (see page 3383)

Query (see page 3384)

Referer (see page 3385)

RemoteAddr (see page 3386)

RemoteHost (see page 3386)

ScriptName (see page 3386)

Title (see page 3387)

URL (see page 3388)

UserAgent (see page 3388)

40.1.1.3.5 TISAPIRequest.ReadClient Method

Reads Count bytes from the HTTP request into Buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count);
```

Description

Use ReadClient to load a chunk of information from the contents of an HTTP request into a buffer. ReadClient allows the server application to get the next chunk of information when the content of the request message is too large for the Content property. ReadClient returns the number of bytes actually read. If no more content is available, ReadClient returns -1.

See Also

ReadString (see page 4515)

WriteClient (see page 4516)

40.1.1.3.6 TISAPIRequest.ReadString Method

Reads Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): AnsiString; override;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count);
```

Description

Use ReadString to load the next chunk of content information form the HTTP request into a string. ReadString allows the server application to get the next chunk of information when the content of the request message is too large for the Content property.

See Also

GetStringVariable (see page 4514)

ReadClient (see page 4515)

WriteString (see page 4517)

40.1.1.3.7 TISAPIRequest.TranslateURI Method

Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.

Pascal

```
function TranslateURI(const URI: string): string; override;
```

C++

```
virtual __fastcall UnicodeString TranslateURI(const UnicodeString URI);
```

Description

Use TranslateURI is used to translate the URI specified in a client request into a format that can be used by the server application.

See Also

PathTranslated (see page 3383)

URL (see page 3388)

40.1.1.3.8 TISAPIRequest.WriteClient Method

Writes Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count);
```

Description

Use WriteClient to write the contents of a response back to the Web client.

See Also

ReadClient (see page 4515)

TISAPIResponse (see page 4518)

WriteString (see page 4517)

WriteHeaders (see page 4516)

40.1.1.3.9 TISAPIRequest.WriteHeaders Method

Writes the header contents back to the HTTP client.

Pascal

```
function WriteHeaders(StatusCode: Integer; const StatusString: AnsiString; const Headers:
AnsiString): Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteHeaders(int StatusCode, const AnsiString StatusString, const
AnsiString Headers);
```

Description

WriteHeaders writes the HTTP headers of a response message before a call to the WriteClient (see page 4516) method. These headers are used on the HTTP message that the WriteClient (see page 4516) method sends in response to this request.

StatusCode is the status code of the response. This informs the client of the status of the request. For a list of status code values, see the StatusCode property of TWebResponse (see page 3389).

StatusString (see page 3442) is a text string that describes the status that StatusCode indicates.

Headers is a formatted string that includes any other HTTP headers to add. It represents each header as a string of the form 'Headername: Value', where name/value pairs are separated by line breaks.

WriteHeaders returns true if it succeeds in writing the specified headers.

See Also

WriteClient (see page 4516)

StatusCode (see page 3413)

40.1.1.3.10 TISAPIRequest.WriteString Method

Writes the string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: AnsiString): Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString);
```

Description

Use WriteString to write the contents of a response back to the Web client.

See Also

ReadString (see page 4515)

TISAPIResponse (see page 4518)

WriteClient (see page 4516)

40.1.1.4 TISAPIRequest Properties

The properties of the TISAPIRequest class are listed here.

Public Properties

	Name	Description
	ECB (see page 4518)	Indicates the extension control block structure used by the server application DLL to communicate with the Web server.

40.1.1.4.1 TISAPIRequest.ECB Property

Indicates the extension control block structure used by the server application DLL to communicate with the Web server.

Pascal

```
property ECB: PEXTENSION_CONTROL_BLOCK;
```

C++

```
__property PEXTENSION_CONTROL_BLOCK ECB;
```

Description

Read ECB to gain direct access to the extension control block that the server application DLL uses to represent the client request message. Most applications will not need to use the ECB property because the header fields values can be read using the GetFieldByName (see page 4513) method.

See Also

GetFieldByName (see page 4513)

40.1.2 TISAPIResponse Class

TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

Class Hierarchy

Pascal

```
TISAPIResponse = class(TWebResponse);
```

C++

```
class TISAPIResponse : public TWebResponse;
```

File

IsapiHTTP

Description

Use TISAPIResponse to specify the response to an HTTP request message.

An ISAPI application creates a TISAPIResponse object based on the TISAPIRequest (see page 4510) object for an incoming HTTP request message. TWebDispatcher (see page 3349) then passes the TISAPIResponse object to the TWebActionItem (see page 3335) associated with the TISAPIRequest (see page 4510) object. The TWebActionItem (see page 3335) sets the properties of the TISAPIResponse object.

See Also

TISAPIApplication (see page 4500)

TISAPIRequest (see page 4510)

TWebActionItem (see page 3335)

TWebDispatcher (see page 3349)

Creating HTTP Response Messages

40.1.2.1 TISAPIResponse Members

The following tables list the members exposed by TISAPIResponse.

Protected Fields

	Name	Description
	FCustomHeaders (see page 3392)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 3393)	This is FHTTPRequest, a member of class TWebResponse.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 3394)	Appends the custom headers specified by the CustomHeaders (see page 3408) property to the Headers parameter.
	FormatAuthenticate (see page 3394)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).
V	GetContent (see page 4521)	Reads the value of the Content property.
V	GetDateVariable (see page 4521)	Extracts a date from the internal table of HTTP header variables.
V	GetIntegerVariable (see page 4522)	Extracts an integer value from the internal table of HTTP header variables.
V	GetLogMessage (see page 4522)	Reads the value of the LogMessage property.
V	GetStatusCode (see page 4522)	Reads the value of the StatusCode property.
V	GetStringVariable (see page 4523)	Extracts a string value from the internal table of HTTP header variables.
V	InitResponse (see page 4523)	This is InitResponse, a member of class TISAPIResponse.
V	SetContent (see page 4525)	Sets the Content property.
V	SetContentStream (see page 3400)	Sets the value of the ContentStream (see page 3406) property to the Value parameter.
V	SetDateVariable (see page 4525)	Sets a date-time valued HTTP header property.
V	SetIntegerVariable (see page 4526)	Sets an integer valued HTTP header property.
V	SetLogMessage (see page 4526)	Sets the value of the LogMessage property.
V	SetStatusCode (see page 4526)	Sets the StatusCode property.
V	SetStringVariable (see page 4527)	Sets a string valued HTTP header property.

Public Methods

	Name	Description
	Create (see page 4520)	Creates an instance of TISAPIResponse (see page 4518) to represent the response to the HTTPRequest parameter.
V	Destroy (see page 3392)	Disposes of an object instance.
	GetCustomHeader (see page 3395)	Returns the value of the custom header with the specified name.
V	SendRedirect (see page 4524)	Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.
V	SendResponse (see page 4524)	Sends the ISAPI response message to the web client.
V	SendStream (see page 4524)	Sends the ISAPI response message with the contents set to the contents of AStream.
V	Sent (see page 4525)	Indicates whether the ISAPI response message has been sent.
	SetCookieField (see page 3400)	Adds a cookie header to the response message.
	SetCustomHeader (see page 3400)	Replaces the value of a specified custom header.

Public Properties

	Name	Description
	Allow (see page 3404)	Lists the HTTP request methods that the application can service.
	Content (see page 3404)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 3405)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 3405)	Specifies the length, in bytes, of the Content (see page 3404) property.
	ContentStream (see page 3406)	Specifies the stream object for writing the contents of the HTTP response message to the web client.

	ContentType (see page 3406)	Indicates the media type of the Content (see page 3404) of the HTTP response message.
	ContentVersion (see page 3407)	Represents the version associated with the Content (see page 3404) of the HTTP response message.
R	Cookies (see page 3407)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 3408)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 3408)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 3409)	Specifies the URI of the resource where the Content (see page 3404) of the HTTP response originated.
	Expires (see page 3409)	Specifies the date and time after which the Content (see page 3404) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 3410)	Represents the HTTP request message for which this is a response.
	LastModified (see page 3410)	Indicates the date and time that the value of the Content (see page 3404) property was last updated.
	Location (see page 3411)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 3411)	Describes the HTTP response message for event logging purposes.
	RawContent (see page 3411)	This is RawContent, a member of class TWebResponse.
	Realm (see page 3411)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 3412)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 3412)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 3413)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 3414)	Specifies a title for the Content (see page 3404) of the HTTP response message.
	Version (see page 3415)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 3415)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 3413) 401 (Unauthorized).

40.1.2.2 TISAPIResponse.Create Constructor

Creates an instance of TISAPIResponse (see page 4518) to represent the response to the HTTPRequest parameter.

Pascal

```
constructor Create(HTTPRequest: TWebRequest);
```

C++

```
__fastcall TISAPIResponse(TWebRequest HTTPRequest);
```

Description

Applications don't need to call Create to instantiate a ISAPI response object. TISAPIApplication creates a TISAPIResponse (see page 4518) for each TISAPIRequest (see page 4510) it creates.

See Also

TISAPIApplication (see page 4500)

TISAPIRequest (see page 4510)

40.1.2.3 TISAPIResponse Methods

The methods of the TISAPIResponse class are listed here.

Protected Methods

	Name	Description
	GetContent (see page 4521)	Reads the value of the Content property.
	GetDateVariable (see page 4521)	Extracts a date from the internal table of HTTP header variables.
	GetIntegerVariable (see page 4522)	Extracts an integer value from the internal table of HTTP header variables.
	GetLogMessage (see page 4522)	Reads the value of the LogMessage property.
	GetStatusCode (see page 4522)	Reads the value of the StatusCode property.
	GetStringVariable (see page 4523)	Extracts a string value from the internal table of HTTP header variables.
	InitResponse (see page 4523)	This is InitResponse, a member of class TISAPIResponse.
	SetContent (see page 4525)	Sets the Content property.
	SetDateVariable (see page 4525)	Sets a date-time valued HTTP header property.
	SetIntegerVariable (see page 4526)	Sets an integer valued HTTP header property.
	SetLogMessage (see page 4526)	Sets the value of the LogMessage property.
	SetStatusCode (see page 4526)	Sets the StatusCode property.
	SetStringVariable (see page 4527)	Sets a string valued HTTP header property.

Public Methods

	Name	Description
	SendRedirect (see page 4524)	Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.
	SendResponse (see page 4524)	Sends the ISAPI response message to the web client.
	SendStream (see page 4524)	Sends the ISAPI response message with the contents set to the contents of AStream.
	Sent (see page 4525)	Indicates whether the ISAPI response message has been sent.

40.1.2.3.1 TISAPIResponse.GetContent Method

Reads the value of the Content property.

Pascal

```
function GetContent: AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetContent();
```

Description

GetContent is the protected access function for the Content property.

See Also

Content (see page 3404)

SetContent (see page 4525)

40.1.2.3.2 TISAPIResponse.GetDateVariable Method

Extracts a date from the internal table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

The ISAPI response object uses GetDateVariable to access the properties that represent dates.

See Also

Date (see page 3408)

Expires (see page 3409)

LastModified (see page 3410)

SetDateVariable (see page 4525)

40.1.2.3.3 TISAPIResponse.GetIntegerVariable Method

Extracts an integer value from the internal table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

The ISAPI response object uses GetIntegerVariable to access the properties that represent integers.

See Also

ContentLength (see page 3405)

SetIntegerVariable (see page 4526)

40.1.2.3.4 TISAPIResponse.GetLogMessage Method

Reads the value of the LogMessage property.

Pascal

```
function GetLogMessage: string; override;
```

C++

```
virtual __fastcall UnicodeString GetLogMessage();
```

Description

GetLogMessage is the protected access function for LogMessage property.

See Also

LogMessage (see page 3411)

SetLogMessage (see page 4526)

40.1.2.3.5 TISAPIResponse.GetStatusCode Method

Reads the value of the StatusCode property.

Pascal

```
function GetStatusCode: Integer; override;
```

C++

```
virtual __fastcall int GetStatusCode();
```

Description

GetStatusCode is the protected access function for the StatusCode property.

See Also

StatusCode (see page 3413)

SetStatusCode (see page 3402)

40.1.2.3.6 TISAPIResponse.GetStringVariable Method

Extracts a string value from the internal table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

The ISAPI response object uses GetStringVariable to access the properties that represent strings.

See Also

Allow (see page 3404)

ContentEncoding (see page 3405)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

Location (see page 3411)

Realm (see page 3411)

ReasonString (see page 3412)

Server (see page 3412)

SetStringVariable (see page 4527)

Title (see page 3414)

Version (see page 3415)

WWWAuthenticate (see page 3415)

40.1.2.3.7 TISAPIResponse.InitResponse Method

Pascal

```
procedure InitResponse; virtual;
```

C++

```
virtual __fastcall InitResponse();
```

Description

This is InitResponse, a member of class TISAPIResponse.

40.1.2.3.8 TISAPIResponse.SendRedirect Method

Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.

Pascal

```
procedure SendRedirect(const URI: AnsiString); override;
```

C++

```
virtual __fastcall SendRedirect(const AnsiString URI);
```

Description

Call SendRedirect to redirect the web client request to another URI that is the proper target of the request. SendRedirect does the same thing as setting the StatusCode to 301, the Location to URI, and calling SendResponse (see page 4524).

See Also

Location (see page 3411)

SendResponse (see page 4524)

StatusCode (see page 3413)

40.1.2.3.9 TISAPIResponse.SendResponse Method

Sends the ISAPI response message to the web client.

Pascal

```
procedure SendResponse; override;
```

C++

```
virtual __fastcall SendResponse();
```

Description

Call SendResponse after setting the properties of the HTTP response object to specify the response that should be made to the web client. If the request should be redirected to another URI, it is more efficient to call SendRedirect (see page 4524).

See Also

SendRedirect (see page 4524)

40.1.2.3.10 TISAPIResponse.SendStream Method

Sends the ISAPI response message with the contents set to the contents of AStream.

Pascal

```
procedure SendStream(AStream: TStream); override;
```

C++

```
virtual __fastcall SendStream(TStream AStream);
```

Description

Only call SendStream after first calling SendResponse (see page 4524). Unlike SendStream, SendResponse (see page 4524) fills the header of the HTTP response message, based on the properties of the TISAPIResponse (see page 4518) object. If SendResponse (see page 4524) has not been called first to set the header of the HTTP response message, set the ContentStream property to AStream, and call SendResponse (see page 4524).

See Also

ContentStream (see page 3406)

SendResponse (see page 4524)

Sent (see page 4525)

40.1.2.3.11 TISAPIResponse.Sent Method

Indicates whether the ISAPI response message has been sent.

Pascal

```
function Sent: Boolean; override;
```

C++

```
virtual __fastcall Boolean Sent();
```

Description

Sent returns true if the SendRedirect (see page 4524) or SendStream (see page 4524) method has been called. TISAPIApplication objects read Sent to determine whether they should send a response to the web client.

See Also

SendRedirect (see page 4524)

SendStream (see page 4524)

40.1.2.3.12 TISAPIResponse.SetContent Method

Sets the Content property.

Pascal

```
procedure SetContent(const Value: AnsiString); override;
```

C++

```
virtual __fastcall SetContent(const AnsiString Value);
```

Description

SetContent is the protected method used by ISAPI response objects to set the Content property.

See Also

Content (see page 3404)

GetContent (see page 4521)

40.1.2.3.13 TISAPIResponse.SetDateVariable Method

Sets a date-time valued HTTP header property.

Pascal

```
procedure SetDateVariable(Index: Integer; const Value: TDateTime); override;
```

C++

```
virtual __fastcall SetDateVariable(int Index, const TDateTime Value);
```

Description

SetDateVariable writes date-time to the internal table used to store HTTP header variables. The ISAPI response object uses SetDateVariable to set the properties that represent dates.

See Also

Date (see page 3408)

Expires (see page 3409)

GetDateVariable (see page 4521)

LastModified (see page 3410)

40.1.2.3.14 TISAPIResponse.SetIntegerVariable Method

Sets an integer valued HTTP header property.

Pascal

```
procedure SetIntegerVariable(Index: Integer; Value: Integer); override;
```

C++

```
virtual __fastcall SetIntegerVariable(int Index, int Value);
```

Description

SetIntegerVariable writes integer values to the internal table used to store HTTP header variables. The ISAPI response object uses SetIntegerVariable to set the properties that represent integers.

See Also

ContentLength (see page 3405)

GetIntegerVariable (see page 4522)

40.1.2.3.15 TISAPIResponse.SetLogMessage Method

Sets the value of the LogMessage property.

Pascal

```
procedure SetLogMessage(const Value: string); override;
```

C++

```
virtual __fastcall SetLogMessage(const UnicodeString Value);
```

Description

SetLogMessage is the protected method used by ISAPI response objects to set the LogMessage property.

See Also

GetLogMessage (see page 4522)

LogMessage (see page 3411)

40.1.2.3.16 TISAPIResponse.SetStatusCode Method

Sets the StatusCode property.

Pascal

```
procedure SetStatusCode(Value: Integer); override;
```

C++

```
virtual __fastcall SetStatusCode(int Value);
```

Description

SetStatusCode is the protected method used by ISAPI response objects to set the StatusCode property.

See Also

StatusCode (see page 3413)

GetStatusCode (see page 3396)

40.1.2.3.17 TISAPIResponse.SetStringVariable Method

Sets a string valued HTTP header property.

Pascal

```
procedure SetStringVariable(Index: Integer; const Value: AnsiString); override;
```

C++

```
virtual __fastcall SetStringVariable(int Index, const AnsiString Value);
```

Description

SetStringVariable writes string values to the internal table used to store HTTP header variables. The ISAPI response object uses SetStringVariable to set the properties that represent strings.

See Also

Allow (see page 3404)

ContentEncoding (see page 3405)

ContentType (see page 3406)

ContentVersion (see page 3407)

DerivedFrom (see page 3409)

GetStringVariable (see page 4523)

Location (see page 3411)

Realm (see page 3411)

ReasonString (see page 3412)

Server (see page 3412)

Title (see page 3414)

Version (see page 3415)

WWWAuthenticate (see page 3415)

41 Java Namespace

This is namespace Java.

Namespaces

Name	Description
Sql (see page 4530)	This is namespace Java.Sql.

41.1 Java.Sql Namespace

This is namespace Java.Sql.

Topics

Name	Description
Classes (see page 4530)	The following table lists classes in this documentation.

Classes

	Name	Description
	SQLException (see page 4530)	This is class Java.Sql.SQLException.

41.1.1 Classes

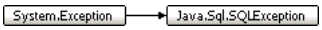
The following table lists classes in this documentation.

Classes

	Name	Description
	SQLException (see page 4530)	This is class Java.Sql.SQLException.

41.1.1.1 SQLException Class

Class Hierarchy



Pascal

```
SQLException = class(System.Exception);
```

C++

```
class SQLException : public System.Exception;
```

File

SQLException.cs

Description

This is class Java.Sql.SQLException.

41.1.1.1.1 SQLException Members

The following tables list the members exposed by SQLException.

Public Methods

	Name	Description
	SQLException (see page 4531)	This is the overview for the SQLException constructor overload.

Public Properties

	Name	Description
	ErrorCode (see page 4531)	This is ErrorCode, a member of class SQLException.
	SQLState (see page 4532)	This is SQLState, a member of class SQLException.

41.1.1.1.2 SQLException Constructor

This is the overview for the SQLException constructor overload.

Overload List

	Name	Description
	SQLException.SQLException (string) (see page 4531)	This is SQLException, a member of class SQLException.
	SQLException.SQLException (string, string, int, Exception) (see page 4531)	This is SQLException, a member of class SQLException.

41.1.1.1.2.1 SQLException.SQLException Constructor (string)

Pascal

```
constructor Create(reason: string);
```

C++

```
SQLException(string reason);
```

Description

This is SQLException, a member of class SQLException.

41.1.1.1.2.2 SQLException.SQLException Constructor (string, string, int, Exception)

Pascal

```
constructor Create(reason: string; SQLState: string; vendorCode: Integer; cause: Exception);
```

C++

```
SQLException(string reason, string SQLState, int vendorCode, Exception cause);
```

Description

This is SQLException, a member of class SQLException.

41.1.1.1.3 SQLException Properties

The properties of the SQLException class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 4531)	This is ErrorCode, a member of class SQLException.
	SQLState (see page 4532)	This is SQLState, a member of class SQLException.

41.1.1.1.3.1 SQLException.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class SQLException.

41.1.1.1.3.2 SQLException.SQLState Property

Pascal

```
property SQLState: string;
```

C++

```
__property string SQLState;
```

Description

This is SQLState, a member of class SQLException.

42 ListActns Namespace

This is namespace ListActns.

Topics

Name	Description
Classes (see page 4535)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4586)	The following table lists structs, records, enums in this documentation.
Types (see page 4587)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomListAction (see page 4535)	TCustomListAction is the base class for action objects that provide items to client controls.
	TCustomStaticListAction (see page 4543)	This is class ListActns.TCustomStaticListAction.
	TCustomVirtualListAction (see page 4548)	This is class ListActns.TCustomVirtualListAction.
	TListActionLink (see page 4550)	This is class ListActns.TListActionLink.
	TListControlItem (see page 4554)	TListControlItem represents an item with a text value, an image, and a data pointer.
	TListControlItems (see page 4560)	TListControlItems manages a collection of TListControlItem (see page 4554) objects.
	TStaticListAction (see page 4567)	TStaticListAction provides items to client controls.
	TStaticListItems (see page 4576)	TStaticListItems represents the items that TStaticListAction (see page 4567) supplies to client controls.
	TVirtualListAction (see page 4578)	TVirtualListAction provides items to a client controls.

Structs, Records, Enums

	Name	Description
	TListItemsSortType (see page 4586)	This is record ListActns.TListItemsSortType.

Types

Name	Description
TGetItemCountEvent (see page 4587)	This is type ListActns.TGetItemCountEvent.
TGetItemEvent (see page 4587)	This is type ListActns.TGetItemEvent.
TGetVirtualItemEvent (see page 4588)	This is type ListActns.TGetVirtualItemEvent.
TItemSelectedEvent (see page 4588)	This is type ListActns.TItemSelectedEvent.
TListCompareEvent (see page 4588)	This is type ListActns.TListCompareEvent.
TListControlItemClass (see page 4589)	This is type ListActns.TListControlItemClass.

TListItemsCompare (see page 4589)	This is type ListActns.TListItemsCompare.

42.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomListAction (see page 4535)	TCustomListAction is the base class for action objects that provide items to client controls.
	TCustomStaticListAction (see page 4543)	This is class ListActns.TCustomStaticListAction.
	TCustomVirtualListAction (see page 4548)	This is class ListActns.TCustomVirtualListAction.
	TListActionLink (see page 4550)	This is class ListActns.TListActionLink.
	TListControlItem (see page 4554)	TListControlItem represents an item with a text value, an image, and a data pointer.
	TListControlItems (see page 4560)	TListControlItems manages a collection of TListControlItem (see page 4554) objects.
	TStaticListAction (see page 4567)	TStaticListAction provides items to client controls.
	TStaticListItems (see page 4576)	TStaticListItems represents the items that TStaticListAction (see page 4567) supplies to client controls.
	TVirtualListAction (see page 4578)	TVirtualListAction provides items to a client controls.

42.1.1 TCustomListAction Class

TCustomListAction is the base class for action objects that provide items to client controls.

Class Hierarchy

Pascal

```
TCustomListAction = class(TCustomAction);
```

C++

```
class TCustomListAction : public TCustomAction;
```

File

ListActns

Description

TCustomListAction implements the common behavior for actions that supply the items to a list view, list box or combo box.

When an application sets the Active (see page 4540) property of a list action to true, the action fills all client list boxes and combo boxes with items. Each item may include information such as a text caption, an image, and a pointer to data.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 4542) event.

When a TCustomListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TStaticListAction (see page 4567)

TVirtualListAction (see page 4578)

42.1.1.1 TCustomListAction Members

The following tables list the members exposed by TCustomListAction.

Protected Events

	Name	Description
	OnGetItemCount (see page 4542)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 4542)	Occurs when a client control executes the action because a user selected one of the items.

Protected Methods

	Name	Description
V	GetCount (see page 4537)	This is GetCount, a member of class TCustomListAction.
V	GetString (see page 4538)	Returns the value of the Strings (see page 4541) property.
V	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
V	SetString (see page 4539)	Sets the value of the Strings (see page 4541) property.

Protected Properties

	Name	Description
	Images (see page 4540)	Lists the images that the list action supplies along with items.
R	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Methods

	Name	Description
V	Create (see page 4536)	Creates an instance of TCustomListAction (see page 4535).
V	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
V	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
	Active (see page 4540)	Indicates whether the list action has supplied items to its client controls.
R	Count (see page 4540)	This is Count, a member of class TCustomListAction.
	ItemIndex (see page 4541)	Specifies the index of the selected item.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

42.1.1.2 TCustomListAction.Create Constructor

Creates an instance of TCustomListAction (see page 4535).

Pascal

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

C++

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

Description

Most applications do not need to call the TCustomListAction (see page 4535) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomListAction (see page 4535) instance.

See Also

Owner

42.1.1.3 TCustomListAction Methods

The methods of the TCustomListAction class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 4537)	This is GetCount, a member of class TCustomListAction.
	GetString (see page 4538)	Returns the value of the Strings (see page 4541) property.
	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
	SetString (see page 4539)	Sets the value of the Strings (see page 4541) property.

Public Methods

	Name	Description
	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

42.1.1.3.1 TCustomListAction.ExecuteTarget Method

Generates an OnItemSelected (see page 4542) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget is called automatically when the user selects one of the items that the list action supplies to a client list box or combo box.

Target is the control whose item was selected.

ExecuteTarget updates the ItemIndex (see page 4541) property of all client controls that descend from TCustomListControl, and then generates an OnItemSelected (see page 4542) event, where your application can respond to the user's selection.

See Also

OnItemSelected (see page 4542)

ItemIndex (see page 4541)

42.1.1.3.2 TCustomListAction.GetCount Method

Pascal

```
function GetCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TCustomListAction.

42.1.1.3.3 TCustomListAction.GetString Method

Returns the value of the Strings (see page 4541) property.

Pascal

```
function GetString(Index: Integer): String; virtual;
```

C++

```
virtual __fastcall UnicodeString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings (see page 4541) property.

In TCustomListAction (see page 4535), GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings (see page 4541) property. For example, TStaticListAction (see page 4567) overrides GetString to return the caption of the specified item.

See Also

strings (see page 4541)

42.1.1.3.4 TCustomListAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true because the list action's ability to perform its function is not dependant on any particular target.

See Also

ExecuteTarget (see page 4537)

UpdateTarget

42.1.1.3.5 TCustomListAction.Loaded Method

Initializes the action after it is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded is called automatically when the form or data module containing the action has been completely loaded into memory. TCustomListAction (see page 4535) overrides the inherited method to set Active (see page 4540) and Images (see page 4540) properties now that the other components involved (the client controls whose items are set when Active (see page 4540) becomes true and the image list component that is the value of Images (see page 4540)) are known to be loaded into memory.

See Also

Loaded

Images (see page 4540)

Active (see page 4540)

42.1.1.3.6 TCustomListAction.SetString Method

Sets the value of the Strings (see page 4541) property.

Pascal

```
procedure SetString(Index: Integer; const Value: String); virtual;
```

C++

```
virtual __fastcall SetString(int Index, const UnicodeString Value);
```

Description

SetString is the protected write implementation of the Strings (see page 4541) property.

In TCustomListAction (see page 4535), SetString does nothing.

Override SetString to provide a more meaningful value to the Strings (see page 4541) property. For example, TStaticListAction (see page 4567) overrides SetString to set the caption of the specified item.

See Also

strings (see page 4541)

42.1.1.4 TCustomListAction Properties

The properties of the TCustomListAction class are listed here.

Protected Properties

	Name	Description
	Images (see page 4540)	Lists the images that the list action supplies along with items.
R	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Properties

	Name	Description
	Active (see page 4540)	Indicates whether the list action has supplied items to its client controls.
R	Count (see page 4540)	This is Count, a member of class TCustomListAction.
	ItemIndex (see page 4541)	Specifies the index of the selected item.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

42.1.1.4.1 TCustomListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count (see page 4540).

See Also

GetItem

getitem

Items

items

Items

count (see page 4540)

Items

strings (see page 4541)

42.1.1.4.2 TCustomListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomListAction.

42.1.1.4.3 TCustomListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages

Images

42.1.1.4.4 TCustomListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction (see page 4535) automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 4542)

ItemIndex

42.1.1.4.5 TCustomListAction.Loading Property

Indicates whether the action is currently adding items to client controls.

Pascal

```
property Loading: Boolean;
```

C++

```
__property Boolean Loading;
```

Description

Loading is used internally to detect when the list action is in the process of supplying client controls with items.

See Also

Active (see page 4540)

42.1.1.4.6 TCustomListAction.Strings Property

Pascal

```
property Strings [Index: Integer]: String;
```

C++

```
__property UnicodeString Strings[int Index];
```

Description

This is Strings, a member of class TCustomListAction.

42.1.1.5 TCustomListAction Events

The events of the TCustomListAction class are listed here.

Protected Events

	Name	Description
	OnGetItemCount (see page 4542)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 4542)	Occurs when a client control executes the action because a user selected one of the items.

42.1.1.5.1 TCustomListAction.OnGetItemCount Event

Occurs when the list action needs to determine the number of items it supplies to client controls.

Pascal

```
property OnGetItemCount: TGetItemCountEvent;
```

C++

```
__property TGetItemCountEvent OnGetItemCount;
```

Description

The GetCount (see page 4537) method generates an OnGetItemCount event to supply the value of the Count (see page 4540) property. Write an OnGetItemCount event handler to indicate the number of items for which you are supplying values in the OnGetItem event handler.

Sender is the list action that is supplying data to client controls.

Count (see page 4540) returns the number of items this list action supplies to client controls.

See Also

getcount (see page 4537)

count (see page 4540)

42.1.1.5.2 TCustomListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex (see page 4541) property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 4537)

ItemIndex (see page 4541)

42.1.2 TCustomStaticListAction Class

Class Hierarchy

Pascal

```
TCustomStaticListAction = class(TCustomListAction);
```

C++

```
class TCustomStaticListAction : public TCustomListAction;
```

File

ListActns

Description

This is class ListActns.TCustomStaticListAction.

42.1.2.1 TCustomStaticListAction Members

The following tables list the members exposed by TCustomStaticListAction.

Protected Events

	Name	Description
	OnGetItemCount (see page 4542)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 4542)	Occurs when a client control executes the action because a user selected one of the items.

Protected Methods

	Name	Description
V	GetCount (see page 4545)	This is GetCount, a member of class TCustomStaticListAction.
	GetItem (see page 4545)	This is GetItem, a member of class TCustomStaticListAction.
V	GetItemClass (see page 4545)	This is GetItemClass, a member of class TCustomStaticListAction.
V	GetString (see page 4546)	Returns the value of the Strings property.
V	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
V	SetString (see page 4546)	Sets the value of the Strings property.

Protected Properties

	Name	Description
	Clients (see page 4547)	This is Clients, a member of class TCustomStaticListAction.
	Images (see page 4540)	Lists the images that the list action supplies along with items.
R	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Events

	Name	Description
	OnGetItem (see page 4547)	This is OnGetItem, a member of class TCustomStaticListAction.

Public Methods

	Name	Description
	Create (see page 4544)	Creates an instance of TCustomListAction.
	Destroy (see page 4544)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
	Active (see page 4540)	Indicates whether the list action has supplied items to its client controls.
	Count (see page 4547)	This is Count, a member of class TCustomStaticListAction.
	ItemIndex (see page 4541)	Specifies the index of the selected item.
	Items (see page 4547)	This is Items, a member of class TCustomStaticListAction.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

42.1.2.2 TCustomStaticListAction.Create Constructor

Creates an instance of TCustomListAction.

Pascal

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

C++

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

Description

Most applications do not need to call the TCustomListAction constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomListAction instance.

See Also

Owner

42.1.2.3 TCustomStaticListAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStaticListAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

42.1.2.4 TCustomStaticListAction Methods

The methods of the TCustomStaticListAction class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 4545)	This is GetCount, a member of class TCustomStaticListAction.
	GetItem (see page 4545)	This is GetItem, a member of class TCustomStaticListAction.
	GetItemClass (see page 4545)	This is GetItemClass, a member of class TCustomStaticListAction.
	GetString (see page 4546)	Returns the value of the Strings property.
	SetString (see page 4546)	Sets the value of the Strings property.

42.1.2.4.1 TCustomStaticListAction.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TCustomStaticListAction.

42.1.2.4.2 TCustomStaticListAction.GetItem Method

Pascal

```
function GetItem(const Index: Integer; AnItem: TListControlItem): Boolean;
```

C++

```
__fastcall Boolean GetItem(const int Index, TListControlItem AnItem);
```

Description

This is GetItem, a member of class TCustomStaticListAction.

42.1.2.4.3 TCustomStaticListAction.GetItemClass Method

Pascal

```
function GetItemClass: TListControlItemClass; virtual;
```

C++

```
virtual __fastcall TListControlItemClass GetItemClass();
```

Description

This is GetItemClass, a member of class TCustomStaticListAction.

42.1.2.4.4 TCustomStaticListAction.GetString Method

Returns the value of the Strings property.

Pascal

```
function GetString(Index: Integer): String; override;
```

C++

```
virtual __fastcall UnicodeString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings property.

In TCustomListAction, GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides GetString to return the caption of the specified item.

See Also

strings (see page 4541)

42.1.2.4.5 TCustomStaticListAction.SetString Method

Sets the value of the Strings property.

Pascal

```
procedure SetString(Index: Integer; const Value: String); override;
```

C++

```
virtual __fastcall SetString(int Index, const UnicodeString Value);
```

Description

SetString is the protected write implementation of the Strings property.

In TCustomListAction, SetString does nothing.

Override SetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides SetString to set the caption of the specified item.

See Also

strings (see page 4541)

42.1.2.5 TCustomStaticListAction Properties

The properties of the TCustomStaticListAction class are listed here.

Protected Properties

	Name	Description
	Clients (see page 4547)	This is Clients, a member of class TCustomStaticListAction.

Public Properties

	Name	Description
	Count (see page 4547)	This is Count, a member of class TCustomStaticListAction.
	Items (see page 4547)	This is Items, a member of class TCustomStaticListAction.

42.1.2.5.1 TCustomStaticListAction.Clients Property

Pascal

```
property Clients;
```

C++

```
__property Clients;
```

Description

This is Clients, a member of class TCustomStaticListAction.

42.1.2.5.2 TCustomStaticListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomStaticListAction.

42.1.2.5.3 TCustomStaticListAction.Items Property

Pascal

```
property Items: TStaticListItems;
```

C++

```
__property TStaticListItems Items;
```

Description

This is Items, a member of class TCustomStaticListAction.

42.1.2.6 TCustomStaticListAction Events

The events of the TCustomStaticListAction class are listed here.

Public Events

	Name	Description
	OnGetItem (see page 4547)	This is OnGetItem, a member of class TCustomStaticListAction.

42.1.2.6.1 TCustomStaticListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetItemEvent;
```

C++

```
__property TGetItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TCustomStaticListAction.

42.1.3 TCustomVirtualListAction Class

Class Hierarchy

Pascal

```
TCustomVirtualListAction = class(TCustomListAction);
```

C++

```
class TCustomVirtualListAction : public TCustomListAction;
```

File

ListActns

Description

This is class ListActns.TCustomVirtualListAction.

42.1.3.1 TCustomVirtualListAction Members

The following tables list the members exposed by TCustomVirtualListAction.

Protected Events

	Name	Description
	OnGetItemCount (see page 4542)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 4542)	Occurs when a client control executes the action because a user selected one of the items.

Protected Methods

	Name	Description
V	GetCount (see page 4537)	This is GetCount, a member of class TCustomListAction.
	GetItem (see page 4549)	This is GetItem, a member of class TCustomVirtualListAction.
V	GetString (see page 4549)	Returns the value of the Strings property.
V	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
V	SetString (see page 4539)	Sets the value of the Strings (see page 4541) property.

Protected Properties

	Name	Description
	Images (see page 4540)	Lists the images that the list action supplies along with items.
R	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Events

	Name	Description
	OnGetItem (see page 4550)	This is OnGetItem, a member of class TCustomVirtualListAction.

Public Methods

	Name	Description
	Create (see page 4536)	Creates an instance of TCustomListAction (see page 4535).
	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
	Active (see page 4540)	Indicates whether the list action has supplied items to its client controls.
	Count (see page 4550)	This is Count, a member of class TCustomVirtualListAction.
	ItemIndex (see page 4541)	Specifies the index of the selected item.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

42.1.3.2 TCustomVirtualListAction Methods

The methods of the TCustomVirtualListAction class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 4549)	This is GetItem, a member of class TCustomVirtualListAction.
	GetString (see page 4549)	Returns the value of the Strings property.

42.1.3.2.1 TCustomVirtualListAction.GetItem Method

Pascal

```
function GetItem(const Index: Integer; var Value: String; var ImageIndex: Integer; var Data:
TCustomData): Boolean;
```

C++

```
__fastcall Boolean GetItem(const int Index, UnicodeString Value, int ImageIndex, TCustomData
Data);
```

Description

This is GetItem, a member of class TCustomVirtualListAction.

42.1.3.2.2 TCustomVirtualListAction.GetString Method

Returns the value of the Strings property.

Pascal

```
function GetString(Index: Integer): String; override;
```

C++

```
virtual __fastcall UnicodeString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings property.

In TCustomListAction, GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides GetString to return the caption of the specified item.

See Also

strings (see page 4541)

42.1.3.3 TCustomVirtualListAction Properties

The properties of the TCustomVirtualListAction class are listed here.

Public Properties

	Name	Description
	Count (see page 4550)	This is Count, a member of class TCustomVirtualListAction.

42.1.3.3.1 TCustomVirtualListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomVirtualListAction.

42.1.3.4 TCustomVirtualListAction Events

The events of the TCustomVirtualListAction class are listed here.

Public Events

	Name	Description
	OnGetItem (see page 4550)	This is OnGetItem, a member of class TCustomVirtualListAction.

42.1.3.4.1 TCustomVirtualListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetVirtualItemEvent;
```

C++

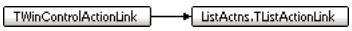
```
__property TGetVirtualItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TCustomVirtualListAction.

42.1.4 TListActionLink Class

Class Hierarchy



Pascal

```
TListActionLink = class(TWinControlActionLink);
```

C++

```
class TListActionLink : public TWinControlActionLink;
```

File

ListActns

Description

This is class ListActns.TListActionLink.

42.1.4.1 TListActionLink Members

The following tables list the members exposed by TListActionLink.

Protected Methods

	Name	Description
	AddItem (see page 4551)	This is the overview for the AddItem method overload.
	IsActiveLinked (see page 4552)	This is IsActiveLinked, a member of class TListActionLink.
	IsImagesLinked (see page 4552)	This is IsImagesLinked, a member of class TListActionLink.
	RefreshControl (see page 4553)	This is RefreshControl, a member of class TListActionLink.
	SetAction (see page 4553)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 4553)	This is SetActive, a member of class TListActionLink.
	SetImages (see page 4553)	This is SetImages, a member of class TListActionLink.
	SetItemIndex (see page 4553)	This is SetItemIndex, a member of class TListActionLink.

42.1.4.2 TListActionLink Methods

The methods of the TListActionLink class are listed here.

Protected Methods

	Name	Description
	AddItem (see page 4551)	This is the overview for the AddItem method overload.
	IsActiveLinked (see page 4552)	This is IsActiveLinked, a member of class TListActionLink.
	IsImagesLinked (see page 4552)	This is IsImagesLinked, a member of class TListActionLink.
	RefreshControl (see page 4553)	This is RefreshControl, a member of class TListActionLink.
	SetAction (see page 4553)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 4553)	This is SetActive, a member of class TListActionLink.
	SetImages (see page 4553)	This is SetImages, a member of class TListActionLink.
	SetItemIndex (see page 4553)	This is SetItemIndex, a member of class TListActionLink.

42.1.4.2.1 AddItem Method

This is the overview for the AddItem method overload.

Overload List

	Name	Description
	TListActionLink.AddItem (String, Integer, TCustomData) (see page 4552)	This is AddItem, a member of class TListActionLink.

	TListActionLink.AddItem (TListControlItem) (see page 4552)	This is AddItem, a member of class TListActionLink.

42.1.4.2.1.1 TListActionLink.AddItem Method (String, Integer, TCustomData)

Pascal

```
procedure AddItem(ACaption: String; AImageIndex: Integer; DataPtr: TCustomData); virtual;
overload;
```

C++

```
virtual __fastcall AddItem(UnicodeString ACaption, int AImageIndex, TCustomData DataPtr);
```

Description

This is AddItem, a member of class TListActionLink.

42.1.4.2.1.2 TListActionLink.AddItem Method (TListControlItem)

Pascal

```
procedure AddItem(AnItem: TListControlItem); virtual; overload;
```

C++

```
virtual __fastcall AddItem(TListControlItem AnItem);
```

Description

This is AddItem, a member of class TListActionLink.

42.1.4.2.2 TListActionLink.IsActiveLinked Method

Pascal

```
function IsActiveLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsActiveLinked();
```

Description

This is IsActiveLinked, a member of class TListActionLink.

42.1.4.2.3 TListActionLink.IsImagesLinked Method

Pascal

```
function IsImagesLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsImagesLinked();
```

Description

This is IsImagesLinked, a member of class TListActionLink.

42.1.4.2.4 TListActionLink.RefreshControl Method

Pascal

```
procedure RefreshControl;
```

C++

```
__fastcall RefreshControl();
```

Description

This is RefreshControl, a member of class TListActionLink.

42.1.4.2.5 TListActionLink.SetAction Method

Pascal

```
procedure SetAction(Value: TBasicAction); override;
```

C++

```
virtual __fastcall SetAction(TBasicAction Value);
```

Description

This is SetAction, a member of class TListActionLink.

42.1.4.2.6 TListActionLink.SetActive Method

Pascal

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

C++

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

Description

This is SetActive, a member of class TListActionLink.

42.1.4.2.7 TListActionLink.SetImages Method

Pascal

```
procedure SetImages(Value: TCustomImageList); virtual;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

This is SetImages, a member of class TListActionLink.

42.1.4.2.8 TListActionLink.SetItemIndex Method

Pascal

```
procedure SetItemIndex(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

This is SetItemIndex, a member of class TListActionLink.

42.1.5 TListControlItem Class

TListControlItem represents an item with a text value, an image, and a data pointer.

Class Hierarchy

Pascal

```
TListControlItem = class(TCollectionItem);
```

C++

```
class TListControlItem : public TCollectionItem;
```

File

ListActns

Description

TListControlItem represents a single item in a control that lists items. TListControlItem objects are used by TStaticListAction (see page 4567) to represent the items in a variety of client controls such as list views, list boxes, and combo boxes. In addition, TListControlItem is the base class for TComboExItem, which is specialized to work with the TComboBoxEx control.

Each TListControlItem has

A text value, which it stores as the Caption (see page 4559) property.

An associated image, which is identified by the ImageIndex (see page 4560) property.

A data pointer, which it stores as the Data (see page 4560) property.

See Also

TListControlItems (see page 4560)

TComboExItem

42.1.5.1 TListControlItem Members

The following tables list the members exposed by TListControlItem.

Protected Fields

	Name	Description
	FCaption (see page 4555)	This is FCaption, a member of class TListControlItem.
	FData (see page 4556)	This is FData, a member of class TListControlItem.
	FImageIndex (see page 4556)	This is FImageIndex, a member of class TListControlItem.

Protected Methods

	Name	Description
	Changed (see page 4557)	Responds when any of the item's properties changes.
	GetDisplayName (see page 4558)	Specifies the name of the item as it appears in the Object Inspector.
	SetCaption (see page 4558)	Sets the value of the Caption (see page 4559) property.

	SetData (see page 4558)	Sets the value of the Data (see page 4560) property.
	SetImageIndex (see page 4559)	Sets the value of the ImageIndex (see page 4560) property.

Public Methods

	Name	Description
	Assign (see page 4556)	Copies the contents of another, similar object.
	Create (see page 4555)	Creates and initializes a TListControlItem (see page 4554) instance.

Public Properties

	Name	Description
	Data (see page 4560)	Points to user-defined data associated with the control.

Published Properties

	Name	Description
	Caption (see page 4559)	Specifies the text value of the item.
	ImageIndex (see page 4560)	Identifies the image associated with this item.

42.1.5.2 TListControlItem.Create Constructor

Creates and initializes a TListControlItem (see page 4554) instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TListControlItem(TCollection Collection);
```

Description

Do not call the constructor to create a TListControlItem (see page 4554) instance. Instead, use the collection's Add method to create a new list control item.

Collection is the collection object that owns and manages this list control item.

See Also

Add

42.1.5.3 TListControlItem Fields

The fields of the TListControlItem class are listed here.

Protected Fields

	Name	Description
	FCaption (see page 4555)	This is FCaption, a member of class TListControlItem.
	FData (see page 4556)	This is FData, a member of class TListControlItem.
	FImageIndex (see page 4556)	This is FImageIndex, a member of class TListControlItem.

42.1.5.3.1 TListControlItem.FCaption Field

Pascal

```
FCaption: String;
```

C++

```
UnicodeString FCaption;
```

Description

This is FCaption, a member of class TListControlItem.

42.1.5.3.2 TListControlItem.FData Field

Pascal

```
FData: TCustomData;
```

C++

```
TCustomData FData;
```

Description

This is FData, a member of class TListControlItem.

42.1.5.3.3 TListControlItem.FImageIndex Field

Pascal

```
FImageIndex: TImageIndex;
```

C++

```
TImageIndex FImageIndex;
```

Description

This is FImageIndex, a member of class TListControlItem.

42.1.5.4 TListControlItem Methods

The methods of the TListControlItem class are listed here.

Protected Methods

	Name	Description
	Changed (see page 4557)	Responds when any of the item's properties changes.
	GetDisplayName (see page 4558)	Specifies the name of the item as it appears in the Object Inspector.
	SetCaption (see page 4558)	Sets the value of the Caption (see page 4559) property.
	SetData (see page 4558)	Sets the value of the Data (see page 4560) property.
	SetImageIndex (see page 4559)	Sets the value of the ImageIndex (see page 4560) property.

Public Methods

	Name	Description
	Assign (see page 4556)	Copies the contents of another, similar object.

42.1.5.4.1 TListControlItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

42.1.5.4.2 TListControlItem.Changed Method

Responds when any of the item's properties changes.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is called automatically when the item's properties change. It calls the Update method of the corresponding TListControlItems (see page 4560) object, so that the collection can perform any necessary adjustments.

See Also

Update

42.1.5.4.3 TListControlItem.GetDisplayName Method

Specifies the name of the item as it appears in the Object Inspector.

Pascal

```
function GetDisplayName: String; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName returns the string that represents the item in the Collection editor and the Object Inspector.

GetDisplayName returns the value of the Caption (see page 4559) if it is set. Otherwise, it returns the class name of the TListControlItem (see page 4554) descendant for which this is an instance, with a number added to ensure uniqueness.

See Also

Caption (see page 4559)

42.1.5.4.4 TListControlItem.SetCaption Method

Sets the value of the Caption (see page 4559) property.

Pascal

```
procedure SetCaption(const Value: String); virtual;
```

C++

```
virtual __fastcall SetCaption(const UnicodeString Value);
```

Description

SetCaption is the protected write implementation of the Caption (see page 4559) property. It sets the Caption (see page 4559) property to Value, and then calls the Changed (see page 4557) method so that the collection can update itself.

See Also

Caption (see page 4559)

Changed (see page 4557)

42.1.5.4.5 TListControlItem.SetData Method

Sets the value of the Data (see page 4560) property.

Pascal

```
procedure SetData(const Value: TCustomData); virtual;
```

C++

```
virtual __fastcall SetData(const TCustomData Value);
```

Description

SetData is the protected write implementation of the Data (see page 4560) property. It sets the Data (see page 4560) property to Value, and then calls the Changed (see page 4557) method so that the collection can update itself.

See Also

Data (see page 4560)

Changed (see page 4557)

42.1.5.4.6 TListControlItem.SetImageIndex Method

Sets the value of the ImageIndex (see page 4560) property.

Pascal

```
procedure SetImageIndex(const Value: TImageIndex); virtual;
```

C++

```
virtual __fastcall SetImageIndex(const TImageIndex Value);
```

Description

SetImageIndex is the protected write implementation of the ImageIndex (see page 4560) property. It sets the ImageIndex (see page 4560) property to Value, and then calls the Changed (see page 4557) method so that the collection can update itself.

See Also

ImageIndex (see page 4560)

Changed (see page 4557)

42.1.5.5 TListControlItem Properties

The properties of the TListControlItem class are listed here.

Public Properties

	Name	Description
	Data (see page 4560)	Points to user-defined data associated with the control.

Published Properties

	Name	Description
	Caption (see page 4559)	Specifies the text value of the item.
	ImageIndex (see page 4560)	Identifies the image associated with this item.

42.1.5.5.1 TListControlItem.Caption Property

Specifies the text value of the item.

Pascal

```
property Caption: String;
```

C++

```
__property UnicodeString Caption;
```

Description

Set Caption to provide the item with a text value. This is the value that appears in list boxes or combo boxes, or which labels items in list view controls.

See Also

ImageIndex (see page 4560)

Data (see page 4560)

42.1.5.5.2 TListControlItem.Data Property

Points to user-defined data associated with the control.

Pascal

```
property Data: TCustomData;
```

C++

```
__property TCustomData Data;
```

Description

Set Data to associate this item with application-specific data. Data has no predefined meaning, although it is used for sorting items when the managing collection's SortType property is stData.

See Also

SortType (see page 4566)

ImageIndex (see page 4560)

Caption (see page 4559)

42.1.5.5.3 TListControlItem.ImageIndex Property

Identifies the image associated with this item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to associate this item with an image. ImageIndex is an index into the Images property of the component that uses this TListControlItem (see page 4554) object to represent one of its items.

See Also

Images (see page 4540)

Images

42.1.6 TListControlItems Class

TListControlItems manages a collection of TListControlItem (see page 4554) objects.

Class Hierarchy

Pascal

```
TListControlItems = class(TOwnedCollection);
```

C++

```
class TListControlItems : public TOwnedCollection;
```

File

ListActns

Description

TListControlItems manages a collection of items that each include a text value, associated image, and a data pointer.

Use the properties and methods of TListControlItems to:

Access a specific item.

Add (see page 4562) or delete items from the collection.

Find out how many items are in the collection.

Sort (see page 4564) the items in the collection.

Although TListControlItems can be used on its own, it also acts as a base class to classes that manage the items for a specific component. TListControlItems descendants include TComboExItems, which manages the items of a TComboBoxEx control, and TStaticListItems (see page 4576), which manages items for a list action component.

See Also

TListControlItem (see page 4554)

TStaticListItems (see page 4576)

TComboExItems

42.1.6.1 TListControlItems Members

The following tables list the members exposed by TListControlItems.

Protected Methods

	Name	Description
	CompareItems (see page 4563)	Returns the relative order of two items in the collection.
	Update (see page 4564)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4562)	Adds a new item to the collection.
	Create (see page 4562)	Creates and initializes a TOwnedCollection instance.
	CustomSort (see page 4563)	Sorts the items in the collection using a specified comparison function.
	Sort (see page 4564)	Sorts the items in the collection.

Public Properties

	Name	Description
	Items (see page 4565)	Lists the items in the collection.

Published Events

	Name	Description
	OnCompare (see page 4567)	Occurs to when two items need to be compared for sorting the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 4565)	Controls whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 4566)	Indicates how the items should be sorted.

42.1.6.2 TListControlItems.Create Constructor

Creates and initializes a TOwnedCollection instance.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass); virtual;
```

C++

```
virtual __fastcall TListControlItems(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes two parameters: the name of an instance object descended from TPersistent and the name of a TCollectionItem descendant class. The first parameter is the owner of the TOwnedCollection instance. The second parameter determines the class of the items created by the Add method.

See Also

Add

42.1.6.3 TListControlItems Methods

The methods of the TListControlItems class are listed here.

Protected Methods

	Name	Description
	CompareItems (see page 4563)	Returns the relative order of two items in the collection.
	Update (see page 4564)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4562)	Adds a new item to the collection.
	CustomSort (see page 4563)	Sorts the items in the collection using a specified comparison function.
	Sort (see page 4564)	Sorts the items in the collection.

42.1.6.3.1 TListControlItems.Add Method

Adds a new item to the collection.

Pascal

```
function Add: TListControlItem;
```

C++

```
__fastcall TListControlItem Add();
```

Description

Call Add to add a new item to the collection. Add instantiates an object of the type that was specified when the TListControlItems (see page 4560) instance was created. It returns the newly created object.

See Also

Items (see page 4565)

Delete

42.1.6.3.2 TListControlItems.CompareItems Method

Returns the relative order of two items in the collection.

Pascal

```
function CompareItems(I1: TListControlItem; I2: TListControlItem): Integer; virtual;
```

C++

```
virtual __fastcall int CompareItems(TListControlItem I1, TListControlItem I2);
```

Description

CompareItems is used internally to provide the default sort order for items in the list. It is used when the SortType (see page 4566) property is not stNone. The Sort (see page 4564) method also uses this as a comparison function.

If I1 is greater than I2, CompareItems returns a value greater than 0. If I1 is the same as I2, CompareItems returns 0. If I1 is less than I2, CompareItems returns a value less than 0.

CompareItems always uses an OnCompare (see page 4567) event handler if it is assigned. Otherwise, it compares the captions of the items, taking into account the CaseSensitive (see page 4565) property. Descendant classes can override CompareItems to provide a different default sort order.

See Also

SortType (see page 4566)

CustomSort (see page 4563)

Sort (see page 4564)

CaseSensitive (see page 4565)

OnCompare (see page 4567)

42.1.6.3.3 TListControlItems.CustomSort Method

Sorts the items in the collection using a specified comparison function.

Pascal

```
procedure CustomSort(Compare: TListItemsCompare);
```

C++

```
__fastcall CustomSort(TListItemsCompare Compare);
```

Description

Call Sort (see page 4564) to sort the items in the collection based on the comparison function that is the value of the Compare parameter.

Compare is a function that compares two items in the list. Its List parameter is this TListControlItems (see page 4560) object. The Index1 and Index2 parameters are the indexes of the items to compare in the Items (see page 4565) property array. This comparison function should return a value less than 0 if the first item is less than the second, a value greater than 0 if it is greater than the second, and 0 if the two items are equivalent.

Note: CustomSort sorts the items in the list, but does not maintain the sort order when new items are added. To provide a custom sort order that persists when new items are added, set the SortType (see page 4566) property to stData and supply an OnCompare (see page 4567) event handler.

See Also

SortType (see page 4566)

Sort (see page 4564)

OnCompare (see page 4567)

Items (see page 4565)

42.1.6.3.4 TListControlItems.Sort Method

Sorts the items in the collection.

Pascal

```
procedure Sort;
```

C++

```
__fastcall Sort();
```

Description

Call Sort to sort the items in the collection. Sort uses the comparison function specified by the protected CompareItems (see page 4563) method to determine the sort order.

Note: Sort sorts the items in the list, but does not maintain the sort order when new items are added. To impose a sort order that persists when new items are added, set the SortType (see page 4566) property to a value other than stNone and, if desired, supply an OnCompare (see page 4567) event handler.

See Also

SortType (see page 4566)

CustomSort (see page 4563)

OnCompare (see page 4567)

Items (see page 4565)

CompareItems (see page 4563)

42.1.6.3.5 TListControlItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed

Changed

42.1.6.4 TListControlItems Properties

The properties of the TListControlItems class are listed here.

Public Properties

	Name	Description
	Items (see page 4565)	Lists the items in the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 4565)	Controls whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 4566)	Indicates how the items should be sorted.

42.1.6.4.1 TListControlItems.CaseSensitive Property

Controls whether items are sorted in a case-sensitive or case-insensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether the case of item captions should be considered when sorting the items in the list.

See Also

Items (see page 4565)

SortType (see page 4566)

Caption (see page 4559)

42.1.6.4.2 TListControlItems.Items Property

Lists the items in the collection.

Pascal

```
property Items [const Index: Integer]: TListControlItem;
```

C++

```
__property TListControlItem Items[int const Index];
```

Description

Use Items to access the items in this collection.

Index indicates the item to access, where 0 is the index of the first item, 1 is the index of the second item, and so on. The Count property indicates the maximum number of items.

Note: In Delphi code, Items is the default property for TListControlItems (see page 4560). This means that the property name can be omitted when using the Items property. For example,

```
StaticListAction.Items.Items[0].Caption := Edit1.Text;
```

Note: can also be written

```
StaticListAction.Items[0].Caption := Edit1.Text;
```

Note: Items is a read-only property. To add or delete items from the collection, use the Add (see page 4562) and Delete methods.

See Also

Add (see page 4562)

Delete

Count

operator_[]

42.1.6.4.3 TListControlItems.SortType Property

Indicates how the items should be sorted.

Pascal

```
property SortType: TListItemsSortType;
```

C++

```
__property TListItemsSortType SortType;
```

Description

Use SortType to indicate how you want the items sorted, if at all. The following table lists the possible values:

Value	Meaning
stNone	Items (see page 4565) are not automatically sorted. You can still sort the items explicitly by calling the Sort (see page 4564) or CustomSort (see page 4563) method.
stData	Items (see page 4565) are sorted based on the value of their Data property. You must supply an OnCompare (see page 4567) event handler to specify the sort order.
stText	Items (see page 4565) are sorted base on the value of their Caption property. The CaseSensitive (see page 4565) property specifies whether comparisons are case-sensitive.
stBoth	Items (see page 4565) are sorted based on an OnCompare (see page 4567) event handler if one is assigned, otherwise based on the value of their Caption property.

See Also

Items (see page 4565)

OnCompare (see page 4567)

Data (see page 4560)

CaseSensitive (see page 4565)

CustomSort (see page 4563)

Sort (see page 4564)

42.1.6.5 TListControlItems Events

The events of the TListControlItems class are listed here.

Published Events

	Name	Description
	OnCompare (see page 4567)	Occurs to when two items need to be compared for sorting the collection.

42.1.6.5.1 TListControlItems.OnCompare Event

Occurs to when two items need to be compared for sorting the collection.

Pascal

```
property OnCompare: TListCompareEvent;
```

C++

```
__property TListCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to indicate the sort order when the SortType (see page 4566) property is stData or stBoth.

The OnCompare event handler compares the items passed as the Item1 and Item2 parameters. If Item1 is the same as Item2 in the sort order, the event handler should return 0. If Item1 is less than Item2, the event handler should return a value less than 0. If Item1 is greater than Item2, the event handler should return value greater than 0.

See Also

SortType (see page 4566)

42.1.7 TStaticListAction Class

TStaticListAction provides items to client controls.

Class Hierarchy

Pascal

```
TStaticListAction = class(TCustomStaticListAction);
```

C++

```
class TStaticListAction : public TCustomStaticListAction;
```

File

ListActns

Description

TStaticListAction supplies items to client controls such as list views, list boxes, or combo boxes. TStaticListAction maintains a list of items in its Items (see page 4572) property. This lets you specify, at design time, the items it supplies to clients.

When an application sets the Active (see page 4569) property of a list action to true, the action fills all client controls with

items. Each item may include text, an image, and a pointer to data. Each client control determines which of these features of the item it accepts.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 4574) event.

When a TStaticListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TVirtualListAction (see page 4578)

TCustomCombo

TCustomListView

TCustomListBox

TCustomComboBox

42.1.7.1 TStaticListAction Members

The following tables list the members exposed by TStaticListAction.

Protected Events

	Name	Description
	OnGetItemCount (see page 4542)	Occurs when the list action needs to determine the number of items it supplies to client controls.

Protected Methods

	Name	Description
V	GetCount (see page 4537)	This is GetCount, a member of class TCustomListAction.
V	GetCount (see page 4545)	This is GetCount, a member of class TCustomStaticListAction.
	GetItem (see page 4545)	This is GetItem, a member of class TCustomStaticListAction.
V	GetItemClass (see page 4545)	This is GetItemClass, a member of class TCustomStaticListAction.
V	GetString (see page 4538)	Returns the value of the Strings (see page 4541) property.
V	GetString (see page 4546)	Returns the value of the Strings property.
V	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
V	SetString (see page 4539)	Sets the value of the Strings (see page 4541) property.
V	SetString (see page 4546)	Sets the value of the Strings property.

Protected Properties

	Name	Description
	Clients (see page 4547)	This is Clients, a member of class TCustomStaticListAction.
R	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Methods

	Name	Description
V	Create (see page 4536)	Creates an instance of TCustomListAction (see page 4535).
V	Create (see page 4544)	Creates an instance of TCustomListAction.
V	Destroy (see page 4544)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
V	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
R	Count (see page 4540)	This is Count, a member of class TCustomListAction.

	Count (see page 4547)	This is Count, a member of class TCustomStaticListAction.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

Published Events

	Name	Description
	OnGetItem (see page 4574)	This is OnGetItem, a member of class TStaticListAction.
	OnHint (see page 4574)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 4574)	Occurs when a client control executes the action because a user selected one of the items.
	OnUpdate (see page 4575)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Active (see page 4569)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 4570)	Represents the caption of client controls and menu items.
	Enabled (see page 4570)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 4571)	Indicates the help context ID for client controls and menu items.
	Hint (see page 4571)	Indicates the Help hint for client controls and menu items.
	Images (see page 4571)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 4572)	Specifies the index of the selected item.
	Items (see page 4572)	Lists the items that this list action supplies to clients.
	SecondaryShortCuts (see page 4572)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 4573)	Specifies the ShortCut property for client menu items.
	Visible (see page 4573)	Specifies the Visible property for client controls and menu items.

42.1.7.2 TStaticListAction Properties

The properties of the TStaticListAction class are listed here.

Published Properties

	Name	Description
	Active (see page 4569)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 4570)	Represents the caption of client controls and menu items.
	Enabled (see page 4570)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 4571)	Indicates the help context ID for client controls and menu items.
	Hint (see page 4571)	Indicates the Help hint for client controls and menu items.
	Images (see page 4571)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 4572)	Specifies the index of the selected item.
	Items (see page 4572)	Lists the items that this list action supplies to clients.
	SecondaryShortCuts (see page 4572)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 4573)	Specifies the ShortCut property for client menu items.
	Visible (see page 4573)	Specifies the Visible property for client controls and menu items.

42.1.7.2.1 TStaticListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count.

See Also

GetItem

getitem

Items

items

Items

count (see page 4540)

Items

strings (see page 4541)

42.1.7.2.2 TStaticListAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

42.1.7.2.3 TStaticListAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

42.1.7.2.4 TStaticListAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

42.1.7.2.5 TStaticListAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

42.1.7.2.6 TStaticListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text

value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages

Images

42.1.7.2.7 TStaticListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 4542)

ItemIndex

42.1.7.2.8 TStaticListAction.Items Property

Lists the items that this list action supplies to clients.

Pascal

```
property Items: TStaticListItems;
```

C++

```
__property TStaticListItems Items;
```

Description

Set Items to indicate the items that this action supplies to clients. At design time, you can use the collection editor to add items to the list and set their properties.

See Also

Count

strings

42.1.7.2.9 TStaticListAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

42.1.7.2.10 TStaticListAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

42.1.7.2.11 TStaticListAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

42.1.7.3 TStaticListAction Events

The events of the TStaticListAction class are listed here.

Published Events

	Name	Description
	OnGetItem (see page 4574)	This is OnGetItem, a member of class TStaticListAction.
	OnHint (see page 4574)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 4574)	Occurs when a client control executes the action because a user selected one of the items.
	OnUpdate (see page 4575)	Occurs when the application is idle or when the action list updates.

42.1.7.3.1 TStaticListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetItemEvent;
```

C++

```
__property TGetItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TStaticListAction.

42.1.7.3.2 TStaticListAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

42.1.7.3.3 TStaticListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 4537)

ItemIndex (see page 4541)

42.1.7.3.4 TStaticListAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
```

```
  applications to execute centralized code for enabling and
  disabling, checking and unchecking, and so on. For example,
  the following code illustrates the OnUpdate event handler
  for an action that is "checked" when the toolbar is visible.
  This example requires a TActionList (with one TAction) and a
  TToolBar on a form. This example requires a TActionList
  (with one TAction) and a TToolBar on a form.  Double click
  on the TActionList and create an TAction named Action1.  The
  TAction has an OnUpdate event handler called Action1Update.
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

42.1.8 TStaticListItems Class

TStaticListItems represents the items that TStaticListAction (see page 4567) supplies to client controls.

Class Hierarchy

Pascal

```
TStaticListItems = class(TListControlItems);
```

C++

```
class TStaticListItems : public TListControlItems;
```

File

ListActns

Description

TStaticListItems manages a collection of items for a TStaticListAction (see page 4567) component. Each item in the collection can include a text value, associated image, and a data pointer. If the associated TStaticListAction (see page 4567) is active, then when items are added to or deleted from TStaticListItems, the list action's clients are updated so that they reflect the current set of items.

See Also

TStaticListAction (see page 4567)

42.1.8.1 TStaticListItems Members

The following tables list the members exposed by TStaticListItems.

Protected Methods

	Name	Description
	CompareItems (see page 4563)	Returns the relative order of two items in the collection.
	Notify (see page 4577)	Responds when items are added to or removed from the collection.
	Update (see page 4578)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4562)	Adds a new item to the collection.
	Create (see page 4562)	Creates and initializes a TOwnedCollection instance.
	CustomSort (see page 4563)	Sorts the items in the collection using a specified comparison function.
	Sort (see page 4564)	Sorts the items in the collection.

Public Properties

	Name	Description
	Items (see page 4565)	Lists the items in the collection.

Published Events

	Name	Description
	OnCompare (see page 4567)	Occurs to when two items need to be compared for sorting the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 4565)	Controls whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 4566)	Indicates how the items should be sorted.

42.1.8.2 TStaticListItems Methods

The methods of the TStaticListItems class are listed here.

Protected Methods

	Name	Description
	Notify (see page 4577)	Responds when items are added to or removed from the collection.
	Update (see page 4578)	Updates the collection to reflect changes to its items.

42.1.8.2.1 TStaticListItems.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added

Deleting

42.1.8.2.2 TStaticListItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed

Changed

42.1.9 TVirtualListAction Class

TVirtualListAction provides items to a client controls.

Class Hierarchy

Pascal

```
TVirtualListAction = class(TCustomVirtualListAction);
```

C++

```
class TVirtualListAction : public TCustomVirtualListAction;
```

File

ListActns

Description

TVirtualListAction supplies items dynamically at runtime to client controls such as list views, list boxes, or combo boxes. These items are supplied by the code you supply to an OnGetItem (see page 4584) event handler.

When an application sets the Active (see page 4580) property of a list action to true, the action fills all client controls with items. Each item may include text, an image, and a pointer to data. Each client control determines which of these features of the

item it accepts.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 4585) event.

When a TVirtualListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TStaticListAction (see page 4567)

TCustomCombo

TCustomListView

TCustomListBox

TCustomComboBox

42.1.9.1 TVirtualListAction Members

The following tables list the members exposed by TVirtualListAction.

Protected Methods

	Name	Description
	GetCount (see page 4537)	This is GetCount, a member of class TCustomListAction.
	GetItem (see page 4549)	This is GetItem, a member of class TCustomVirtualListAction.
	GetString (see page 4538)	Returns the value of the Strings (see page 4541) property.
	GetString (see page 4549)	Returns the value of the Strings property.
	Loaded (see page 4538)	Initializes the action after it is loaded into memory.
	SetString (see page 4539)	Sets the value of the Strings (see page 4541) property.

Protected Properties

	Name	Description
	Loading (see page 4541)	Indicates whether the action is currently adding items to client controls.

Public Methods

	Name	Description
	Create (see page 4536)	Creates an instance of TCustomListAction (see page 4535).
	ExecuteTarget (see page 4537)	Generates an OnItemSelected (see page 4542) event.
	HandlesTarget (see page 4538)	Indicates whether the action can execute its function.

Public Properties

	Name	Description
	Count (see page 4540)	This is Count, a member of class TCustomListAction.
	Count (see page 4550)	This is Count, a member of class TCustomVirtualListAction.
	Strings (see page 4541)	This is Strings, a member of class TCustomListAction.

Published Events

	Name	Description
	OnGetItem (see page 4584)	This is OnGetItem, a member of class TVirtualListAction.
	OnGetItemCount (see page 4584)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnHint (see page 4585)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 4585)	Occurs when a client control executes the action because a user selected one of the items.

Published Properties

	Name	Description
	Active (see page 4580)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 4581)	Represents the caption of client controls and menu items.
	Enabled (see page 4581)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 4581)	Indicates the help context ID for client controls and menu items.
	Hint (see page 4582)	Indicates the Help hint for client controls and menu items.
	Images (see page 4582)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 4582)	Specifies the index of the selected item.
	SecondaryShortCuts (see page 4583)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 4583)	Specifies the ShortCut property for client menu items.
	Visible (see page 4584)	Specifies the Visible property for client controls and menu items.

42.1.9.2 TVirtualListAction Properties

The properties of the TVirtualListAction class are listed here.

Published Properties

	Name	Description
	Active (see page 4580)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 4581)	Represents the caption of client controls and menu items.
	Enabled (see page 4581)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 4581)	Indicates the help context ID for client controls and menu items.
	Hint (see page 4582)	Indicates the Help hint for client controls and menu items.
	Images (see page 4582)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 4582)	Specifies the index of the selected item.
	SecondaryShortCuts (see page 4583)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 4583)	Specifies the ShortCut property for client menu items.
	Visible (see page 4584)	Specifies the Visible property for client controls and menu items.

42.1.9.2.1 TVirtualListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count.

See Also

GetItem

getitem

Items

items

Items

count (see page 4540)

Items

strings (see page 4541)

42.1.9.2.2 TVirtualListAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption (see page 4902)

42.1.9.2.3 TVirtualListAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled (see page 4912)

42.1.9.2.4 TVirtualListAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

42.1.9.2.5 TVirtualListAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint (see page 4916)

42.1.9.2.6 TVirtualListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages

Images

42.1.9.2.7 TVirtualListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 4542)

ItemIndex

42.1.9.2.8 TVirtualListAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

42.1.9.2.9 TVirtualListAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 4920)

SecondaryShortCuts

42.1.9.2.10 TVirtualListAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 4921)

Visible

42.1.9.3 TVirtualListAction Events

The events of the TVirtualListAction class are listed here.

Published Events

	Name	Description
	OnGetItem (see page 4584)	This is OnGetItem, a member of class TVirtualListAction.
	OnGetItemCount (see page 4584)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnHint (see page 4585)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 4585)	Occurs when a client control executes the action because a user selected one of the items.

42.1.9.3.1 TVirtualListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetVirtualItemEvent;
```

C++

```
__property TGetVirtualItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TVirtualListAction.

42.1.9.3.2 TVirtualListAction.OnGetItemCount Event

Occurs when the list action needs to determine the number of items it supplies to client controls.

Pascal

```
property OnGetItemCount: TGetItemCountEvent;
```

C++

```
__property TGetItemCountEvent OnGetItemCount;
```

Description

The GetCount method generates an OnGetItemCount event to supply the value of the Count property. Write an OnGetItemCount event handler to indicate the number of items for which you are supplying values in the OnGetItem event handler.

Sender is the list action that is supplying data to client controls.

Count returns the number of items this list action supplies to client controls.

See Also

getcount (see page 4537)

count (see page 4540)

42.1.9.3.3 TVirtualListAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

42.1.9.3.4 TVirtualListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 4537)

ItemIndex (see page 4541)

42.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TListItemsSortType (see page 4586)	This is record ListActns.TListItemsSortType.

42.2.1 ListActns.TListItemsSortType Enumeration

Pascal

```
TListItemsSortType = (
  stNone,
  stData,
  stText,
  stBoth
);
```

C++

```
enum TListItemsSortType {
  stNone,
  stData,
  stText,
  stBoth
};
```

File

ListActns

Description

This is record ListActns.TListItemsSortType.

42.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetItemCountEvent (see page 4587)	This is type ListActns.TGetItemCountEvent.
TGetItemEvent (see page 4587)	This is type ListActns.TGetItemEvent.
TGetVirtualItemEvent (see page 4588)	This is type ListActns.TGetVirtualItemEvent.
TItemSelectedEvent (see page 4588)	This is type ListActns.TItemSelectedEvent.
TListCompareEvent (see page 4588)	This is type ListActns.TListCompareEvent.
TListControlItemClass (see page 4589)	This is type ListActns.TListControlItemClass.
TListItemsCompare (see page 4589)	This is type ListActns.TListItemsCompare.

42.3.1 ListActns.TGetItemCountEvent Type

Pascal

```
TGetItemCountEvent = procedure (Sender: TCustomListAction; var Count: Integer) of object;
```

C++

```
(Sender: TCustomListAction; var Count: Integer) ( TGetItemCountEvent)();
```

File

ListActns

Description

This is type ListActns.TGetItemCountEvent.

42.3.2 ListActns.TGetItemEvent Type

Pascal

```
TGetItemEvent = procedure (Sender: TCustomListAction; const Index: Integer; var Item:
TListControlItem) of object;
```

C++

```
(Sender: TCustomListAction; const Index: Integer; var Item: TListControlItem) (
TGetItemEvent)();
```

File

ListActns

Description

This is type ListActns.TGetItemEvent.

42.3.3 ListActns.TGetVirtualItemEvent Type

Pascal

```
TGetVirtualItemEvent = procedure (Sender: TCustomListAction; const Index: Integer; var Value:
String; var ImageIndex: Integer; var Data: TCustomData) of object;
```

C++

```
(Sender: TCustomListAction; const Index: Integer; var Value: String; var ImageIndex: Integer;
var Data: TCustomData) ( TGetVirtualItemEvent)();
```

File

ListActns

Description

This is type ListActns.TGetVirtualItemEvent.

42.3.4 ListActns.TItemSelectedEvent Type

Pascal

```
TItemSelectedEvent = procedure (Sender: TCustomListAction; Control: TControl) of object;
```

C++

```
(Sender: TCustomListAction; Control: TControl) ( TItemSelectedEvent)();
```

File

ListActns

Description

This is type ListActns.TItemSelectedEvent.

42.3.5 ListActns.TListCompareEvent Type

Pascal

```
TListCompareEvent = function (List: TListControlItems; Item1, Item2: TListControlItem):
Integer of object;
```

C++

```
(List: TListControlItems; Item1, Item2: TListControlItem): Integer ( TListCompareEvent)();
```

File

ListActns

Description

This is type ListActns.TListCompareEvent.

42.3.6 ListActns.TListControlItemClass Type

Pascal

```
TListControlItemClass = class of TListControlItem;
```

C++

```
TListControlItem TListControlItemClass;
```

File

ListActns

Description

This is type ListActns.TListControlItemClass.

42.3.7 ListActns.TListItemsCompare Type

Pascal

```
TListItemsCompare = function (List: TListControlItems; Index1, Index2: Integer): Integer;
```

C++

```
(List: TListControlItems; Index1, Index2: Integer): Integer ( TListItemsCompare)();
```

File

ListActns

Description

This is type ListActns.TListItemsCompare.

43 Mask Namespace

This is namespace Mask.

Topics

Name	Description
Classes (see page 4592)	The following table lists classes in this documentation.
Types (see page 4681)	The following table lists types in this documentation.

Classes

	Name	Description
	EDBEditError (see page 4592)	EDBEditError is the exception class for data that are incompatible with a specified mask.
	TCustomMaskEdit (see page 4592)	TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.
	TMaskEdit (see page 4610)	TMaskEdit implements a generic masked edit control.

Types

Name	Description
TMaskedState (see page 4681)	TMaskedState is a set of flags that describe the current state of a masked edit control.

43.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDBEditError (see page 4592)	EDBEditError is the exception class for data that are incompatible with a specified mask.
	TCustomMaskEdit (see page 4592)	TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.
	TMaskEdit (see page 4610)	TMaskEdit implements a generic masked edit control.

43.1.1 EDBEditError Class

EDBEditError is the exception class for data that are incompatible with a specified mask.

Class Hierarchy

Pascal

```
EDBEditError = class(Exception);
```

C++

```
class EDBEditError : public Exception;
```

File

Mask

Description

EDBEditError is raised when an application attempts to use data that are incompatible with the mask specified for a field.

A mask restricts the data that can be put into an edit box or data field. Use the EditMask property to specify a mask.

See Also

TCustomMaskEdit (see page 4592)

EditMask

43.1.2 TCustomMaskEdit Class

TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.

Class Hierarchy

Pascal

```
TCustomMaskEdit = class(TCustomEdit);
```

C++

```
class TCustomMaskEdit : public TCustomEdit;
```

File

Mask

Description

TCustomMaskEdit introduces new properties and methods to expand the capabilities of TCustomEdit for validity checking using an edit mask. To create an edit control that takes advantage of the masking capabilities of TCustomMaskEdit, use a TMaskEdit (see page 4610) object. To create an edit control that does not validate the input, use the TEdit object instead.

Do not create instances of TCustomMaskEdit. Use TCustomMaskEdit as a base class when declaring control objects that implement a masked edit control. Properties and methods of TCustomMaskEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

EDBEditError (see page 4592)

TDBEdit

TEdit

TMaskEdit (see page 4610)

43.1.2.1 TCustomMaskEdit Members

The following tables list the members exposed by TCustomMaskEdit.

Protected Methods

	Name	Description
	CheckCursor (see page 4595)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
V	EditCanModify (see page 4596)	Permits the user to edit control contents.
	GetFirstEditChar (see page 4596)	Returns the index in the EditMask (see page 4604) of the first nonliteral character.
	GetLastEditChar (see page 4597)	Returns the index in the EditMask (see page 4604) of the last nonliteral character.
	GetMaxChars (see page 4597)	Returns the length of the EditText (see page 4606).
	GetNextEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetPriorEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 4599)	Returns the endpoints of the current selection in the EditText (see page 4606).
V	KeyDown (see page 4599)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
V	KeyPress (see page 4600)	Supplements the inherited method to process the Esc and Enter keys.
V	KeyUp (see page 4600)	Key up event dispatcher.
	ReformatText (see page 4601)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
V	Reset (see page 4601)	Returns the EditText (see page 4606) to its value at the time when the edit control last lost focus.
	SetCursor (see page 4602)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetSel (see page 4602)	Selects character range.
V	Validate (see page 4603)	Validates a specified string against the current mask.

	ValidateError (see page 4603)	Beeps and raises an EDBEditError (see page 4592) exception.

Protected Properties

	Name	Description
	EditMask (see page 4604)	Specifies the mask that represents what text is valid for the masked edit control.
	MaskState (see page 4607)	Indicates state information about how the mask is being applied.
	MaxLength (see page 4607)	Specifies the maximum number of characters that can appear in the edit control.

Public Methods

	Name	Description
	Clear (see page 4595)	Sets the Text (see page 4608) property to an empty string.
	Create (see page 4594)	Creates an instance of TCustomMaskEdit (see page 4592).
	GetTextLen (see page 4599)	Returns the length of the underlying text.
	ValidateEdit (see page 4603)	Validates the EditText (see page 4606) against the current mask.

Public Properties

	Name	Description
	EditText (see page 4606)	Specifies the text that appears in the window of the masked edit control.
R	IsMasked (see page 4606)	Indicates whether a mask exists for the edit control.
	Text (see page 4608)	Denotes the underlying text for the masked edit control before the mask has been applied.

43.1.2.2 TCustomMaskEdit.Create Constructor

Creates an instance of TCustomMaskEdit (see page 4592).

Pascal

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

C++

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

Description

Call Create to instantiate a masked edit control at runtime. AOwner is the component that is responsible for freeing the masked edit control (typically the form). It becomes the value of the Owner property.

See Also

Owner

43.1.2.3 TCustomMaskEdit Methods

The methods of the TCustomMaskEdit class are listed here.

Protected Methods

	Name	Description
	CheckCursor (see page 4595)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	EditCanModify (see page 4596)	Permits the user to edit control contents.
	GetFirstEditChar (see page 4596)	Returns the index in the EditMask (see page 4604) of the first nonliteral character.
	GetLastEditChar (see page 4597)	Returns the index in the EditMask (see page 4604) of the last nonliteral character.
	GetMaxChars (see page 4597)	Returns the length of the EditText (see page 4606).
	GetNextEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or greater than an offset.

	GetPriorEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 4599)	Returns the endpoints of the current selection in the EditText (see page 4606).
V	KeyDown (see page 4599)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
V	KeyPress (see page 4600)	Supplements the inherited method to process the Esc and Enter keys.
V	KeyUp (see page 4600)	Key up event dispatcher.
	ReformatText (see page 4601)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
V	Reset (see page 4601)	Returns the EditText (see page 4606) to its value at the time when the edit control last lost focus.
	SetCursor (see page 4602)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetSel (see page 4602)	Selects character range.
V	Validate (see page 4603)	Validates a specified string against the current mask.
V	ValidateError (see page 4603)	Beeps and raises an EDBEditError (see page 4592) exception.

Public Methods

	Name	Description
V	Clear (see page 4595)	Sets the Text (see page 4608) property to an empty string.
	GetTextLen (see page 4599)	Returns the length of the underlying text.
V	ValidateEdit (see page 4603)	Validates the EditText (see page 4606) against the current mask.

43.1.2.3.1 TCustomMaskEdit.CheckCursor Method

Moves the cursor to the closest editable character and selects it if nothing is currently selected.

Pascal

```
procedure CheckCursor;
```

C++

```
__fastcall CheckCursor();
```

Description

CheckCursor is called whenever the position of the cursor may have changed. It ensures that at least one character is always selected. Because editing a masked edit control consists of replacing the blank characters in the mask, there must always be a selection to replace. The selection corresponds to one or more nonliteral characters in the mask, which are either blank or contain characters for the user to reenter.

See Also

GetSel (see page 4599)

SetCursor (see page 4602)

SetSel (see page 4602)

43.1.2.3.2 TCustomMaskEdit.Clear Method

Sets the Text (see page 4608) property to an empty string.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to remove all text that was entered into the masked edit control. Clear sets the value of Text (see page 4608) to an empty string. It does not clear the EditMask (see page 4604). If the edit control has a mask, EditText (see page 4606) is not empty, but has all non-literal characters replaced by blanks.

See Also

EditMask (see page 4604)

EditText (see page 4606)

IsMasked (see page 4606)

Reset (see page 4601)

Text (see page 4608)

43.1.2.3.3 TCustomMaskEdit.EditCanModify Method

Permits the user to edit control contents.

Pascal

```
function EditCanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean EditCanModify();
```

Description

The EditCanModify method defined by TCustomMaskEdit (see page 4592) always returns true, allowing the Text (see page 4608) to be edited. EditCanModify exists primarily to provide an interface for a method that can conditionally disallow changes to the Text (see page 4608) by returning false.

See Also

Text (see page 4608)

43.1.2.3.4 TCustomMaskEdit.GetFirstEditChar Method

Returns the index in the EditMask (see page 4604) of the first nonliteral character.

Pascal

```
function GetFirstEditChar: Integer;
```

C++

```
__fastcall int GetFirstEditChar();
```

Description

Use GetFirstEditChar to find the first position of the EditMask (see page 4604) where the user can modify the EditText (see page 4606). 0 specifies the first character, 1 the second character, and so on. If there is no mask, GetFirstEditChar returns zero.

Use GetFirstEditChar as a starting point for programmatically processing all the nonliteral characters in the EditText (see page 4606). Use GetNextEditChar (see page 4598) to continue to loop through the editable characters in the EditText (see page 4606), and GetLastEditChar (see page 4597) to determine when all nonliteral characters have been processed.

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetLastEditChar (see page 4597)

GetNextEditChar (see page 4598)

GetPriorEditChar (see page 4598)

43.1.2.3.5 TCustomMaskEdit.GetLastEditChar Method

Returns the index in the EditMask (see page 4604) of the last nonliteral character.

Pascal

```
function GetLastEditChar: Integer;
```

C++

```
__fastcall int GetLastEditChar();
```

Description

Use GetLastEditChar to find the last position of the EditMask (see page 4604) where the user can modify the EditText (see page 4606). 0 specifies the first character, 1 the second character, and so on. If there is no mask, GetLastEditChar returns the length of the text.

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetFirstEditChar (see page 4596)

GetMaxChars (see page 4597)

GetNextEditChar (see page 4598)

GetPriorEditChar (see page 4598)

43.1.2.3.6 TCustomMaskEdit.GetMaxChars Method

Returns the length of the EditText (see page 4606).

Pascal

```
function GetMaxChars: Integer;
```

C++

```
__fastcall int GetMaxChars();
```

Description

If there is a mask, GetMaxChars returns the length of the EditText (see page 4606), including all blanks. This is the same as MaxLength (see page 4607), because editing in a masked edit control consists of replacing nonliteral characters. If there is no mask, GetMaxChars returns the length of the text in the window. In this case, the value returned by GetMaxChars may be less than MaxLength (see page 4607), which is the maximum number of characters that can be entered into the edit control.

See Also

EditText (see page 4606)

GetLastEditChar (see page 4597)

MaxLength (see page 4607)

Text (see page 4608)

43.1.2.3.7 TCustomMaskEdit.GetNextEditChar Method

Returns the first index of a nonliteral character equal to or greater than an offset.

Pascal

```
function GetNextEditChar(Offset: Integer): Integer;
```

C++

```
__fastcall int GetNextEditChar(int Offset);
```

Description

Call GetNextEditChar to find the first position of the EditMask (see page 4604) at or following Offset where the user can modify the EditText (see page 4606). 0 specifies the first character, 1 the second character, and so on. If there is no nonliteral character at or following Offset, GetNextEditChar returns MaxLength (see page 4607).

Use GetNextEditChar to loop through the editable characters in the EditText (see page 4606), starting with the value returned by GetFirstEditChar (see page 4596) and using GetLastEditChar (see page 4597) to determine when all nonliteral characters have been processed.

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetFirstEditChar (see page 4596)

GetLastEditChar (see page 4597)

GetPriorEditChar (see page 4598)

43.1.2.3.8 TCustomMaskEdit.GetPriorEditChar Method

Returns the first index of a nonliteral character equal to or less than an offset.

Pascal

```
function GetPriorEditChar(Offset: Integer): Integer;
```

C++

```
__fastcall int GetPriorEditChar(int Offset);
```

Description

Call GetPriorEditChar to find the first position of the EditMask (see page 4604) at or preceding Offset, where the user can modify the EditText (see page 4606). 0 specifies the first character, 1 the second character, and so on. If there is no nonliteral character at or preceding Offset, GetPriorEditChar returns the index of the first editable character following Offset.

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetFirstEditChar (see page 4596)

GetLastEditChar (see page 4597)

GetNextEditChar (see page 4598)

43.1.2.3.9 TCustomMaskEdit.GetSel Method

Returns the endpoints of the current selection in the EditText (see page 4606).

Pascal

```
procedure GetSel(var SelStart: Integer; var SelStop: Integer);
```

C++

```
__fastcall GetSel(int SelStart, int SelStop);
```

Description

Use GetSel to obtain the indexes of the first and last selected characters of the EditText (see page 4606), where 0 is the index of the first character. Not all the characters in the selection will necessarily be editable, if the mask contains literal characters. Do not use the resulting indexes on the Text (see page 4608) property if there is a mask, because the EditText (see page 4606) may contain blanks or nonliteral characters not found in the Text (see page 4608).

To programmatically change the selected text, use GetNextEditChar (see page 4598) to cycle through the editable characters of the selection from SelStart to SelStop.

See Also

EditText (see page 4606)

GetNextEditChar (see page 4598)

SelText

SetSel (see page 4602)

Text (see page 4608)

43.1.2.3.10 TCustomMaskEdit.GetTextLen Method

Returns the length of the underlying text.

Pascal

```
function GetTextLen: Integer; reintroduce;
```

C++

```
__fastcall int GetTextLen();
```

Description

Use GetTextLen to determine the length of the underlying text for the masked edit control. The underlying text has not been formatted for display by the mask, and may differ from the EditText (see page 4606) that appears in the window of the control.

See Also

EditText (see page 4606)

GetTextBuf

MaxLength (see page 4607)

Text (see page 4608)

43.1.2.3.11 TCustomMaskEdit.KeyDown Method

Supplements the inherited KeyDown to process navigation keys and the Delete key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After calling the inherited KeyDown, KeyDown ensures that moving the cursor within the masked edit control places the cursor on an editable character, and that deleting the current selection replaces nonliteral characters with blanks. Derived objects can override this method to further supplement the processing that takes place when the user presses a key down.

See Also

CheckCursor (see page 4595)

KeyPress (see page 4600)

KeyUp (see page 4600)

SetSel (see page 4602)

43.1.2.3.12 TCustomMaskEdit.KeyPress Method

Supplements the inherited method to process the Esc and Enter keys.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

After calling the inherited method, KeyPress calls the Reset (see page 4601) method in response to the Esc key, or calls the ValidateEdit (see page 4603) method in response to the Enter key. Derived classes can override this method to further supplement the processing of specific keystrokes.

See Also

KeyDown (see page 4599)

Reset (see page 4601)

ValidateEdit (see page 4603)

43.1.2.3.13 TCustomMaskEdit.KeyUp Method

Key up event dispatcher.

Pascal

```
procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyUp(Word Key, TShiftState Shift);
```

Description

After calling the inherited method, KeyUp makes sure that selection using the Ctrl+Left and Ctrl+Right key combinations ends with an editable selection in the edit window. Derived classes can override this method to further supplement the processing that

takes place when the user releases a key.

See Also

CheckCursor (see page 4595)

KeyDown (see page 4599)

KeyPress (see page 4600)

SetSel (see page 4602)

43.1.2.3.14 TCustomMaskEdit.ReformatText Method

Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.

Pascal

```
procedure ReformatText(const NewMask: string);
```

C++

```
__fastcall ReformatText(const UnicodeString NewMask);
```

Description

ReformatText does the bulk of the work of setting the EditMask (see page 4604) property at runtime. ReformatText should not be used in place of the EditMask (see page 4604) property, however, as it does not deal with the MaskState (see page 4607) property or the positioning of the cursor.

See Also

CheckCursor (see page 4595)

EditMask (see page 4604)

EditText (see page 4606)

MaskState (see page 4607)

Text (see page 4608)

43.1.2.3.15 TCustomMaskEdit.Reset Method

Returns the EditText (see page 4606) to its value at the time when the edit control last lost focus.

Pascal

```
procedure Reset; virtual;
```

C++

```
virtual __fastcall Reset();
```

Description

Reset cancels the most recent edits to the masked edit control. The EditText (see page 4606) returns to the value at the time the control last lost focus, not the time when it first regained focus. Thus, any changes to the EditText (see page 4606) that occurred programmatically while the edit control did not have focus are also discarded.

See Also

Clear (see page 4595)

EditText (see page 4606)

KeyPress (see page 4600)

43.1.2.3.16 TCustomMaskEdit.SetCursor Method

Moves the cursor to the first editable character at or before Pos, and selects that character.

Pascal

```
procedure SetCursor(Pos: Integer);
```

C++

```
__fastcall SetCursor(int Pos);
```

Description

SetCursor handles positioning of the cursor within the EditText (see page 4606) so that it is always on an editable character. It also ensures that at least one character is always selected. Because editing a masked edit control consists of replacing the blank characters in the mask, there must always be a selection to replace. The selection corresponds to one or more nonliteral characters in the mask, which are either blank or contain characters for the user to reenter.

See Also

CheckCursor (see page 4595)

GetPriorEditChar (see page 4598)

SetSel (see page 4602)

43.1.2.3.17 TCustomMaskEdit.SetSel Method

Selects character range.

Pascal

```
procedure SetSel(SelStart: Integer; SelStop: Integer);
```

C++

```
__fastcall SetSel(int SelStart, int SelStop);
```

Description

SetSel selects the indicated range of characters in the EditText (see page 4606), regardless of whether any of them are editable. If SelStart is the same as SelStop, SetSel positions the cursor at the SelStart character. When using SetSel, leave the cursor in an editable position if the intent is to create a selection for the user to replace.

SetSel can be used to select non-editable characters that can then be captured by using the SelText property or by copying to the Clipboard. Once the selection has been captured, return the cursor to an editable position by calling CheckCursor (see page 4595).

See Also

CheckCursor (see page 4595)

EditText (see page 4606)

GetSel (see page 4599)

SelText

SetCursor (see page 4602)

43.1.2.3.18 TCustomMaskEdit.Validate Method

Validates a specified string against the current mask.

Pascal

```
function Validate(const Value: string; var Pos: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Validate(const UnicodeString Value, int Pos);
```

Description

The ValidateEdit (see page 4603) method calls Validate to perform the actual validation of the current text against the edit mask. It returns true if the string specified by Value is consistent with the edit mask, false if it detects an invalid character. If ValidateEdit (see page 4603) returns false, it returns the position of the invalid character as the Pos parameter.

See Also

EditMask (see page 4604)

ValidateEdit (see page 4603)

43.1.2.3.19 TCustomMaskEdit.ValidateEdit Method

Validates the EditText (see page 4606) against the current mask.

Pascal

```
procedure ValidateEdit; virtual;
```

C++

```
virtual __fastcall ValidateEdit();
```

Description

ValidateEdit attempts to validate the EditText (see page 4606) after each keystroke, and when the user attempts to leave the masked edit field. If the EditText (see page 4606) does not match the specifications of the EditMask (see page 4604), ValidateEdit calls ValidateError (see page 4603) to raise an exception, and returns the cursor to the first position where the EditText (see page 4606) did not comply with the EditMask (see page 4604).

See Also

EditMask (see page 4604)

EditText (see page 4606)

KeyPress (see page 4600)

ValidateError (see page 4603)

43.1.2.3.20 TCustomMaskEdit.ValidateError Method

Beeps and raises an EDBEditError (see page 4592) exception.

Pascal

```
procedure ValidateError; virtual;
```

C++

```
virtual __fastcall ValidateError();
```

Description

ValidateEdit (see page 4603) calls ValidateError when the EditText (see page 4606) fails to comply with the EditMask (see page 4604).

See Also

EditMask (see page 4604)

EditText (see page 4606)

ValidateEdit (see page 4603)

43.1.2.4 TCustomMaskEdit Properties

The properties of the TCustomMaskEdit class are listed here.

Protected Properties

	Name	Description
	EditMask (see page 4604)	Specifies the mask that represents what text is valid for the masked edit control.
	MaskState (see page 4607)	Indicates state information about how the mask is being applied.
	MaxLength (see page 4607)	Specifies the maximum number of characters that can appear in the edit control.

Public Properties

	Name	Description
	EditText (see page 4606)	Specifies the text that appears in the window of the masked edit control.
R	IsMasked (see page 4606)	Indicates whether a mask exists for the edit control.
	Text (see page 4608)	Denotes the underlying text for the masked edit control before the mask has been applied.

43.1.2.4.1 TCustomMaskEdit.EditMask Property

Specifies the mask that represents what text is valid for the masked edit control.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter into the masked edit control to valid characters and formats. If the user attempts to enter an invalid character, the edit control does not accept the character. Validation is performed on a character-by-character basis by the ValidateEdit (see page 4603) method.

Setting EditMask to an empty string removes the mask.

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
```

```
  using bit-wise AND operations and displayed in the third mask edit box.
  An additional test is made to assure that the values in the IP address
  and in the subnet mask are in the interval from 0 to 255,
  as this validation test cannot be accomplished
  using the edit mask property alone.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { initialize the maskedit boxes to contain the empty string }
    MaskEdit1.Text := '';
    MaskEdit2.Text := '';
    MaskEdit3.Text := '';

    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');
```

```
    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
```

See Also

EditText (see page 4606)

IsMasked (see page 4606)

MaskState (see page 4607)

Text (see page 4608)

ValidateEdit (see page 4603)

ValidateError (see page 4603)

43.1.2.4.2 TCustomMaskEdit.EditText Property

Specifies the text that appears in the window of the masked edit control.

Pascal

```
property EditText: string;
```

C++

```
__property UnicodeString EditText;
```

Description

EditText is the value of the text for the edit control after it is formatted using the EditMask (see page 4604). Before the text is fully entered into a masked edit control, the EditText includes a blank character for each unentered character. As characters are entered, the blank characters in the EditText are replaced by the values that the user types.

If literal mask characters are included in the text, and no character is substituted for blanks, the values of EditText and Text (see page 4608) are the same.

Note: EditText only applies at runtime.

See Also

EditMask (see page 4604)

Text (see page 4608)

43.1.2.4.3 TCustomMaskEdit.IsMasked Property

Indicates whether a mask exists for the edit control.

Pascal

```
property IsMasked: Boolean;
```

C++

```
__property Boolean IsMasked;
```

Description

Read IsMasked to determine whether the EditMask (see page 4604) property has been set. If IsMasked is true, the control has a mask, and input is validated at runtime. If IsMasked is false, the control does not restrict editing at runtime.

See Also

EditMask (see page 4604)

43.1.2.4.4 TCustomMaskEdit.MaskState Property

Indicates state information about how the mask is being applied.

Pascal

```
property MaskState: TMaskedState;
```

C++

```
__property TMaskedState MaskState;
```

Description

MaskState is used internally to determine appropriate actions based on how the mask is currently being applied. MaskState is a set drawn from among the following values:

Value	Meaning
msMasked	A mask has been assigned. MaskState includes msMasked when IsMasked (see page 4606) is true.
msReEnter	An invalid character has been rejected and the next edit should be a valid replacement character.
msDBSetText	The text comes from a database field and should not be validated.

MaskState is exposed as a protected property so that derived classes can maintain and use this state information in the implementation of properties and methods.

See Also

IsMasked (see page 4606)

ValidateEdit (see page 4603)

43.1.2.4.5 TCustomMaskEdit.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

MaxLength is the length of the EditText (see page 4606). Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask (see page 4604). If there is an EditMask (see page 4604), MaxLength is implied by the mask itself, and cannot be changed. The EditText (see page 4606) contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

See Also

EditMask (see page 4604)

EditText (see page 4606)

IsMasked (see page 4606)

Text (see page 4608)

43.1.2.4.6 TCustomMaskEdit.Text Property

Denotes the underlying text for the masked edit control before the mask has been applied.

Pascal

```
property Text: TMaskedText;
```

C++

```
__property TMaskedText Text;
```

Description

Use Text to determine the underlying value of the text before it has been formatted by the mask. Text may differ from the EditText (see page 4606) visible in the masked edit control if the mask specifies that literal characters should be removed, if the mask includes spaces, or if the mask includes characters that have not yet been filled in by the user. Text will not contain the blank characters, spaces will not be replaced by _, and literal characters in the mask will be removed if the mask indicates they should be.

When setting Text, the value of the text is formatted using the mask, and the resulting EditText (see page 4606) is used to update the window.

If there is no mask, Text is the text that appears in the edit control.

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
```

```
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
```

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetTextLen (see page 4599)

IsMasked (see page 4606)

ReformatText (see page 4601)

43.1.3 TMaskEdit Class

TMaskEdit implements a generic masked edit control.

Class Hierarchy

Pascal

```
TMaskEdit = class(TCustomMaskEdit);
```

C++

```
class TMaskEdit : public TCustomMaskEdit;
```

File

Mask

Description

Use a TMaskEdit object to put a masked edit control on your form. Masked edit controls validate the text the user enters against a mask that encodes the valid forms the text can take. The mask can also format text that is displayed to the user.

TMaskEdit implements the generic behavior introduced in TCustomMaskEdit (see page 4592). TMaskEdit publishes many of the properties and methods inherited from TCustomMaskEdit (see page 4592), but does not introduce any new behavior.

See Also

TDBEdit

TEdit

TCustomMaskEdit (see page 4592)

Edit Controls

43.1.3.1 TMaskEdit Members

The following tables list the members exposed by TMaskEdit.

Protected Methods

	Name	Description
	CheckCursor (see page 4595)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
V	EditCanModify (see page 4596)	Permits the user to edit control contents.
	GetFirstEditChar (see page 4596)	Returns the index in the EditMask (see page 4604) of the first nonliteral character.
	GetLastEditChar (see page 4597)	Returns the index in the EditMask (see page 4604) of the last nonliteral character.
	GetMaxChars (see page 4597)	Returns the length of the EditText (see page 4606).
	GetNextEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetPriorEditChar (see page 4598)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 4599)	Returns the endpoints of the current selection in the EditText (see page 4606).
V	KeyDown (see page 4599)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
V	KeyPress (see page 4600)	Supplements the inherited method to process the Esc and Enter keys.
V	KeyUp (see page 4600)	Key up event dispatcher.

	ReformatText (see page 4601)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
V	Reset (see page 4601)	Returns the EditText (see page 4606) to its value at the time when the edit control last lost focus.
	SetCursor (see page 4602)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetSel (see page 4602)	Selects character range.
V	Validate (see page 4603)	Validates a specified string against the current mask.
V	ValidateError (see page 4603)	Beeps and raises an EDBEditError (see page 4592) exception.

Protected Properties

	Name	Description
	MaskState (see page 4607)	Indicates state information about how the mask is being applied.

Public Methods

	Name	Description
V	Clear (see page 4595)	Sets the Text (see page 4608) property to an empty string.
V	Create (see page 4594)	Creates an instance of TCustomMaskEdit (see page 4592).
	GetTextLen (see page 4599)	Returns the length of the underlying text.
V	ValidateEdit (see page 4603)	Validates the EditText (see page 4606) against the current mask.

Public Properties

	Name	Description
	EditText (see page 4606)	Specifies the text that appears in the window of the masked edit control.
R	IsMasked (see page 4606)	Indicates whether a mask exists for the edit control.

Published Events

	Name	Description
	OnChange (see page 4649)	Occurs when the text for the edit control may have changed.
	OnClick (see page 4649)	Occurs when the user clicks the control.
	OnDblClick (see page 4651)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4652)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4655)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4658)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4658)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4659)	Occurs when a control receives the input focus.
	OnExit (see page 4662)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4664)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4668)	Occurs when a key is pressed.
	OnKeyUp (see page 4669)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4671)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4672)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4674)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4674)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4675)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4677)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4679)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4680)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4614)	Determines how the control aligns within its container (parent control).
	Alignment (see page 4617)	Determines how the text is aligned within the text edit control.
	Anchors (see page 4618)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 4618)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 4619)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 4620)	Specifies which edges of the control are beveled.
	BevelInner (see page 4620)	Specifies the cut of the inner bevel.
	BevelKind (see page 4620)	Specifies the control's bevel style.
	BevelOuter (see page 4621)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4621)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4622)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4622)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 4622)	Determines the case of the text within the edit control.
	Color (see page 4623)	Specifies the background color of the control.
	Constraints (see page 4624)	Specifies the size constraints for the control.
	Ctl3D (see page 4624)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 4625)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4627)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4627)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4627)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 4629)	Specifies the mask that represents what text is valid for the masked edit control.
	Enabled (see page 4631)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4631)	Controls the attributes of text written on or in the control.
	ImeMode (see page 4632)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4633)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 4633)	Specifies the maximum number of characters that can appear in the edit control.
	ParentBiDiMode (see page 4634)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4634)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4635)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4635)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4635)	Determines where a control looks for its font information.
	ParentShowHint (see page 4636)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 4636)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 4637)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4642)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 4642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4644)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4644)	Determines if the user can tab to a control.
	Text (see page 4645)	Denotes the underlying text for the masked edit control before the mask has been applied.
	TextHint (see page 4647)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 4647)	Determines whether the component appears onscreen.

43.1.3.2 TMaskEdit Properties

The properties of the TMaskEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 4614)	Determines how the control aligns within its container (parent control).
	Alignment (see page 4617)	Determines how the text is aligned within the text edit control.
	Anchors (see page 4618)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 4618)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 4619)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 4620)	Specifies which edges of the control are beveled.
	BevelInner (see page 4620)	Specifies the cut of the inner bevel.
	BevelKind (see page 4620)	Specifies the control's bevel style.
	BevelOuter (see page 4621)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4621)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4622)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4622)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 4622)	Determines the case of the text within the edit control.
	Color (see page 4623)	Specifies the background color of the control.
	Constraints (see page 4624)	Specifies the size constraints for the control.
	Ctl3D (see page 4624)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 4625)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4627)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4627)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4627)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 4629)	Specifies the mask that represents what text is valid for the masked edit control.
	Enabled (see page 4631)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4631)	Controls the attributes of text written on or in the control.
	ImeMode (see page 4632)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4633)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 4633)	Specifies the maximum number of characters that can appear in the edit control.
	ParentBiDiMode (see page 4634)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4634)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4635)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4635)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4635)	Determines where a control looks for its font information.
	ParentShowHint (see page 4636)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 4636)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 4637)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4642)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 4642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4644)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4644)	Determines if the user can tab to a control.

	Text (see page 4645)	Denotes the underlying text for the masked edit control before the mask has been applied.
	TextHint (see page 4647)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 4647)	Determines whether the component appears onscreen.

43.1.3.2.1 TMaskEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
```

```
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
```

```
  run this example, add ExtCtrls and FileCtrl to the uses
  clause of the form's implementation section.
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
```

```
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

43.1.3.2.2 TMaskEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment;
```

C++

```
__property Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment

Text

Displaying and Editing Fields in an Edit Box

43.1.3.2.3 TMaskEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

43.1.3.2.4 TMaskEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect;
```

C++

```
__property AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll

SelLength

SelStart

SelText

SetSelTextBuf

43.1.3.2.5 TMaskEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle

ClientHeight

ControlStyle

Height

43.1.3.2.6 TMaskEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

43.1.3.2.7 TMaskEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

43.1.3.2.8 TMaskEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

43.1.3.2.9 TMaskEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

43.1.3.2.10 TMaskEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

43.1.3.2.11 TMaskEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

43.1.3.2.12 TMaskEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

43.1.3.2.13 TMaskEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase;
```

C++

```
__property CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

43.1.3.2.14 TMaskEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
```

```
end;
```

See Also

ParentColor

43.1.3.2.15 TMaskEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

43.1.3.2.16 TMaskEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

43.1.3.2.17 TMaskEdit.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

43.1.3.2.18 TMaskEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

43.1.3.2.19 TMaskEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

43.1.3.2.20 TMaskEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

43.1.3.2.21 TMaskEdit.EditMask Property

Specifies the mask that represents what text is valid for the masked edit control.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter into the masked edit control to valid characters and formats. If the user attempts to enter an invalid character, the edit control does not accept the character. Validation is performed on a character-by-character basis by the ValidateEdit method.

Setting EditMask to an empty string removes the mask.

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
```

```
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
```

See Also

EditText (see page 4606)

IsMasked (see page 4606)

MaskState (see page 4607)

Text (see page 4608)

ValidateEdit (see page 4603)

ValidateError (see page 4603)

43.1.3.2.22 TMaskEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

43.1.3.2.23 TMaskEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

43.1.3.2.24 TMaskEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

43.1.3.2.25 TMaskEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

43.1.3.2.26 TMaskEdit.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

MaxLength is the length of the EditText. Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask. If there is an EditMask, MaxLength is implied by the mask itself, and cannot be changed. The EditText contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

See Also

EditMask (see page 4604)

EditText (see page 4606)

IsMasked (see page 4606)

Text (see page 4608)

43.1.3.2.27 TMaskEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

43.1.3.2.28 TMaskEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

43.1.3.2.29 TMaskEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

43.1.3.2.30 TMaskEdit.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

43.1.3.2.31 TMaskEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

43.1.3.2.32 TMaskEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

43.1.3.2.33 TMaskEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar;
```

C++

```
__property PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

43.1.3.2.34 TMaskEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
```

```
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
```

```
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;
```

```
{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);
```

```
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
```

```
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

43.1.3.2.35 TMaskEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard

GetSelTextBuf

43.1.3.2.36 TMaskEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

43.1.3.2.37 TMaskEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

43.1.3.2.38 TMaskEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

43.1.3.2.39 TMaskEdit.Text Property

Denotes the underlying text for the masked edit control before the mask has been applied.

Pascal

```
property Text: TMaskedText;
```

C++

```
__property TMaskedText Text;
```

Description

Use Text to determine the underlying value of the text before it has been formatted by the mask. Text may differ from the EditText visible in the masked edit control if the mask specifies that literal characters should be removed, if the mask includes spaces, or if the mask includes characters that have not yet been filled in by the user. Text will not contain the blank characters, spaces will not be replaced by _, and literal characters in the mask will be removed if the mask indicates they should be.

When setting Text, the value of the text is formatted using the mask, and the resulting EditText is used to update the window.

If there is no mask, Text is the text that appears in the edit control.

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
```

```
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
```

See Also

EditMask (see page 4604)

EditText (see page 4606)

GetTextLen (see page 4599)

IsMasked (see page 4606)

ReformatText (see page 4601)

43.1.3.2.40 TMaskEdit.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint;
```

C++

```
__property TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

43.1.3.2.41 TMaskEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
```

```
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

43.1.3.3 TMaskEdit Events

The events of the TMaskEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 4649)	Occurs when the text for the edit control may have changed.
	OnClick (see page 4649)	Occurs when the user clicks the control.
	OnDblClick (see page 4651)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4652)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4655)	Occurs when the user drags an object over a control.

	OnEndDock (see page 4658)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4658)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4659)	Occurs when a control receives the input focus.
	OnExit (see page 4662)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4664)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4668)	Occurs when a key is pressed.
	OnKeyUp (see page 4669)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4671)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4672)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4674)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4674)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4675)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4677)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4679)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4680)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

43.1.3.3.1 TMaskEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change

Text

43.1.3.3.2 TMaskEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an

OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
```

```
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

43.1.3.3.3 TMaskEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

43.1.3.3.4 TMaskEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

43.1.3.3.5 TMaskEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

43.1.3.3.6 TMaskEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

43.1.3.3.7 TMaskEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

43.1.3.3.8 TMaskEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

43.1.3.3.9 TMaskEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

43.1.3.3.10 TMaskEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
```

```
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
```

```
  Behavior is dependent on keyboard driver(USB, serial, etc).
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

43.1.3.3.11 TMaskEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

43.1.3.3.12 TMaskEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

43.1.3.3.13 TMaskEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

43.1.3.3.14 TMaskEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
```

```
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
```

```
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

43.1.3.3.15 TMaskEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

43.1.3.3.16 TMaskEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

43.1.3.3.17 TMaskEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
```

```
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

43.1.3.3.18 TMaskEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
```

```
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

43.1.3.3.19 TMaskEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

43.1.3.3.20 TMaskEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

43.2 Types

The following table lists types in this documentation.

Types

Name	Description
TMaskedState (see page 4681)	TMaskedState is a set of flags that describe the current state of a masked edit control.

43.2.1 Mask.TMaskedState Type

TMaskedState is a set of flags that describe the current state of a masked edit control.

Pascal

```
TMaskedState = set of (msMasked, msReEnter, msDBSetText);
```

C++

```
(msMasked, msReEnter, msDBSetText) TMaskedState;
```

File

Mask

Description

The following table indicates the meaning of the various TMaskedState flags:

Value	Meaning
msMasked	A mask has been assigned. MaskState includes msMasked when IsMasked is true.
msReEnter	An invalid character has been rejected and the next edit should be a valid replacement character.
msDBSetText	The text comes from a database field and should not be validated.

44 Masks Namespace

This is namespace Masks.

Topics

Name	Description
Classes (see page 4684)	The following table lists classes in this documentation.
Functions (see page 4688)	The following table lists functions in this documentation.

Classes

	Name	Description
	EMaskException (see page 4684)	EMaskException is the exception class for errors that occur when comparing a file name to a mask.
	TMask (see page 4684)	TMask allows the comparison of strings containing wildcards to a file mask.

Functions

	Name	Description
	MatchesMask (see page 4688)	Indicates whether a file name conforms to the format specified by a filter string.

44.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMaskException (see page 4684)	EMaskException is the exception class for errors that occur when comparing a file name to a mask.
	TMask (see page 4684)	TMask allows the comparison of strings containing wildcards to a file mask.

44.1.1 EMaskException Class

EMaskException is the exception class for errors that occur when comparing a file name to a mask.

Class Hierarchy

Pascal

```
EMaskException = class(Exception);
```

C++

```
class EMaskException : public Exception;
```

File

Masks

Description

EMaskException is raised when the TMask (see page 4684) object or the global MatchesMask (see page 4688) function encounters an improperly formed mask string.

See Also

TMask (see page 4684)

MatchesMask (see page 4688)

44.1.2 TMask Class

TMask allows the comparison of strings containing wildcards to a file mask.

Class Hierarchy

Pascal

```
TMask = class;
```

C++

```
class TMask;
```

File

Masks

Description

Use TMask to determine if a string matches the format described by a mask. The mask contains literal characters, wildcards that can match any character, and sets that match a single character that is included in the set.

Note: Do not confuse TMask with the EditMask of a field or masked edit object. While both are used for comparing strings to a symbolic description of valid values, the special mask symbols and matching rules are completely different.

See Also

TCustomMaskEdit (see page 4592)

TField

44.1.2.1 TMask Members

The following tables list the members exposed by TMask.

Public Methods

	Name	Description
	Create (see page 4685)	Creates an instance of TMask (see page 4684) for a given mask.
	Destroy (see page 4686)	Destroys an instance of TMask (see page 4684).
	Matches (see page 4686)	Indicates whether a specified string matches the mask.

44.1.2.2 TMask.Create Constructor

Creates an instance of TMask (see page 4684) for a given mask.

Pascal

```
constructor Create(const MaskValue: string);
```

C++

```
__fastcall TMask(const UnicodeString MaskValue);
```

Description

Call Create to create a mask object for comparing strings to the mask given by the MaskValue parameter. A valid mask consists of literal characters, sets, and wildcards.

Each literal character must match a single character in the string. The comparison to literal characters is case-insensitive.

Each set begins with an opening bracket ([) and ends with a closing bracket (]). Between the brackets are the elements of the set. Each element is a literal character or a range. Ranges are specified by an initial value, a dash (-), and a final value. Do not use spaces or commas to separate the elements of the set. A set must match a single character in the string. The character matches the set if it is the same as one of the literal characters in the set, or if it is in one of the ranges in the set. A character is in a range if it matches the initial value, the final value, or falls between the two values. All comparisons are case-insensitive. If the first character after the opening bracket of a set is an exclamation point (!), then the set matches any character that is not in the set.

Wildcards are asterisks (*) or question marks (?). An asterisk matches any number of characters. A question mark matches a

single arbitrary character.

See Also

Matches (see page 4686)

44.1.2.3 TMask.Destroy Destructor

Destroys an instance of TMask (see page 4684).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMask();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TMask (see page 4684) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store a parsed image of the mask.

See Also

System::TObject::Free

44.1.2.4 TMask Methods

The methods of the TMask class are listed here.

Public Methods

	Name	Description
	Matches (see page 4686)	Indicates whether a specified string matches the mask.

44.1.2.4.1 TMask.Matches Method

Indicates whether a specified string matches the mask.

Pascal

```
function Matches(const Filename: string): Boolean;
```

C++

```
__fastcall Boolean Matches(const UnicodeString Filename);
```

Description

Call Matches to compare the Filename parameter to the mask that describes valid values. Matches returns true if the string matches the mask. Matches returns false if the string does not match the mask. Matches raises an exception if the mask is syntactically invalid.

The mask is set when the TMask (see page 4684) object is created.

Note: Filename parameter does not need to be a file name. TMask (see page 4684) can be used to check strings against any syntactically correct mask.

See Also

Create (see page 4685)

44.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	MatchesMask (see page 4688)	Indicates whether a file name conforms to the format specified by a filter string.

44.2.1 Masks.MatchesMask Function

Indicates whether a file name conforms to the format specified by a filter string.

Pascal

```
function MatchesMask(const Filename: string; const Mask: string): Boolean;
```

C++

```
Boolean MatchesMask(const UnicodeString Filename, const UnicodeString Mask);
```

File

Masks

Description

Call MatchesMask to check the Filename parameter using the Mask (see page 4591) parameter to describe valid values. A valid mask consists of literal characters, sets, and wildcards.

Each literal character must match a single character in the string. The comparison to literal characters is case-insensitive.

Each set begins with an opening bracket ([) and ends with a closing bracket (]). Between the brackets are the elements of the set. Each element is a literal character or a range. Ranges are specified by an initial value, a dash (-), and a final value. Do not use spaces or commas to separate the elements of the set. A set must match a single character in the string. The character matches the set if it is the same as one of the literal characters in the set, or if it is in one of the ranges in the set. A character is in a range if it matches the initial value, the final value, or falls between the two values. All comparisons are case-insensitive. If the first character after the opening bracket of a set is an exclamation point (!), then the set matches any character that is not in the set.

Wildcards are asterisks (*) or question marks (?). An asterisk matches any number of characters. A question mark matches a single arbitrary character.

MatchesMask returns true if the string matches the mask. MatchesMask returns false if the string does not match the mask. MatchesMask raises an exception if the mask is syntactically invalid.

Note: The Filename parameter does not need to be a file name. MatchesMask can be used to check strings against any syntactically correct mask.

See Also

TMask (see page 4684)

45 MaskUtils Namespace

This is namespace MaskUtils.

Topics

Name	Description
Functions (see page 4691)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4698)	The following table lists structs, records, enums in this documentation.
Types (see page 4699)	The following table lists types in this documentation.
Variables (see page 4702)	The following table lists variables in this documentation.
Constants (see page 4704)	The following table lists constants in this documentation.

Constants

Name	Description
mDirLiteral (see page 4704)	mDirLiteral is used internally.
mDirLowerCase (see page 4704)	mDirLowerCase is used internally.
mDirReverse (see page 4705)	mDirReverse is used internally.
mDirUpperCase (see page 4705)	mDirUpperCase is used internally.
mMskAlpha (see page 4706)	mMskAlpha is used internally.
mMskAlphaNum (see page 4706)	mMskAlphaNum is used internally.
mMskAlphaNumOpt (see page 4706)	mMskAlphaNumOpt is used internally.
mMskAlphaOpt (see page 4707)	mMskAlphaOpt is used internally.
mMskAscii (see page 4707)	mMskAscii is used internally.
mMskAsciiOpt (see page 4708)	mMskAsciiOpt is used internally.
mMskDateSeparator (see page 4708)	mMskDateSeparator is used internally.
mMskNumeric (see page 4708)	mMskNumeric is used internally.
mMskNumericOpt (see page 4709)	mMskNumericOpt is used internally.
mMskNumSymOpt (see page 4709)	mMskNumSymOpt is used internally.
mMskTimeSeparator (see page 4710)	mMskTimeSeparator is used internally.

Functions

	Name	Description
	FormatMaskText (see page 4691)	Returns a string formatted using an edit mask.
	IsLiteralChar (see page 4693)	IsLiteralChar is used internally.
	MaskDoFormatText (see page 4693)	MaskDoFormatText is used internally.
	MaskGetCharType (see page 4693)	MaskGetCharType is used internally.
	MaskGetCurrentDirectives (see page 4694)	MaskGetCurrentDirectives is used internally.

	MaskGetFldSeparator (see page 4694)	MaskGetFldSeparator is used internally.
	MaskGetMaskBlank (see page 4695)	MaskGetMaskBlank is used internally.
	MaskGetMaskSave (see page 4695)	MaskGetMaskSave is used internally.
	MaskIntlLiteralToChar (see page 4695)	MaskIntlLiteralToChar is used internally.
	MaskOffsetToOffset (see page 4696)	MaskOffsetToOffset is used internally.
	MaskOffsetToWideOffset (see page 4696)	MaskOffsetToWideOffset is used internally.
	OffsetToMaskOffset (see page 4697)	OffsetToMaskOffset is used internally.
	PadInputLiterals (see page 4697)	PadInputLiterals is used internally.

Structs, Records, Enums

	Name	Description
	TMaskCharType (see page 4698)	TMaskCharType is used internally.

Types

Name	Description
TEditMask (see page 4699)	Represents the mask that validates and formats user input.
TMaskDirectives (see page 4700)	TMaskDirectives is used internally.
TMaskedText (see page 4701)	TMaskedText is the type for the text that is typed into a masked edit control.

Variables

Name	Description
DefaultBlank (see page 4702)	DefaultBlank is used internally.
MaskFieldSeparator (see page 4702)	MaskFieldSeparator is used internally.
MaskNoSave (see page 4703)	MaskNoSave is used internally.

45.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FormatMaskText (see page 4691)	Returns a string formatted using an edit mask.
	IsLiteralChar (see page 4693)	IsLiteralChar is used internally.
	MaskDoFormatText (see page 4693)	MaskDoFormatText is used internally.
	MaskGetCharType (see page 4693)	MaskGetCharType is used internally.
	MaskGetCurrentDirectives (see page 4694)	MaskGetCurrentDirectives is used internally.
	MaskGetFldSeparator (see page 4694)	MaskGetFldSeparator is used internally.
	MaskGetMaskBlank (see page 4695)	MaskGetMaskBlank is used internally.
	MaskGetMaskSave (see page 4695)	MaskGetMaskSave is used internally.
	MaskIntlLiteralToChar (see page 4695)	MaskIntlLiteralToChar is used internally.
	MaskOffsetToOffset (see page 4696)	MaskOffsetToOffset is used internally.
	MaskOffsetToWideOffset (see page 4696)	MaskOffsetToWideOffset is used internally.
	OffsetToMaskOffset (see page 4697)	OffsetToMaskOffset is used internally.
	PadInputLiterals (see page 4697)	PadInputLiterals is used internally.

45.1.1 MaskUtils.FormatMaskText Function

Returns a string formatted using an edit mask.

Pascal

```
function FormatMaskText(const EditMask: string; const Value: string): string;
```

C++

```
UnicodeString FormatMaskText(const UnicodeString EditMask, const UnicodeString Value);
```

File

MaskUtils

Description

Call FormatMaskText to apply the mask specified by the EditMask parameter to the text string specified by the Value parameter. The edit mask string consists of three fields with semicolons separating them. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of the mask are matched to characters in the Value parameter or are inserted into the Value string. The third part of the mask is the character used to represent missing characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the returned string as leading blanks. If a ! character is not present, optional characters are represented in the returned string as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.

<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case present in the Value parameter.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.
C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel, that character is substituted in the returned string.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel, that character is substituted in the returned string.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the returned string.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters are inserted automatically if the second field of the mask is 0, or matched to characters in the Value parameter if the second field is any other value. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask are included in the Value parameter. For example, the mask for a telephone number with area code could be the following string:

(000)_000-0000;0;*

The 0 in the second field indicates that the Value parameter should consist of the 10 digits of the phone number, rather than the 14 characters that make up the final formatted string.

A 0 in the second field indicates that literals are inserted into the Value string, any other character indicates that they should be included. The character to indicate whether literals should be included can be changed by changing the MaskNoSave (see page 4703) constant that is declared in the MaskUtils (see page 4689) unit.

The third field of the mask is the character that appears in the returned string for blanks (characters that do not appear in Value). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in the returned string.

Note: When working with multibyte character sets, each special mask character represents a single byte. To specify double-byte characters using the L, l, A, a, C, or c specifiers, the mask characters must be doubled as well. For example, LL would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

See Also

TCustomMaskEdit

TField

45.1.2 MaskUtils.IsLiteralChar Function

IsLiteralChar is used internally.

Pascal

```
function IsLiteralChar(const EditMask: string; Offset: Integer): Boolean;
```

C++

```
Boolean IsLiteralChar(const UnicodeString EditMask, int Offset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call IsLiteralChar directly.

See Also

FormatMaskText (see page 4691)

45.1.3 MaskUtils.MaskDoFormatText Function

MaskDoFormatText is used internally.

Pascal

```
function MaskDoFormatText(const EditMask: string; const Value: string; Blank: Char): string;
```

C++

```
UnicodeString MaskDoFormatText(const UnicodeString EditMask, const UnicodeString Value, Char
Blank);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskDoFormatText directly.

See Also

FormatMaskText (see page 4691)

45.1.4 MaskUtils.MaskGetCharType Function

MaskGetCharType is used internally.

Pascal

```
function MaskGetCharType(const EditMask: string; MaskOffset: Integer): TMaskCharType;
```

C++

```
TMaskCharType MaskGetCharType(const UnicodeString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskGetCharType directly.

See Also

FormatMaskText (see page 4691)

45.1.5 MaskUtils.MaskGetCurrentDirectives Function

MaskGetCurrentDirectives is used internally.

Pascal

```
function MaskGetCurrentDirectives(const EditMask: string; MaskOffset: Integer):
TMaskDirectives;
```

C++

```
TMaskDirectives MaskGetCurrentDirectives(const UnicodeString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskGetCurrentDirectives directly.

See Also

FormatMaskText (see page 4691)

45.1.6 MaskUtils.MaskGetFldSeparator Function

MaskGetFldSeparator is used internally.

Pascal

```
function MaskGetFldSeparator(const EditMask: string): Integer;
```

C++

```
int MaskGetFldSeparator(const UnicodeString EditMask);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskGetFldSeparator directly.

See Also

FormatMaskText (see page 4691)

45.1.7 MaskUtils.MaskGetMaskBlank Function

MaskGetMaskBlank is used internally.

Pascal

```
function MaskGetMaskBlank(const EditMask: string): Char;
```

C++

```
Char MaskGetMaskBlank(const UnicodeString EditMask);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskGetMaskBlank directly.

See Also

FormatMaskText (see page 4691)

45.1.8 MaskUtils.MaskGetMaskSave Function

MaskGetMaskSave is used internally.

Pascal

```
function MaskGetMaskSave(const EditMask: string): Boolean;
```

C++

```
Boolean MaskGetMaskSave(const UnicodeString EditMask);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskGetMaskSave directly.

See Also

FormatMaskText (see page 4691)

45.1.9 MaskUtils.MaskIntlLiteralToChar Function

MaskIntlLiteralToChar is used internally.

Pascal

```
function MaskIntlLiteralToChar(IChar: Char): Char;
```

C++

```
Char MaskIntlLiteralToChar(Char IChar);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskIntlLiteralToChar directly.

See Also

FormatMaskText (see page 4691)

45.1.10 MaskUtils.MaskOffsetToOffset Function

MaskOffsetToOffset is used internally.

Pascal

```
function MaskOffsetToOffset(const EditMask: String; MaskOffset: Integer): Integer;
```

C++

```
int MaskOffsetToOffset(const UnicodeString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskOffsetToOffset directly.

See Also

FormatMaskText (see page 4691)

45.1.11 MaskUtils.MaskOffsetToWideOffset Function

MaskOffsetToWideOffset is used internally.

Pascal

```
function MaskOffsetToWideOffset(const EditMask: String; MaskOffset: Integer): Integer;
```

C++

```
int MaskOffsetToWideOffset(const UnicodeString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call MaskOffsetToWideOffset directly.

See Also

FormatMaskText (see page 4691)

45.1.12 MaskUtils.OffsetToMaskOffset Function

OffsetToMaskOffset is used internally.

Pascal

```
function OffsetToMaskOffset(const EditMask: string; Offset: Integer): Integer;
```

C++

```
int OffsetToMaskOffset(const UnicodeString EditMask, int Offset);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call OffsetToMaskOffset directly.

See Also

FormatMaskText (see page 4691)

45.1.13 MaskUtils.PadInputLiterals Function

PadInputLiterals is used internally.

Pascal

```
function PadInputLiterals(const EditMask: String; const Value: string; Blank: Char): string;
```

C++

```
UnicodeString PadInputLiterals(const UnicodeString EditMask, const UnicodeString Value, Char
Blank);
```

File

MaskUtils

Description

This function is used internally by FormatMaskText (see page 4691). Do not call PadInputLiterals directly.

See Also

FormatMaskText (see page 4691)

45.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMaskCharType (see page 4698)	TMaskCharType is used internally.

45.2.1 MaskUtils.TMaskCharType Enumeration

TMaskCharType is used internally.

Pascal

```
TMaskCharType = (
  mcNone,
  mcLiteral,
  mcIntlLiteral,
  mcDirective,
  mcMask,
  mcMaskOpt,
  mcFieldSeparator,
  mcField
);
```

C++

```
enum TMaskCharType {
  mcNone,
  mcLiteral,
  mcIntlLiteral,
  mcDirective,
  mcMask,
  mcMaskOpt,
  mcFieldSeparator,
  mcField
};
```

File

MaskUtils

Description

This record is used internally by FormatMaskText (see page 4691). Do not use TMaskCharType directly.

See Also

FormatMaskText (see page 4691)

45.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEditMask (see page 4699)	Represents the mask that validates and formats user input.
TMaskDirectives (see page 4700)	TMaskDirectives is used internally.
TMaskedText (see page 4701)	TMaskedText is the type for the text that is typed into a masked edit control.

45.3.1 MaskUtils.TEditMask Type

Represents the mask that validates and formats user input.

Pascal

```
TEditMask = type string;
```

C++

```
type string TEditMask;
```

File

MaskUtils

Description

TEditMask is a string that consists of three fields with semicolons separating them. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the text as leading blanks. If a ! character is not present, optional characters are represented in the text as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.
<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal in the data.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.

C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the text. When the user enters characters in the field, the cursor skips the _ character.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters must be matched exactly in the edit control. They are inserted automatically, and the cursor skips over them during editing. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask should be included as part of the text for the edit control. For example, the mask for a telephone number with area code could be the following string:

(000)_000-0000;0;*

The 0 in the second field indicates that the Text property for the edit control would consist of the 10 digits that were entered, rather than the 14 characters that make up the telephone number as it appears in the edit control.

A 0 in the second field indicates that literals should not be included, any other character indicates that they should be included. The character that indicates whether literals should be included can be changed in the Edit Mask (see page 4591) property editor, or programmatically by changing the MaskNoSave (see page 4703) typed constant.

The third field of the mask is the character that appears in the edit control for blanks (characters that have not been entered). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in an edit window. However, when a user edits the text in a masked edit control, the cursor selects each blank character in turn, and skips over the space character.

Note: When working with multibyte character sets, each special mask character represents a single byte. To specify multi-byte characters using the L, l, A, a, C, or c specifiers, the mask characters must be duplicated as well. For example, LL would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

45.3.2 MaskUtils.TMaskDirectives Type

TMaskDirectives is used internally.

Pascal

```
TMaskDirectives = set of (mdReverseDir, mdUpperCase, mdLowerCase, mdLiteralChar);
```

C++

```
(mdReverseDir, mdUpperCase, mdLowerCase, mdLiteralChar) TMaskDirectives;
```

File

MaskUtils

Description

This record is used internally by FormatMaskText (see page 4691). Do not use TMaskDirectives directly.

See Also

FormatMaskText (see page 4691)

45.3.3 MaskUtils.TMaskedText Type

TMaskedText is the type for the text that is typed into a masked edit control.

Pascal

```
TMaskedText = type string;
```

C++

```
type string TMaskedText;
```

File

MaskUtils

Description

TMaskedText is the value of a masked edit control's text before the mask is applied.

45.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefaultBlank (see page 4702)	DefaultBlank is used internally.
MaskFieldSeparator (see page 4702)	MaskFieldSeparator is used internally.
MaskNoSave (see page 4703)	MaskNoSave is used internally.

45.4.1 MaskUtils.DefaultBlank Variable

DefaultBlank is used internally.

Pascal

```
DefaultBlank: Char = '_';
```

C++

```
Char DefaultBlank = '_';
```

File

MaskUtils

Description

This variable is used internally by FormatMaskText (see page 4691). Do not use DefaultBlank directly.

See Also

FormatMaskText (see page 4691)

45.4.2 MaskUtils.MaskFieldSeparator Variable

MaskFieldSeparator is used internally.

Pascal

```
MaskFieldSeparator: Char = ';';
```

C++

```
Char MaskFieldSeparator = ';';
```

File

MaskUtils

Description

This variable is used internally by FormatMaskText (see page 4691). Do not use MaskFieldSeparator directly.

See Also

FormatMaskText (see page 4691)

45.4.3 MaskUtils.MaskNoSave Variable

MaskNoSave is used internally.

Pascal

```
MaskNoSave: Char = '0';
```

C++

```
Char MaskNoSave = '0';
```

File

MaskUtils

Description

This variable is used internally by FormatMaskText (see page 4691). Do not use MaskNoSave directly.

See Also

FormatMaskText (see page 4691)

45.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
mDirLiteral (see page 4704)	mDirLiteral is used internally.
mDirLowerCase (see page 4704)	mDirLowerCase is used internally.
mDirReverse (see page 4705)	mDirReverse is used internally.
mDirUpperCase (see page 4705)	mDirUpperCase is used internally.
mMskAlpha (see page 4706)	mMskAlpha is used internally.
mMskAlphaNum (see page 4706)	mMskAlphaNum is used internally.
mMskAlphaNumOpt (see page 4706)	mMskAlphaNumOpt is used internally.
mMskAlphaOpt (see page 4707)	mMskAlphaOpt is used internally.
mMskAscii (see page 4707)	mMskAscii is used internally.
mMskAsciiOpt (see page 4708)	mMskAsciiOpt is used internally.
mMskDateSeparator (see page 4708)	mMskDateSeparator is used internally.
mMskNumeric (see page 4708)	mMskNumeric is used internally.
mMskNumericOpt (see page 4709)	mMskNumericOpt is used internally.
mMskNumSymOpt (see page 4709)	mMskNumSymOpt is used internally.
mMskTimeSeparator (see page 4710)	mMskTimeSeparator is used internally.

45.5.1 MaskUtils.mDirLiteral Constant

mDirLiteral is used internally.

Pascal

```
mDirLiteral = '\';
```

C++

```
mDirLiteral = '\';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mDirLiteral directly.

See Also

FormatMaskText (see page 4691)

45.5.2 MaskUtils.mDirLowerCase Constant

mDirLowerCase is used internally.

Pascal

```
mDirLowerCase = '<';
```

C++

```
mDirLowerCase = '<';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mDirLowerCase directly.

See Also

FormatMaskText (see page 4691)

45.5.3 MaskUtils.mDirReverse Constant

mDirReverse is used internally.

Pascal

```
mDirReverse = '!';
```

C++

```
mDirReverse = '!';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mDirReverse directly.

See Also

FormatMaskText (see page 4691)

45.5.4 MaskUtils.mDirUpperCase Constant

mDirUpperCase is used internally.

Pascal

```
mDirUpperCase = '>';
```

C++

```
mDirUpperCase = '>';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mDirUpperCase directly.

See Also

FormatMaskText (see page 4691)

45.5.5 MaskUtils.mMskAlpha Constant

mMskAlpha is used internally.

Pascal

```
mMskAlpha = 'L';
```

C++

```
mMskAlpha = 'L';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAlpha directly.

See Also

FormatMaskText (see page 4691)

45.5.6 MaskUtils.mMskAlphaNum Constant

mMskAlphaNum is used internally.

Pascal

```
mMskAlphaNum = 'A';
```

C++

```
mMskAlphaNum = 'A';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAlphaNum directly.

See Also

FormatMaskText (see page 4691)

45.5.7 MaskUtils.mMskAlphaNumOpt Constant

mMskAlphaNumOpt is used internally.

Pascal

```
mMskAlphaNumOpt = 'a';
```

C++

```
mMskAlphaNumOpt = 'a';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAlphaNumOpt directly.

See Also

FormatMaskText (see page 4691)

45.5.8 MaskUtils.mMskAlphaOpt Constant

mMskAlphaOpt is used internally.

Pascal

```
mMskAlphaOpt = 'l';
```

C++

```
mMskAlphaOpt = 'l';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAlphaOpt directly.

See Also

FormatMaskText (see page 4691)

45.5.9 MaskUtils.mMskAscii Constant

mMskAscii is used internally.

Pascal

```
mMskAscii = 'C';
```

C++

```
mMskAscii = 'C';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAscii directly.

See Also

FormatMaskText (see page 4691)

45.5.10 MaskUtils.mMskAsciiOpt Constant

mMskAsciiOpt is used internally.

Pascal

```
mMskAsciiOpt = 'c';
```

C++

```
mMskAsciiOpt = 'c';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskAsciiOpt directly.

See Also

FormatMaskText (see page 4691)

45.5.11 MaskUtils.mMskDateSeparator Constant

mMskDateSeparator is used internally.

Pascal

```
mMskDateSeparator = '/';
```

C++

```
mMskDateSeparator = '/';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskDateSeparator directly.

See Also

FormatMaskText (see page 4691)

45.5.12 MaskUtils.mMskNumeric Constant

mMskNumeric is used internally.

Pascal

```
mMskNumeric = '0';
```

C++

```
mMskNumeric = '0';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskNumeric directly.

See Also

FormatMaskText (see page 4691)

45.5.13 MaskUtils.mMskNumericOpt Constant

mMskNumericOpt is used internally.

Pascal

```
mMskNumericOpt = '9';
```

C++

```
mMskNumericOpt = '9';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskNumericOpt directly.

See Also

FormatMaskText (see page 4691)

45.5.14 MaskUtils.mMskNumSymOpt Constant

mMskNumSymOpt is used internally.

Pascal

```
mMskNumSymOpt = '#';
```

C++

```
mMskNumSymOpt = '#';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskNumSymOpt directly.

See Also

FormatMaskText (see page 4691)

45.5.15 MaskUtils.mMskTimeSeparator Constant

mMskTimeSeparator is used internally.

Pascal

```
mMskTimeSeparator = ':';
```

C++

```
mMskTimeSeparator = ':';
```

File

MaskUtils

Description

This constant is used internally by FormatMaskText (see page 4691). Do not use mMskTimeSeparator directly.

See Also

FormatMaskText (see page 4691)

46 Math Namespace

This is namespace Math.

Topics

Name	Description
Classes (see page 4715)	The following table lists classes in this documentation.
Functions (see page 4717)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4769)	The following table lists structs, records, enums in this documentation.
Types (see page 4773)	The following table lists types in this documentation.
Constants (see page 4776)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidArgument (see page 4715)	EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

Constants

Name	Description
Infinity (see page 4776)	Infinity represents positive infinity.
MaxComp (see page 4777)	MaxComp defines the maximum absolute value for a Comp variable.
MaxDouble (see page 4777)	MaxDouble defines the maximum absolute value for a Double variable.
MaxExtended (see page 4777)	MaxExtended defines the maximum absolute value for an Extended variable.
MaxSingle (see page 4778)	MaxSingle defines the maximum absolute value for a Single variable.
MinComp (see page 4778)	MinComp defines the smallest absolute value for a Comp variable.
MinDouble (see page 4779)	MinDouble defines the smallest absolute value for a Double variable.
MinExtended (see page 4779)	MinExtended defines the smallest absolute value for an Extended variable.
MinSingle (see page 4779)	MinSingle defines the smallest absolute value for a Single variable.
NaN (see page 4780)	NaN represents a value that cannot be represented as a valid number.
NegativeValue (see page 4780)	NegativeValue defines a negative number result from using Sign (see page 4761).
NegInfinity (see page 4781)	NegInfinity represents negative infinity.
PositiveValue (see page 4781)	PositiveValue defines a positive number result from using Sign (see page 4761).
ZeroValue (see page 4782)	ZeroValue defines a zero value number result from using Sign (see page 4761).

Functions

	Name	Description
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCosh (see page 4720)	Calculates the inverse hyperbolic cosine of a given number.
	ArcCot (see page 4720)	Calculates the inverse cotangent of a given number.
	ArcCotH (see page 4721)	Calculates the inverse hyperbolic cotangent of a given number.
	ArcCsc (see page 4721)	Calculates the inverse cosecant of a given number.
	ArcCscH (see page 4722)	Calculates the inverse hyperbolic cosecant of a given number.
	ArcSec (see page 4722)	Calculates the inverse secant of a given number.
	ArcSecH (see page 4722)	Calculates the inverse hyperbolic secant of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSinh (see page 4723)	Calculates the inverse hyperbolic sine of a given number.
	ArcTan2 (see page 4724)	Calculates the arctangent angle and quadrant of a given number.
	ArcTanh (see page 4724)	Calculates the inverse hyperbolic tangent of a given number.
	Ceil (see page 4725)	Rounds variables up toward positive infinity.
	ClearExceptions (see page 4725)	Clears exception bits in FPU status word.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	Cosecant (see page 4726)	Returns the cosecant of an angle.
	Cosh (see page 4727)	Calculates the hyperbolic cosine of an angle.
	Cot (see page 4727)	Calculates the cotangent of an angle.
	Cotan (see page 4728)	Calculates the cotangent of an angle.
	CotH (see page 4728)	Calculates the hyperbolic cotangent of an angle.
	Csc (see page 4729)	Returns the cosecant of an angle.
	CscH (see page 4729)	Returns the hyperbolic cosecant of an angle.
	CycleToDeg (see page 4730)	Converts an angle measurement from cycles to degrees.
	CycleToGrad (see page 4730)	Converts an angle measurement from cycles to grads.
	CycleToRad (see page 4730)	Converts an angle measurement from cycles to radians.
	DegToCycle (see page 4731)	Returns the value of a degree measurement expressed in cycles.
	DegToGrad (see page 4731)	Returns the value of a degree measurement expressed in grads.
	DegToRad (see page 4732)	Returns the value of a degree measurement expressed in radians.
	DivMod (see page 4732)	Returns the result of an integer division, including the remainder.
	DoubleDecliningBalance (see page 4732)	Calculates the depreciation of an asset using the double-declining balance method.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	Floor (see page 4733)	Rounds variables toward negative infinity.
	Frexp (see page 4734)	Separates the Mantissa and Exponent of X
	FutureValue (see page 4734)	Calculates the future value of an investment.
	GetExceptionMask (see page 4735)	Returns the exception mask from the FPU control word.
	GetPrecisionMode (see page 4735)	Returns the FPU precision control mode.
	GetRoundMode (see page 4736)	Returns the FPU rounding mode.
	GradToCycle (see page 4737)	Converts grad measurements to cycles.
	GradToDeg (see page 4737)	Converts grad measurements to degrees.
	GradToRad (see page 4737)	Converts grad measurements to radians.
	Hypot (see page 4738)	Calculates the length of the hypotenuse.

	IfThen (see page 4738)	Conditionally returns one of two specified values.
	IfThen (see page 4738)	Conditionally returns one of two specified values.
	IfThen (see page 4738)	Conditionally returns one of two specified values.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InterestPayment (see page 4739)	Calculates the interest portion of a loan payment.
	InterestRate (see page 4740)	Returns the interest rate required to increase PresentValue (see page 4753) to FutureValue (see page 4734).
	InternalRateOfReturn (see page 4740)	Calculates the internal rate of return for an investment.
	IntPower (see page 4741)	Calculates the integral power of a base value.
	IsInfinite (see page 4741)	Indicates when a variable or expression represents an infinite value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	Ldexp (see page 4743)	Calculates X times (2 to the power of P).
	LnXP1 (see page 4743)	Returns the natural log of (X+1)
	Log10 (see page 4743)	Calculates log base 10.
	Log2 (see page 4744)	Calculates log base 2.
	LogN (see page 4744)	Calculates the log of X for a specified base.
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	MaxIntValue (see page 4745)	Returns the largest signed value in an integer array.
	MaxValue (see page 4746)	Returns the largest signed value in an array.
	Mean (see page 4746)	Returns the average of all values in an array.
	MeanAndStdDev (see page 4747)	Calculates the mean and standard deviation of array elements.
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	MinIntValue (see page 4748)	Returns the smallest signed value in an integer array.
	MinValue (see page 4748)	Returns smallest signed value in an array.
	MomentSkewKurtosis (see page 4749)	Calculates the mean, variance, skew, and kurtosis.
	NetPresentValue (see page 4749)	Calculates the current value from an array of estimated cash flow values.
	Norm (see page 4750)	Returns the Euclidean 'L-2' norm.
	NumberOfPeriods (see page 4750)	Returns the number of payment periods for an investment.
	Payment (see page 4751)	Calculates a fully amortized payment.
	PeriodPayment (see page 4751)	Returns the principal amount from a full payment.
	Poly (see page 4751)	Evaluates a uniform polynomial of one variable at the value X.
	PopnStdDev (see page 4752)	Calculates the population standard deviation.
	PopnVariance (see page 4752)	Calculates the population variance.
	Power (see page 4753)	Raises Base to any power.
	Power (see page 4753)	Raises Base to any power.
	Power (see page 4753)	Raises Base to any power.
	PresentValue (see page 4753)	Calculates the present value of an investment.
	RadToCycle (see page 4754)	Converts radians to cycles.

	RadToDeg (see page 4754)	Converts radians to degrees.
	RadToGrad (see page 4754)	Converts radians to grads.
	RandG (see page 4755)	Generates random numbers with Gaussian distribution.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomRange (see page 4756)	Returns a random integer from a specified range.
	RoundTo (see page 4756)	Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	Sec (see page 4758)	Calculates the secant of an angle.
	Secant (see page 4758)	Calculates the secant of an angle.
	SecH (see page 4759)	Calculates the hyperbolic secant of an angle.
	SetExceptionMask (see page 4759)	Sets the exception mask on the FPU control word.
	SetPrecisionMode (see page 4760)	Sets the FPU precision control mode.
	SetRoundMode (see page 4760)	Sets the FPU rounding mode.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	SimpleRoundTo (see page 4761)	Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.
	SinCos (see page 4762)	Returns sine and cosine of an angle.
	Sinh (see page 4762)	Returns the hyperbolic sine of an angle.
	SLNDepreciation (see page 4763)	Returns the straight-line depreciation allowance of an asset.
	StdDev (see page 4763)	Returns the sample standard deviation for elements in an array.
	Sum (see page 4764)	Returns the sum of the elements in an array.
	SumInt (see page 4764)	Returns the sum of the elements in an integer array.
	SumOfSquares (see page 4765)	Returns the sum of the squared values from a data array.
	SumsAndSquares (see page 4765)	Returns the sum of the values and the sum of the squared values in an array.
	SYDDepreciation (see page 4766)	Calculates depreciation for an asset.
	Tan (see page 4766)	Returns the tangent of X.
	Tanh (see page 4767)	Returns the hyperbolic tangent of X.
	TotalVariance (see page 4767)	Returns the statistical variance from an array of values.
	Variance (see page 4768)	Calculates statistical sample variance from an array of data.

Structs, Records, Enums

	Name	Description
	TFPUException (see page 4769)	TFPUException defines TFPUExceptionMask (see page 4773) values.
	TFPUPrecisionMode (see page 4770)	TFPUPrecisionMode defines how the FPU handles precision.
	TFPURoundingMode (see page 4771)	TFPURoundingMode defines how the FPU handles rounding issues.
	TPaymentTime (see page 4772)	TPaymentTime is an enumerated type used as a parameter in financial functions.

Types

Name	Description
TFPUExceptionMask (see page 4773)	TFPUExceptionMask defines a set of TFPUException (see page 4769) values.
TRoundToEXRangeExtended (see page 4774)	This is type Math.TRoundToEXRangeExtended.
TRoundToRange (see page 4774)	TRoundToRange defines the RoundTo (see page 4756) rounding limits.
TValueSign (see page 4774)	TValueSign defines the Sign (see page 4761) returned value.

46.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidArgument (see page 4715)	EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

46.1.1 EInvalidArgument Class

EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

Class Hierarchy

Pascal

```
EInvalidArgument = class(EMathError);
```

C++

```
class EInvalidArgument : public EMathError;
```

File

Math

Description

EInvalidArgument is raised by some functions in the Math (see page 4711) unit (mostly financial functions) when one of the parameters is out of range or makes the calculation impossible.

Note: On Windows, because EInvalidArgument descends (indirectly) from EExternal, it saves the Windows exception record (if there is one) in its ExceptionRecord data structure.

See Also

EExternal

DoubleDecliningBalance (see page 4732)

FutureValue (see page 4734)

InterestPayment (see page 4739)

NetPresentValue (see page 4749)

PeriodPayment (see page 4751)

Power (see page 4753)

SLNDepreciation (see page 4763)

SYDDepreciation (see page 4766)

EInvalidOp

EOverflow

EUnderflow

EZeroDivide

46.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCos (see page 4719)	Calculates the inverse cosine of a given number.
	ArcCosh (see page 4720)	Calculates the inverse hyperbolic cosine of a given number.
	ArcCot (see page 4720)	Calculates the inverse cotangent of a given number.
	ArcCotH (see page 4721)	Calculates the inverse hyperbolic cotangent of a given number.
	ArcCsc (see page 4721)	Calculates the inverse cosecant of a given number.
	ArcCscH (see page 4722)	Calculates the inverse hyperbolic cosecant of a given number.
	ArcSec (see page 4722)	Calculates the inverse secant of a given number.
	ArcSecH (see page 4722)	Calculates the inverse hyperbolic secant of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSin (see page 4723)	Calculates the inverse sine of a given number.
	ArcSinh (see page 4723)	Calculates the inverse hyperbolic sine of a given number.
	ArcTan2 (see page 4724)	Calculates the arctangent angle and quadrant of a given number.
	ArcTanh (see page 4724)	Calculates the inverse hyperbolic tangent of a given number.
	Ceil (see page 4725)	Rounds variables up toward positive infinity.
	ClearExceptions (see page 4725)	Clears exception bits in FPU status word.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	CompareValue (see page 4726)	Returns the relationship between two numeric values.
	Cosecant (see page 4726)	Returns the cosecant of an angle.
	Cosh (see page 4727)	Calculates the hyperbolic cosine of an angle.
	Cot (see page 4727)	Calculates the cotangent of an angle.
	Cotan (see page 4728)	Calculates the cotangent of an angle.
	CotH (see page 4728)	Calculates the hyperbolic cotangent of an angle.
	Csc (see page 4729)	Returns the cosecant of an angle.
	CscH (see page 4729)	Returns the hyperbolic cosecant of an angle.
	CycleToDeg (see page 4730)	Converts an angle measurement from cycles to degrees.
	CycleToGrad (see page 4730)	Converts an angle measurement from cycles to grads.
	CycleToRad (see page 4730)	Converts an angle measurement from cycles to radians.
	DegToCycle (see page 4731)	Returns the value of a degree measurement expressed in cycles.
	DegToGrad (see page 4731)	Returns the value of a degree measurement expressed in grads.
	DegToRad (see page 4732)	Returns the value of a degree measurement expressed in radians.
	DivMod (see page 4732)	Returns the result of an integer division, including the remainder.
	DoubleDecliningBalance (see page 4732)	Calculates the depreciation of an asset using the double-declining balance method.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 4733)	Returns the closest value to a specified value within a specified range.
	Floor (see page 4733)	Rounds variables toward negative infinity.
	Frexp (see page 4734)	Separates the Mantissa and Exponent of X
	FutureValue (see page 4734)	Calculates the future value of an investment.

	GetExceptionMask (see page 4735)	Returns the exception mask from the FPU control word.
	GetPrecisionMode (see page 4735)	Returns the FPU precision control mode.
	GetRoundMode (see page 4736)	Returns the FPU rounding mode.
	GradToCycle (see page 4737)	Converts grad measurements to cycles.
	GradToDeg (see page 4737)	Converts grad measurements to degrees.
	GradToRad (see page 4737)	Converts grad measurements to radians.
	Hypot (see page 4738)	Calculates the length of the hypotenuse.
	IfThen (see page 4738)	Conditionally returns one of two specified values.
	IfThen (see page 4738)	Conditionally returns one of two specified values.
	IfThen (see page 4738)	Conditionally returns one of two specified values.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InRange (see page 4738)	Indicates whether a value falls within a specified range.
	InterestPayment (see page 4739)	Calculates the interest portion of a loan payment.
	InterestRate (see page 4740)	Returns the interest rate required to increase PresentValue (see page 4753) to FutureValue (see page 4734).
	InternalRateOfReturn (see page 4740)	Calculates the internal rate of return for an investment.
	IntPower (see page 4741)	Calculates the integral power of a base value.
	IsInfinite (see page 4741)	Indicates when a variable or expression represents an infinite value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 4742)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 4742)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	Ldexp (see page 4743)	Calculates X times (2 to the power of P).
	LnXP1 (see page 4743)	Returns the natural log of (X+1)
	Log10 (see page 4743)	Calculates log base 10.
	Log2 (see page 4744)	Calculates log base 2.
	LogN (see page 4744)	Calculates the log of X for a specified base.
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	Max (see page 4745)	Returns the greater of two numeric values (Delphi only).
	MaxIntValue (see page 4745)	Returns the largest signed value in an integer array.
	MaxValue (see page 4746)	Returns the largest signed value in an array.
	Mean (see page 4746)	Returns the average of all values in an array.
	MeanAndStdDev (see page 4747)	Calculates the mean and standard deviation of array elements.
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 4747)	Returns the lesser of two numeric values (Delphi only).
	MinIntValue (see page 4748)	Returns the smallest signed value in an integer array.
	MinValue (see page 4748)	Returns smallest signed value in an array.
	MomentSkewKurtosis (see page 4749)	Calculates the mean, variance, skew, and kurtosis.
	NetPresentValue (see page 4749)	Calculates the current value from an array of estimated cash flow values.
	Norm (see page 4750)	Returns the Euclidean 'L-2' norm.
	NumberOfPeriods (see page 4750)	Returns the number of payment periods for an investment.
	Payment (see page 4751)	Calculates a fully amortized payment.
	PeriodPayment (see page 4751)	Returns the principal amount from a full payment.
	Poly (see page 4751)	Evaluates a uniform polynomial of one variable at the value X.

	PopnStdDev (see page 4752)	Calculates the population standard deviation.
	PopnVariance (see page 4752)	Calculates the population variance.
	Power (see page 4753)	Raises Base to any power.
	Power (see page 4753)	Raises Base to any power.
	Power (see page 4753)	Raises Base to any power.
	PresentValue (see page 4753)	Calculates the present value of an investment.
	RadToCycle (see page 4754)	Converts radians to cycles.
	RadToDeg (see page 4754)	Converts radians to degrees.
	RadToGrad (see page 4754)	Converts radians to grads.
	RandG (see page 4755)	Generates random numbers with Gaussian distribution.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomFrom (see page 4755)	Returns a randomly selected element from an array.
	RandomRange (see page 4756)	Returns a random integer from a specified range.
	RoundTo (see page 4756)	Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 4757)	Indicates whether two floating-point values are (approximately) equal.
	Sec (see page 4758)	Calculates the secant of an angle.
	Secant (see page 4758)	Calculates the secant of an angle.
	SecH (see page 4759)	Calculates the hyperbolic secant of an angle.
	SetExceptionMask (see page 4759)	Sets the exception mask on the FPU control word.
	SetPrecisionMode (see page 4760)	Sets the FPU precision control mode.
	SetRoundMode (see page 4760)	Sets the FPU rounding mode.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 4761)	Indicates whether a numeric value is positive, negative, or zero.
	SimpleRoundTo (see page 4761)	Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.
	SinCos (see page 4762)	Returns sine and cosine of an angle.
	Sinh (see page 4762)	Returns the hyperbolic sine of an angle.
	SLNDepreciation (see page 4763)	Returns the straight-line depreciation allowance of an asset.
	StdDev (see page 4763)	Returns the sample standard deviation for elements in an array.
	Sum (see page 4764)	Returns the sum of the elements in an array.
	SumInt (see page 4764)	Returns the sum of the elements in an integer array.
	SumOfSquares (see page 4765)	Returns the sum of the squared values from a data array.
	SumsAndSquares (see page 4765)	Returns the sum of the values and the sum of the squared values in an array.
	SYDDepreciation (see page 4766)	Calculates depreciation for an asset.
	Tan (see page 4766)	Returns the tangent of X.
	Tanh (see page 4767)	Returns the hyperbolic tangent of X.
	TotalVariance (see page 4767)	Returns the statistical variance from an array of values.
	Variance (see page 4768)	Calculates statistical sample variance from an array of data.

46.2.1 Math.ArcCos Function

Calculates the inverse cosine of a given number.

Pascal

```
function ArcCos(const X: Extended): Extended; overload;
function ArcCos(const X: Double): Double; overload;
function ArcCos(const X: Single): Single; overload;
```

C++

```
Extended ArcCos(const Extended X);
Double ArcCos(const Double X);
Single ArcCos(const Single X);
```

File

Math

Description

ArcCos returns the inverse cosine of X.

X must be between -1 and 1.

The return value is in the range [0..Pi], in radians.

See Also

Cos

ArcCosh (see page 4720)

ArcSin (see page 4723)

46.2.2 Math.ArcCosh Function

Calculates the inverse hyperbolic cosine of a given number.

Pascal

```
function ArcCosh(const X: Extended): Extended;
```

C++

```
Extended ArcCosh(const Extended X);
```

File

Math

Description

ArcCosh returns the inverse hyperbolic cosine of X. The value of X must be greater than or equal to 1.

See Also

Cosh (see page 4727)

ArcCos (see page 4719)

46.2.3 Math.ArcCot Function

Calculates the inverse cotangent of a given number.

Pascal

```
function ArcCot(const X: Extended): Extended;
```

C++

```
Extended ArcCot(const Extended X);
```

File

Math

Description

ArcCot returns the inverse cotangent of X.

See Also

Cot (see page 4727)

ArcCoth

46.2.4 Math.ArcCotH Function

Calculates the inverse hyperbolic cotangent of a given number.

Pascal

```
function ArcCotH(const X: Extended): Extended;
```

C++

```
Extended ArcCotH(const Extended X);
```

File

Math

Description

ArcCot (see page 4720) returns the inverse hyperbolic cotangent of X.

See Also

Cot (see page 4727)

ArcCoth

46.2.5 Math.ArcCsc Function

Calculates the inverse cosecant of a given number.

Pascal

```
function ArcCsc(const X: Extended): Extended;
```

C++

```
Extended ArcCsc(const Extended X);
```

File

Math

Description

ArcCsc returns the inverse cosecant of X.

See Also

Csc (see page 4729)

ArcCsch

46.2.6 Math.ArcCscH Function

Calculates the inverse hyperbolic cosecant of a given number.

Pascal

```
function ArcCscH(const X: Extended): Extended;
```

C++

```
Extended ArcCscH(const Extended X);
```

File

Math

Description

ArcCsc (see page 4721) returns the inverse hyperbolic cosecant of X.

See Also

Csc (see page 4729)

Csch

46.2.7 Math.ArcSec Function

Calculates the inverse secant of a given number.

Pascal

```
function ArcSec(const X: Extended): Extended;
```

C++

```
Extended ArcSec(const Extended X);
```

File

Math

Description

ArcSec returns the inverse secant of X.

See Also

Sec (see page 4758)

ArcSech

46.2.8 Math.ArcSecH Function

Calculates the inverse hyperbolic secant of a given number.

Pascal

```
function ArcSecH(const X: Extended): Extended;
```

C++

```
Extended ArcSecH(const Extended X);
```

File

Math

Description

ArcSec (see page 4722) returns the inverse hyperbolic secant of X.

See Also

Sec (see page 4758)

Sech

46.2.9 Math.ArcSin Function

Calculates the inverse sine of a given number.

Pascal

```
function ArcSin(const X: Extended): Extended; overload;
function ArcSin(const X: Double): Double; overload;
function ArcSin(const X: Single): Single; overload;
```

C++

```
Extended ArcSin(const Extended X);
Double ArcSin(const Double X);
Single ArcSin(const Single X);
```

File

Math

Description

ArcSin returns the inverse sine of X. X must be between -1 and 1. The return value will be in the range [-Pi/2..Pi/2], in radians.

See Also

Sin

ArcSinh (see page 4723)

ArcCos (see page 4719)

46.2.10 Math.ArcSinh Function

Calculates the inverse hyperbolic sine of a given number.

Pascal

```
function ArcSinh(const X: Extended): Extended;
```

C++

```
Extended ArcSinh(const Extended X);
```

File

Math

Description

ArcSinh returns the inverse hyperbolic sine of X.

See Also

Sinh (see page 4762)

46.2.11 Math.ArcTan2 Function

Calculates the arctangent angle and quadrant of a given number.

Pascal

```
function ArcTan2(const Y: Extended; const X: Extended): Extended;
```

C++

```
Extended ArcTan2(const Extended Y, const Extended X);
```

File

Math

Description

ArcTan2 calculates ArcTan(Y/X), and returns an angle in the correct quadrant. The values of X and Y must be between –2^64 and 2^64. In addition, the value of X can't be 0. The return value will fall in the range from -Pi to Pi radians.

46.2.12 Math.ArcTanh Function

Calculates the inverse hyperbolic tangent of a given number.

Pascal

```
function ArcTanh(const X: Extended): Extended;
```

C++

```
Extended ArcTanh(const Extended X);
```

File

Math

Description

ArcTanh returns the inverse hyperbolic tangent of X. The value of X must be between -1 and 1 (inclusive).

See Also

Tanh (see page 4767)

46.2.13 Math.Ceil Function

Rounds variables up toward positive infinity.

Pascal

```
function Ceil(const X: Extended): Integer;
```

C++

```
int Ceil(const Extended X);
```

File

Math

Description

Call Ceil to obtain the lowest integer greater than or equal to X. The absolute value of X must be less than MaxInt. For example:

Ceil(-2.8) = -2

Ceil(2.8) = 3

Ceil(-1.0) = -1

See Also

Trunc

Floor (see page 4733)

Ceil

46.2.14 Math.ClearExceptions Function

Clears exception bits in FPU status word.

Pascal

```
procedure ClearExceptions(RaisePending: Boolean = True);
```

C++

```
ClearExceptions(Boolean RaisePending = True);
```

File

Math

Description

ClearExceptions clears the exception bits in the FPU status word. If RaisePending is true, any pending floating point exceptions cause an error to be raised.

See Also

GetExceptionMask (see page 4735)

SetExceptionMask (see page 4759)

Set8087CW

46.2.15 Math.CompareValue Function

Returns the relationship between two numeric values.

Pascal

```
function CompareValue(const A: Extended; const B: Extended; Epsilon: Extended = 0):
TValueRelationship; overload;
function CompareValue(const A: Double; const B: Double; Epsilon: Double = 0):
TValueRelationship; overload;
function CompareValue(const A: Single; const B: Single; Epsilon: Single = 0):
TValueRelationship; overload;
function CompareValue(const A: Integer; const B: Integer): TValueRelationship; overload;
function CompareValue(const A: Int64; const B: Int64): TValueRelationship; overload;
```

C++

```
TValueRelationship CompareValue(const Extended A, const Extended B, Extended Epsilon = 0);
TValueRelationship CompareValue(const Double A, const Double B, Double Epsilon = 0);
TValueRelationship CompareValue(const Single A, const Single B, Single Epsilon = 0);
TValueRelationship CompareValue(const int A, const int B);
TValueRelationship CompareValue(const Int64 A, const Int64 B);
```

File

Math

Description

Call CompareValue to determine the relationship between two numeric values. When comparing floating-point values, CompareValue lets you specify a maximum difference to use when comparing values, so that they are considered the same if they are within that amount.

A and B are the values to compare.

Epsilon is the maximum amount by which A and B can differ and still be considered the same value.

CompareValue returns

LessThanValue if A is less than B (by more than Epsilon if A and B are floating-point numbers).

EqualsValue if A is equivalent to B (the same, or within Epsilon if A and B are floating-point numbers).

GreaterThanValue if A is larger than B (by more than Epsilon if A and B are floating-point numbers).

See Also

SameValue (see page 4757)

VarCompareValue

46.2.16 Math.Cosecant Function

Returns the cosecant of an angle.

Pascal

```
function Cosecant(const X: Extended): Extended;
```

C++

```
Extended Cosecant(const Extended X);
```

File

Math

Description

Use the Cosecant to calculate the cosecant of X, where X is an angle in radians. The cosecant is calculated as 1/ Sin(X).

See Also

ArcCsc (see page 4721)

CscH (see page 4729)

Secant (see page 4758)

Sin

46.2.17 Math.Cosh Function

Calculates the hyperbolic cosine of an angle.

Pascal

```
function Cosh(const X: Extended): Extended;
```

C++

```
Extended Cosh(const Extended X);
```

File

Math

Description

Use the Cosh to calculate the hyperbolic cosine of X.

See Also

Cos

ArcCosh (see page 4720)

Sinh (see page 4762)

46.2.18 Math.Cot Function

Calculates the cotangent of an angle.

Pascal

```
function Cot(const X: Extended): Extended;
```

C++

```
Extended Cot(const Extended X);
```

File

Math

Description

Call Cot to obtain the cotangent of X. The cotangent is calculated using the formula 1 / Tan (see page 4766)(X).

See Also

Tan (see page 4766)

Sin

Cos

ArcTan

46.2.19 Math.Cotan Function

Calculates the cotangent of an angle.

Pascal

```
function Cotan(const X: Extended): Extended;
```

C++

```
Extended Cotan(const Extended X);
```

File

Math

Description

Call Cotan to obtain the cotangent of X. The cotangent is calculated using the formula

1 / Tan (see page 4766)(X)

Note: Do not call Cotan with X = 0.

See Also

Tan (see page 4766)

Sin

Cos

ArcTan

46.2.20 Math.CotH Function

Calculates the hyperbolic cotangent of an angle.

Pascal

```
function CotH(const X: Extended): Extended;
```

C++

```
Extended CotH(const Extended X);
```

File

Math

Description

Call CotH to obtain the hyperbolic cotangent of X, where X is an angle in Radians.

See Also

Cot (see page 4727)

ArcCot (see page 4720)

46.2.21 Math.Csc Function

Returns the cosecant of an angle.

Pascal

```
function Csc(const X: Extended): Extended;
```

C++

```
Extended Csc(const Extended X);
```

File

Math

Description

Use the Csc to calculate the cosecant of X, where X is an angle in radians.

See Also

ArcCsc (see page 4721)

CscH (see page 4729)

Sec (see page 4758)

Sin

46.2.22 Math.CscH Function

Returns the hyperbolic cosecant of an angle.

Pascal

```
function CscH(const X: Extended): Extended;
```

C++

```
Extended CscH(const Extended X);
```

File

Math

Description

Use the CscH to calculate the hyperbolic cosecant of X, where X is an angle in radians.

See Also

ArcCscH (see page 4722)

Csc (see page 4729)

SecH (see page 4759)

46.2.23 Math.CycleToDeg Function

Converts an angle measurement from cycles to degrees.

Pascal

```
function CycleToDeg(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToDeg(const Extended Cycles);
```

File

Math

Description

CycleToDeg converts angles measured in cycles into degrees, where

degrees = cycles * 360.

46.2.24 Math.CycleToGrad Function

Converts an angle measurement from cycles to grads.

Pascal

```
function CycleToGrad(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToGrad(const Extended Cycles);
```

File

Math

Description

CycleToGrad converts angles measured in cycles into grads.

46.2.25 Math.CycleToRad Function

Converts an angle measurement from cycles to radians.

Pascal

```
function CycleToRad(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToRad(const Extended Cycles);
```

File

Math

Description

CycleToRad converts angles measured in cycles into radians, where

radians = 2pi * cycles.

46.2.26 Math.DegToCycle Function

Returns the value of a degree measurement expressed in cycles.

Pascal

```
function DegToCycle(const Degrees: Extended): Extended;
```

C++

```
Extended DegToCycle(const Extended Degrees);
```

File

Math

Description

Use DegToCycle to convert angles expressed in degrees to the corresponding value in cycles.

46.2.27 Math.DegToGrad Function

Returns the value of a degree measurement expressed in grads.

Pascal

```
function DegToGrad(const Degrees: Extended): Extended;
```

C++

```
Extended DegToGrad(const Extended Degrees);
```

File

Math

Description

Use DegToGrad to convert angles expressed in degrees to the corresponding value in grads.

46.2.28 Math.DegToRad Function

Returns the value of a degree measurement expressed in radians.

Pascal

```
function DegToRad(const Degrees: Extended): Extended;
```

C++

```
Extended DegToRad(const Extended Degrees);
```

File

Math

Description

Use DegToRad to convert angles expressed in degrees to the corresponding value in radians, where

radians = degrees(pi/180).

46.2.29 Math.DivMod Function

Returns the result of an integer division, including the remainder.

Pascal

```
procedure DivMod(Dividend: Cardinal; Divisor: Word; var Result: Word; var Remainder: Word);
```

C++

```
DivMod(unsigned Dividend, Word Divisor, Word Result, Word Remainder);
```

File

Math

Description

Call DivMod to perform 16-bit integer division and fetch the remainder as well, all in one operation.

Dividend is the integer into which you are dividing.

Divisor is the value by which to divide Dividend.

Result returns the result of the integer division.

Remainder returns the remainder (the difference between Result * Divisor and Dividend).

46.2.30 Math.DoubleDecliningBalance Function

Calculates the depreciation of an asset using the double-declining balance method.

Pascal

```
function DoubleDecliningBalance(const Cost: Extended; const Salvage: Extended; Life: Integer;
Period: Integer): Extended;
```

C++

```
Extended DoubleDecliningBalance(const Extended Cost, const Extended Salvage, int Life, int
Period);
```

File

Math

Description

DoubleDecliningBalance determines accelerated depreciation values for an asset, given the initial cost, life expectancy, end value, and depreciation period.

46.2.31 Math.EnsureRange Function

Returns the closest value to a specified value within a specified range.

Pascal

```
function EnsureRange(const AValue: Integer; const AMin: Integer; const AMax: Integer):
Integer; overload;
function EnsureRange(const AValue: Int64; const AMin: Int64; const AMax: Int64): Int64;
overload;
function EnsureRange(const AValue: Double; const AMin: Double; const AMax: Double): Double;
overload;
```

C++

```
int EnsureRange(const int AValue, const int AMin, const int AMax);
Int64 EnsureRange(const Int64 AValue, const Int64 AMin, const Int64 AMax);
Double EnsureRange(const Double AValue, const Double AMin, const Double AMax);
```

File

Math

Description

EnsureRange returns a value that is guaranteed to be within a specified range.

AValue is the value you want as long as it is in range. If AValue is in range, EnsureRange returns AValue.

AMin is the minimum acceptable value. If AValue is less than AMin, EnsureRange returns AMin.

AMax is the maximum acceptable value. If AValue is greater than AMax, EnsureRange returns AMax.

See Also

VarEnsureRange

InRange (see page 4738)

46.2.32 Math.Floor Function

Rounds variables toward negative infinity.

Pascal

```
function Floor(const X: Extended): Integer;
```

C++

```
int Floor(const Extended X);
```

File

Math

Description

Call Floor to obtain the highest integer less than or equal to X. For example:

Floor(-2.8) = -3

Floor(2.8) = 2

Floor(-1.0) = -1

Note: The absolute value of X must be less than MaxInt.

See Also

Trunc

Ceil (see page 4725)

46.2.33 Math.Frexp Function

Separates the Mantissa and Exponent of X

Pascal

```
procedure Frexp(const X: Extended; var Mantissa: Extended; var Exponent: Integer);
```

C++

```
Frexp(const Extended X, Extended Mantissa, int Exponent);
```

File

Math

Description

Frexp returns the mantissa of X as Mantissa and the exponent as Exponent.

46.2.34 Math.FutureValue Function

Calculates the future value of an investment.

Pascal

```
function FutureValue(const Rate: Extended; NPeriods: Integer; const Payment: Extended; const
PresentValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended FutureValue(const Extended Rate, int NPeriods, const Extended Payment, const Extended
PresentValue, TPaymentTime PaymentTime);
```

File

Math

Description

FutureValue returns the future value of an investment of PresentValue (see page 4753) where Payment (see page 4751) is invested for NPeriods at the rate of Rate per period. The PaymentTime parameter indicates whether the investment is an ordinary annuity or an annuity due (enter ptEndOfPeriod if payments are at the end of each period, ptStartOfPeriod if they are at the beginning).

46.2.35 Math.GetExceptionMask Function

Returns the exception mask from the FPU control word.

Pascal

```
function GetExceptionMask: TFPUExceptionMask;
```

C++

```
TFPUExceptionMask GetExceptionMask();
```

File

Math

Description

Call GetExceptionMask to ascertain the types of exceptions the FPU (floating-point unit) generates. Any element set in this mask prevents the FPU from raising the corresponding type of exception. Instead, the FPU returns its best attempt at a value – often NaN (see page 4780), Infinity (see page 4776), or NegInfinity (see page 4781). The value depends on the operation and the current rounding mode.

See Also

Set8087CW

SetExceptionMask (see page 4759)

NaN (see page 4780)

Infinity (see page 4776)

NegInfinity (see page 4781)

IsInfinite (see page 4741)

IsNaN

46.2.36 Math.GetPrecisionMode Function

Returns the FPU precision control mode.

Pascal

```
function GetPrecisionMode: TFPUPrecisionMode;
```

C++

```
TFPUPrecisionMode GetPrecisionMode();
```

File

Math

Description

Call GetPrecisionMode to ascertain the level of precision that the FPU (floating-point unit) uses for floating-point calculations. The precision control mode can be any of the following values:

Value	Meaning
pmSingle	single precision
pmReserved	not used
pmDouble	double precision
pmExtended	extended precision

See Also

Set8087CW

SetPrecisionMode (see page 4760)

46.2.37 Math.GetRoundMode Function

Returns the FPU rounding mode.

Pascal

```
function GetRoundMode: TFPURoundingMode;
```

C++

```
TFPURoundingMode GetRoundMode();
```

File

Math

Description

Call GetRoundingMode to ascertain how the FPU handles rounding issues. The rounding mode can be any of the following values:

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

Set8087CW

SetRoundMode (see page 4760)

46.2.38 Math.GradToCycle Function

Converts grad measurements to cycles.

Pascal

```
function GradToCycle(const Grads: Extended): Extended;
```

C++

```
Extended GradToCycle(const Extended Grads);
```

File

Math

Description

GradToCycle converts angles measured in grads into cycles.

46.2.39 Math.GradToDeg Function

Converts grad measurements to degrees.

Pascal

```
function GradToDeg(const Grads: Extended): Extended;
```

C++

```
Extended GradToDeg(const Extended Grads);
```

File

Math

Description

GradToDeg converts angles measured in grads into degrees.

46.2.40 Math.GradToRad Function

Converts grad measurements to radians.

Pascal

```
function GradToRad(const Grads: Extended): Extended;
```

C++

```
Extended GradToRad(const Extended Grads);
```

File

Math

Description

GradToRad converts angles measured in grads into radians, where

radians = grads(pi/200).

46.2.41 Math.Hypot Function

Calculates the length of the hypotenuse.

Pascal

```
function Hypot(const X: Extended; const Y: Extended): Extended;
```

C++

```
Extended Hypot(const Extended X, const Extended Y);
```

File

Math

Description

Hypot returns the length of the hypotenuse of a right triangle. Specify the lengths of the sides adjacent to the right angle in X and Y. Hypot uses the formula

Sqrt(X**2 + Y**2)

46.2.42 Math.IfThen Function

Conditionally returns one of two specified values.

Pascal

```
function IfThen(AValue: Boolean; const ATrue: Integer; const AFalse: Integer = 0): Integer;
overload;
function IfThen(AValue: Boolean; const ATrue: Int64; const AFalse: Int64 = 0): Int64; overload;
function IfThen(AValue: Boolean; const ATrue: Double; const AFalse: Double = 0.0): Double;
overload;
```

C++

```
int IfThen(Boolean AValue, const int ATrue, const int AFalse = 0);
Int64 IfThen(Boolean AValue, const Int64 ATrue, const Int64 AFalse = 0);
Double IfThen(Boolean AValue, const Double ATrue, const Double AFalse = 0.0);
```

File

Math

Description

IfThen checks the expression passed as AValue and returns ATrue if it evaluates to true, or AFalse if it evaluates to false. In Delphi, if the AFalse parameter is omitted, IfThen returns 0 or an empty string when AValue evaluates to False.

46.2.43 Math.InRange Function

Indicates whether a value falls within a specified range.

Pascal

```
function InRange(const AValue: Integer; const AMin: Integer; const AMax: Integer): Boolean;
overload;
function InRange(const AValue: Int64; const AMin: Int64; const AMax: Int64): Boolean; overload;
function InRange(const AValue: Double; const AMin: Double; const AMax: Double): Boolean;
overload;
```

C++

```
Boolean InRange(const int AValue, const int AMin, const int AMax);
Boolean InRange(const Int64 AValue, const Int64 AMin, const Int64 AMax);
Boolean InRange(const Double AValue, const Double AMin, const Double AMax);
```

File

Math

Description

InRange returns true if AValue is greater than or equal to AMin and less than or Equal to AMax. It returns false if AValue is less than AMin or greater than AMax.

See Also

VarInRange

EnsureRange (see page 4733)

46.2.44 Math.InterestPayment Function

Calculates the interest portion of a loan payment.

Pascal

```
function InterestPayment(const Rate: Extended; Period: Integer; NPeriods: Integer; const
PresentValue: Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended InterestPayment(const Extended Rate, int Period, int NPeriods, const Extended
PresentValue, const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

Call InterestPayment to calculate the portion of a loan payment that reflects the interest.

Rate represents the fixed periodic interest rate.

Period identifies the payment period.

NPeriods is the number of periods of the loan.

PresentValue (see page 4753) represents the amount borrowed (the principal).

FutureValue (see page 4734) is the future value of the investment.

PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

46.2.45 Math.InterestRate Function

Returns the interest rate required to increase PresentValue (see page 4753) to FutureValue (see page 4734).

Pascal

```
function InterestRate(NPeriods: Integer; const Payment: Extended; const PresentValue:
Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended InterestRate(int NPeriods, const Extended Payment, const Extended PresentValue, const
Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

InterestRate calculates the interest rate required in order for an investment of PresentValue (see page 4753), with periodic payments of Payment (see page 4751), to be worth FutureValue (see page 4734) within NPeriods compounding periods. If NPeriods represents years, an annual interest rate results; if NPeriods represents months, a monthly interest rate results, and so on. The PaymentTime parameter indicates whether the cash flows occur at the beginning or end of the period.

46.2.46 Math.InternalRateOfReturn Function

Calculates the internal rate of return for an investment.

Pascal

```
function InternalRateOfReturn(const Guess: Extended; const CashFlows: array of Double):
Extended;
```

C++

```
Extended InternalRateOfReturn(const Extended Guess, const array of Double CashFlows);
```

File

Math

Description

InternalRateOfReturn determines the internal rate of return on an investment. It references an array that contains cash flow information and uses the supplied internal rate of return estimate to calculate results.

Before using this function, define an array containing expected cash flow amounts over a period of time. It is assumed that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. The following amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

Specify the estimated rate of return as the Guess parameter. Pass the array of expected cash flow amounts as the CashFlows parameter. In C++, the CashFlows_Size parameter gives the index of the last entry in CashFlows (one less than the number of entries).

46.2.47 Math.IntPower Function

Calculates the integral power of a base value.

Pascal

```
function IntPower(const Base: Extended; const Exponent: Integer): Extended;
```

C++

```
Extended IntPower(const Extended Base, const int Exponent);
```

File

Math

Description

IntPower raises Base to the power specified by Exponent.

See Also

Power (see page 4753)

46.2.48 Math.IsInfinite Function

Indicates when a variable or expression represents an infinite value.

Pascal

```
function IsInfinite(const AValue: Double): Boolean;
```

C++

```
Boolean IsInfinite(const Double AValue);
```

File

Math

Description

Use IsInfinite to test whether the value specified by AValue represents positive or negative infinity. Use the Sign (see page 4761) function to distinguish positive infinity from negative infinity.

See Also

GetExceptionMask (see page 4735)

NegInfinity (see page 4781)

IsNan (see page 4742)

Infinity (see page 4776)

IsZero (see page 4742)

Sign (see page 4761)

46.2.49 Math.IsNan Function

Indicates when a variable or expression does not evaluate to a numeric value.

Pascal

```
function IsNan(const AValue: Double): Boolean; overload;
function IsNan(const AValue: Single): Boolean; overload;
function IsNan(const AValue: Extended): Boolean; overload;
```

C++

```
Boolean IsNan(const Double AValue);
Boolean IsNan(const Single AValue);
Boolean IsNan(const Extended AValue);
```

File

Math

Description

Use IsNan to test whether the value specified by AValue represents NaN (see page 4780) (not a number). Note that NaN (see page 4780) is distinct from positive and negative infinity, which can be detected using the IsInfinite (see page 4741) function.

See Also

GetExceptionMask (see page 4735)

Nan

IsInfinite (see page 4741)

46.2.50 Math.IsZero Function

Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.

Pascal

```
function IsZero(const A: Extended; Epsilon: Extended = 0): Boolean; overload;
function IsZero(const A: Double; Epsilon: Double = 0): Boolean; overload;
function IsZero(const A: Single; Epsilon: Single = 0): Boolean; overload;
```

C++

```
Boolean IsZero(const Extended A, Extended Epsilon = 0);
Boolean IsZero(const Double A, Double Epsilon = 0);
Boolean IsZero(const Single A, Single Epsilon = 0);
```

File

Math

Description

Use IsZero to test whether the value specified by AValue is zero, or differs from zero by at most Epsilon.

See Also

SameValue (see page 4757)

IsInfinite (see page 4741)

Sign (see page 4761)

46.2.51 Math.Ldexp Function

Calculates X times (2 to the power of P).

Pascal

```
function Ldexp(const X: Extended; const P: Integer): Extended;
```

C++

```
Extended Ldexp(const Extended X, const int P);
```

File

Math

Description

Ldexp returns X times (2 to the power of P).

46.2.52 Math.LnXP1 Function

Returns the natural log of (X+1)

Pascal

```
function LnXP1(const X: Extended): Extended;
```

C++

```
Extended LnXP1(const Extended X);
```

File

Math

Description

LnXP1 returns the natural logarithm of (X+1). Use LnXP1 when X is a value near 0.

46.2.53 Math.Log10 Function

Calculates log base 10.

Pascal

```
function Log10(const X: Extended): Extended;
```

C++

```
Extended Log10(const Extended X);
```

File

Math

Description

Log10 returns the log base 10 of X.

See Also

Log2 (see page 4744)

Ln

LogN (see page 4744)

46.2.54 Math.Log2 Function

Calculates log base 2.

Pascal

```
function Log2(const X: Extended): Extended;
```

C++

```
Extended Log2(const Extended X);
```

File

Math

Description

Log2 returns the log base 2 of X.

See Also

Log10 (see page 4743)

Ln

LogN (see page 4744)

46.2.55 Math.LogN Function

Calculates the log of X for a specified base.

Pascal

```
function LogN(const Base: Extended; const X: Extended): Extended;
```

C++

```
Extended LogN(const Extended Base, const Extended X);
```

File

Math

Description

LogN returns the log base Base of X.

See Also

Log2 (see page 4744)

Log10 (see page 4743)

Ln

46.2.56 Math.Max Function

Returns the greater of two numeric values (Delphi only).

Pascal

```
function Max(const A: Integer; const B: Integer): Integer; overload;
function Max(const A: Int64; const B: Int64): Int64; overload;
function Max(const A: Single; const B: Single): Single; overload;
function Max(const A: Double; const B: Double): Double; overload;
function Max(const A: Extended; const B: Extended): Extended; overload;
```

C++

```
int Max(const int A, const int B);
Int64 Max(const Int64 A, const Int64 B);
Single Max(const Single A, const Single B);
Double Max(const Double A, const Double B);
Extended Max(const Extended A, const Extended B);
```

File

Math

Description

In Delphi, call Max to compare two numeric values. Max returns the greater value of the two.

See Also

MaxValue (see page 4746)

Min (see page 4747)

MaxIntValue (see page 4745)

46.2.57 Math.MaxIntValue Function

Returns the largest signed value in an integer array.

Pascal

```
function MaxIntValue(const Data: array of Integer): Integer;
```

C++

```
int MaxIntValue(const array of Integer Data);
```

File

Math

Description

Call MaxIntValue to obtain the largest integer from the Data array. In C++, Data_Size is the index of the last element in the array (one less than the number of elements).

See Also

MaxValue (see page 4746)

MinIntValue (see page 4748)

46.2.58 Math.MaxValue Function

Returns the largest signed value in an array.

Pascal

```
function MaxValue(const Data: array of Double): Double;
```

C++

```
Double MaxValue(const array of Double Data);
```

File

Math

Description

MaxValue returns the largest signed value in the Data array. In C++, the Data_Size parameter gives the index of the last element of the array (one less than the number of elements).

See Also

MaxIntValue (see page 4745)

MinValue (see page 4748)

46.2.59 Math.Mean Function

Returns the average of all values in an array.

Pascal

```
function Mean(const Data: array of Double): Extended;
```

C++

```
Extended Mean(const array of Double Data);
```

File

Math

Description

Mean calculates the arithmetic average of all the values in the Data array parameter. In C++, the Data_Size parameter gives the index of the last element of the array (one less than the number of elements).

See Also

MeanAndStdDev (see page 4747)

46.2.60 Math.MeanAndStdDev Function

Calculates the mean and standard deviation of array elements.

Pascal

```
procedure MeanAndStdDev(const Data: array of Double; var Mean: Extended; var StdDev: Extended);
```

C++

```
MeanAndStdDev(const array of Double Data, Extended Mean, Extended StdDev);
```

File

Math

Description

MeanAndStdDev calculates Mean (see page 4746) and standard deviation in one pass. Using this is twice as fast as calculating them separately. Precision may be lost when the Mean (see page 4746) is very large (> 10e7) or the variance is very small.

Data specifies the data to be summarized. In C++, Data_Size is the index of the last element in the Data array (one less than the number of elements). The mean is returned as Mean (see page 4746) and the standard deviation is returned as StdDev (see page 4763).

Note: If the sample size is 1 (that is, if Data contains only one entry), StdDev (see page 4763) returns the value of Data[0] and not zero.

See Also

Mean (see page 4746)

StdDev (see page 4763)

46.2.61 Math.Min Function

Returns the lesser of two numeric values (Delphi only).

Pascal

```
function Min(const A: Integer; const B: Integer): Integer; overload;
function Min(const A: Int64; const B: Int64): Int64; overload;
function Min(const A: Single; const B: Single): Single; overload;
function Min(const A: Double; const B: Double): Double; overload;
function Min(const A: Extended; const B: Extended): Extended; overload;
```

C++

```
int Min(const int A, const int B);
Int64 Min(const Int64 A, const Int64 B);
Single Min(const Single A, const Single B);
Double Min(const Double A, const Double B);
Extended Min(const Extended A, const Extended B);
```

File

Math

Description

Call Min to compare two numeric values in Delphi. Min returns the smaller value of the two.

See Also

MinValue (see page 4748)

Max (see page 4745)

MinIntValue (see page 4748)

46.2.62 Math.MinIntValue Function

Returns the smallest signed value in an integer array.

Pascal

```
function MinIntValue(const Data: array of Integer): Integer;
```

C++

```
int MinIntValue(const array of Integer Data);
```

File

Math

Description

Call MinIntValue to obtain the smallest integer in the Data array. In C++, the Data_Size parameter specifies the index of the last element in the array (one less than the total number of elements).

See Also

MaxIntValue (see page 4745)

MinValue (see page 4748)

46.2.63 Math.MinValue Function

Returns smallest signed value in an array.

Pascal

```
function MinValue(const Data: array of Double): Double;
```

C++

```
Double MinValue(const array of Double Data);
```

File

Math

Description

MinValue returns the smallest signed value in the Data array. The Data_Size parameter specifies the index of the last element in the array (one less than the total number of elements).

See Also

MaxValue (see page 4746)

MinIntValue (see page 4748)

46.2.64 Math.MomentSkewKurtosis Function

Calculates the mean, variance, skew, and kurtosis.

Pascal

```
procedure MomentSkewKurtosis(const Data: array of Double; var M1: Extended; var M2: Extended;
var M3: Extended; var M4: Extended; var Skew: Extended; var Kurtosis: Extended);
```

C++

```
MomentSkewKurtosis(const array of Double Data, Extended M1, Extended M2, Extended M3, Extended
M4, Extended Skew, Extended Kurtosis);
```

File

Math

Description

MomentSkewKurtosis calculates the core factors of statistical analysis: the first four moments plus the coefficients of skew and kurtosis.

Data specifies the data to be analyzed. In C++, Data_Size is the index of the last element in the Data array (one less than the number of elements). M1, M2, M3 and M4 return the first four moments (M1 is the mean, M2 is the variance, and so on). Skew reflects symmetry of distribution. Kurtosis reflects flatness of distribution.

46.2.65 Math.NetPresentValue Function

Calculates the current value from an array of estimated cash flow values.

Pascal

```
function NetPresentValue(const Rate: Extended; const CashFlows: array of Double; PaymentTime:
TPaymentTime): Extended;
```

C++

```
Extended NetPresentValue(const Extended Rate, const array of Double CashFlows, TPaymentTime
PaymentTime);
```

File

Math

Description

NetPresentValue calculates the current value of an array of estimated cash flow values, discounted at the given interest rate of Rate.

CashFlows is the array of estimated cash flow values. In C++, CashFlows_Size is the index of the last element in that array (one less than the number of elements).

PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

NetPresentValue helps determine how much an investment is currently worth, based on expected earnings, although its accuracy depends on the accuracy of the cash flows in the array.

46.2.66 Math.Norm Function

Returns the Euclidean 'L-2' norm.

Pascal

```
function Norm(const Data: array of Double): Extended;
```

C++

```
Extended Norm(const array of Double Data);
```

File

Math

Description

Norm returns the Euclidean 'L-2' norm for the values in the Data array. The 'L-2' norm is the square root of the sum of squares.

Note: In C++, Data is an array of values, and Data_Size is the index of the last value in the array (one less than the number of values).

See Also

SumOfSquares (see page 4765)

46.2.67 Math.NumberOfPeriods Function

Returns the number of payment periods for an investment.

Pascal

```
function NumberOfPeriods(const Rate: Extended; Payment: Extended; const PresentValue:
Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended NumberOfPeriods(const Extended Rate, Extended Payment, const Extended PresentValue,
const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

NumberOfPeriods computes the number of payment periods required for an investment of PresentValue (see page 4753) to reach a value of FutureValue (see page 4734), while making regular payments of Payment (see page 4751) and accruing interest at the rate of Rate per compounding period. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

46.2.68 Math.Payment Function

Calculates a fully amortized payment.

Pascal

```
function Payment(Rate: Extended; NPeriods: Integer; const PresentValue: Extended; const
FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended Payment(Extended Rate, int NPeriods, const Extended PresentValue, const Extended
FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

Payment calculates the fully amortized payment of borrowing PresentValue (see page 4753) at Rate percent per period over NPeriods. It assumes that interest is paid at the end of each period.

FutureValue (see page 4734) is the value that the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

46.2.69 Math.PeriodPayment Function

Returns the principal amount from a full payment.

Pascal

```
function PeriodPayment(const Rate: Extended; Period: Integer; NPeriods: Integer; const
PresentValue: Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended PeriodPayment(const Extended Rate, int Period, int NPeriods, const Extended
PresentValue, const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

PeriodPayment gives the part of the payment that is principal. The InterestPayment (see page 4739) function gives the part of the payment that is interest.

The Rate parameter is the interest rate. Period is the number of periods into the loan for which the principal is desired, and NPeriods is the number of periods of the loan. FutureValue (see page 4734) is the value the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

46.2.70 Math.Poly Function

Evaluates a uniform polynomial of one variable at the value X.

Pascal

```
function Poly(const X: Extended; const Coefficients: array of Double): Extended;
```

C++

```
Extended Poly(const Extended X, const array of Double Coefficients);
```

File

Math

Description

Call Poly to evaluate the polynomial represented by the Coefficients parameter at the point where the variable equals the value of the X parameter. The coefficients are ordered in increasing powers of X, as shown in the following Delphi-like example:

Coefficients[0] + Coefficients[1]*X + ... + Coefficients[N]*(X**N)

Note: In C++, the Coefficients_Size parameter is N, the highest power to which X is raised.

46.2.71 Math.PopnStdDev Function

Calculates the population standard deviation.

Pascal

```
function PopnStdDev(const Data: array of Double): Extended;
```

C++

```
Extended PopnStdDev(const array of Double Data);
```

File

Math

Description

PopnStdDev calculates the population standard deviation (the square root of the population variance) of all values in the Data array parameter. In C++, the Data_Size parameter indicates the index of the last element in Data (one less than the number of data points).

This function is used in some business and financial calculations.

See Also

StdDev (see page 4763)

PopnVariance (see page 4752)

46.2.72 Math.PopnVariance Function

Calculates the population variance.

Pascal

```
function PopnVariance(const Data: array of Double): Extended;
```

C++

```
Extended PopnVariance(const array of Double Data);
```

File

Math

Description

PopnVariance calculates the population variance of all values in the Data array parameter, using the n method (biased): TotalVariance (see page 4767) / n. In C++,. the Data_Size parameter is the index of the last value in the Data array (one less than the number of data items).

See Also

Variance (see page 4768)

TotalVariance (see page 4767)

PopnStdDev (see page 4752)

46.2.73 Math.Power Function

Raises Base to any power.

Pascal

```
function Power(const Base: Extended; const Exponent: Extended): Extended; overload;
function Power(const Base: Double; const Exponent: Double): Double; overload;
function Power(const Base: Single; const Exponent: Single): Single; overload;
```

C++

```
Extended Power(const Extended Base, const Extended Exponent);
Double Power(const Double Base, const Double Exponent);
Single Power(const Single Base, const Single Exponent);
```

File

Math

Description

Power raises Base to any power. For fractional exponents or exponents greater than MaxInt, Base must be greater than 0.

See Also

IntPower (see page 4741)

Sqrt

46.2.74 Math.PresentValue Function

Calculates the present value of an investment.

Pascal

```
function PresentValue(const Rate: Extended; NPeriods: Integer; const Payment: Extended; const
FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended PresentValue(const Extended Rate, int NPeriods, const Extended Payment, const
Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

PresentValue calculates the present value of an investment where Payment (see page 4751) is received for NPeriods and is discounted at the rate of Rate per period. FutureValue (see page 4734) is the value the investment may reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

46.2.75 Math.RadToCycle Function

Converts radians to cycles.

Pascal

```
function RadToCycle(const Radians: Extended): Extended;
```

C++

```
Extended RadToCycle(const Extended Radians);
```

File

Math

Description

Use RadToCycle to convert angles measured in radians into cycles, where

cycles = radians/(2pi).

46.2.76 Math.RadToDeg Function

Converts radians to degrees.

Pascal

```
function RadToDeg(const Radians: Extended): Extended;
```

C++

```
Extended RadToDeg(const Extended Radians);
```

File

Math

Description

Use RadToDeg to convert angles measured in radians to degrees, where

degrees = radians(180/pi).

46.2.77 Math.RadToGrad Function

Converts radians to grads.

Pascal

```
function RadToGrad(const Radians: Extended): Extended;
```

C++

```
Extended RadToGrad(const Extended Radians);
```

File

Math

Description

Use RadToGrad to convert angles measured in radians to grads, where

grads = radians(200/pi).

46.2.78 Math.RandG Function

Generates random numbers with Gaussian distribution.

Pascal

```
function RandG(Mean: Extended; StdDev: Extended): Extended;
```

C++

```
Extended RandG(Extended Mean, Extended StdDev);
```

File

Math

Description

RandG produces random numbers with Gaussian distribution about the Mean (see page 4746). This is useful for simulating data with sampling errors and expected deviations from the Mean (see page 4746).

46.2.79 Math.RandomFrom Function

Returns a randomly selected element from an array.

Pascal

```
function RandomFrom(const AValues: array of Integer): Integer; overload;
function RandomFrom(const AValues: array of Int64): Int64; overload;
function RandomFrom(const AValues: array of Double): Double; overload;
```

C++

```
int RandomFrom(const array of Integer AValues);
Int64 RandomFrom(const array of Int64 AValues);
Double RandomFrom(const array of Double AValues);
```

File

Math

Description

RandomFrom returns a randomly selected element from the array passed as the AValues argument. In C++, AValues_Size is

the index of the last array element (one less than the number of elements.)

To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomFrom.

See Also

Randomize

RandSeed

46.2.80 Math.RandomRange Function

Returns a random integer from a specified range.

Pascal

```
function RandomRange(const AFrom: Integer; const ATo: Integer): Integer;
```

C++

```
int RandomRange(const int AFrom, const int ATo);
```

File

Math

Description

RandomRange returns a random integer from the range that extends between AFrom and ATo (non-inclusive). RandomRange can handle negative ranges (where AFrom is greater than ATo).

To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomRange.

See Also

Randomize

RandSeed

RandomFrom (see page 4755)

Random

46.2.81 Math.RoundTo Function

Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".

Pascal

```
function RoundTo(const AValue: Extended; const ADigit: TRoundToEXRangeExtended): Extended;
```

C++

```
Extended RoundTo(const Extended AValue, const TRoundToEXRangeExtended ADigit);
```

File

Math

Description

Call RoundTo to round AValue to a specified power of ten.

AValue is the value to round.

ADigit indicates the power of ten to which you want AValue rounded. It can be any value from –37 to 37 (inclusive).

RoundTo uses "Banker's Rounding" to determine how to round values that are exactly midway between the two values that have the desired number of significant digits. This method rounds to an even number in the case that AValue is not nearer to either value.

The following examples illustrate the use of RoundTo:

Expression	Value
RoundTo(1234567, 3)	1235000
RoundTo(1.234, -2)	1.23
RoundTo(1.235, -2)	1.24
RoundTo(1.245, -2)	1.24

Note: The behavior of RoundTo can be affected by the Set8087CW procedure or SetRoundMode (see page 4760) function.

See Also

Round

SetRoundMode (see page 4760)

Set8087CW

SimpleRoundTo (see page 4761)

46.2.82 Math.SameValue Function

Indicates whether two floating-point values are (approximately) equal.

Pascal

```
function SameValue(const A: Extended; const B: Extended; Epsilon: Extended = 0): Boolean;
overload;
function SameValue(const A: Double; const B: Double; Epsilon: Double = 0): Boolean; overload;
function SameValue(const A: Single; const B: Single; Epsilon: Single = 0): Boolean; overload;
```

C++

```
Boolean SameValue(const Extended A, const Extended B, Extended Epsilon = 0);
Boolean SameValue(const Double A, const Double B, Double Epsilon = 0);
Boolean SameValue(const Single A, const Single B, Single Epsilon = 0);
```

File

Math

Description

Call SameValue to determine when two floating-point values are equal, or approximately equal.

A and B are the values to compare.

Epsilon is the maximum amount by which A and B can differ and still be considered the same value.

See Also

CompareValue (see page 4726)

IsZero (see page 4742)

46.2.83 Math.Sec Function

Calculates the secant of an angle.

Pascal

```
function Sec(const X: Extended): Extended;
```

C++

```
Extended Sec(const Extended X);
```

File

Math

Description

Call Sec to obtain the secant of X, where X is an angle in radians. The secant is calculated using the formula 1 / Cos(X).

Note: Sec is identical to the Secant (see page 4758) function.

See Also

ArcSec (see page 4722)

Secant (see page 4758)

SecH (see page 4759)

Cos

46.2.84 Math.Secant Function

Calculates the secant of an angle.

Pascal

```
function Secant(const X: Extended): Extended;
```

C++

```
Extended Secant(const Extended X);
```

File

Math

Description

Call Secant to obtain the secant of X, where X is an angle in radians. The secant is calculated using the formula 1 / Cos(X).

See Also

Sec (see page 4758)

SecH (see page 4759)

Cos

46.2.85 Math.SecH Function

Calculates the hyperbolic secant of an angle.

Pascal

```
function SecH(const X: Extended): Extended;
```

C++

```
Extended SecH(const Extended X);
```

File

Math

Description

Call SecH to obtain the hyperbolic secant of X, where X is an angle in Radians.

See Also

Sec (see page 4758)

ArcSec (see page 4722)

46.2.86 Math.SetExceptionMask Function

Sets the exception mask on the FPU control word.

Pascal

```
function SetExceptionMask(const Mask: TFPUExceptionMask): TFPUExceptionMask;
```

C++

```
TFPUExceptionMask SetExceptionMask(const TFPUExceptionMask Mask);
```

File

Math

Description

Call SetExceptionMask to specify the types of exceptions the FPU (floating-point unit) generates. Any element set in this mask prevents the FPU from raising the corresponding type of exception. Instead, the FPU returns its best attempt at a value – often NaN (see page 4780), Infinity (see page 4776), or NegInfinity (see page 4781). The value depends on the operation and the current rounding mode.

SetExceptionMask returns the previous exception mask.

See Also

Set8087CW

GetExceptionMask (see page 4735)

NaN (see page 4780)

Infinity (see page 4776)

NegInfinity (see page 4781)

46.2.87 Math.SetPrecisionMode Function

Sets the FPU precision control mode.

Pascal

```
function SetPrecisionMode(const Precision: TFPUPrecisionMode): TFPUPrecisionMode;
```

C++

```
TFPUPrecisionMode SetPrecisionMode(const TFPUPrecisionMode Precision);
```

File

Math

Description

Call SetPrecisionMode to specify the level of precision that the FPU (floating-point unit) uses for floating-point calculations. The precision control mode can be any of the following values:

Value	Meaning
pmSingle	single precision
pmReserved	not used
pmDouble	double precision
pmExtended	extended precision

SetPrecisionMode returns the previous precision control mode.

See Also

Set8087CW

GetPrecisionMode (see page 4735)

46.2.88 Math.SetRoundMode Function

Sets the FPU rounding mode.

Pascal

```
function SetRoundMode(const RoundMode: TFPURoundingMode): TFPURoundingMode;
```

C++

```
TFPURoundingMode SetRoundMode(const TFPURoundingMode RoundMode);
```

File

Math

Description

Call SetRoundingMode to specify how the FPU handles rounding issues. The rounding mode can be any of the following values:

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

Set8087CW

GetRoundMode (see page 4736)

46.2.89 Math.Sign Function

Indicates whether a numeric value is positive, negative, or zero.

Pascal

```
function Sign(const AValue: Integer): TValueSign; overload;
function Sign(const AValue: Int64): TValueSign; overload;
function Sign(const AValue: Double): TValueSign; overload;
```

C++

```
TValueSign Sign(const int AValue);
TValueSign Sign(const Int64 AValue);
TValueSign Sign(const Double AValue);
```

File

Math

Description

Use Sign to test the sign of a numeric value. Sign returns

0 if AValue is zero.

1 if AValue is greater than zero.

-1 if AValue is less than zero.

See Also

IsZero (see page 4742)

46.2.90 Math.SimpleRoundTo Function

Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.

Pascal

```
function SimpleRoundTo(const AValue: Extended; const ADigit: TRoundToRange = -2): Extended;
```

C++

```
Extended SimpleRoundTo(const Extended AValue, const TRoundToRange ADigit = -2);
```

File

Math

Description

Call SimpleRoundTo to round AValue to a specified power of ten.

AValue is the value to round.

ADigit indicates the power of ten to which you want AValue rounded. It can be any value from –37 to 37 (inclusive).

SimpleRoundTo uses asymmetric arithmetic rounding to determine how to round values that are exactly midway between the two values that have the desired number of significant digits. This method always rounds to the larger value.

The following examples illustrate the use of SimpleRoundTo:

Expression	Value
SimpleRoundTo(1234567, 3)	1234000
SimpleRoundTo(1.234, -2)	1.23
SimpleRoundTo(1.235, -2)	1.24
SimpleRoundTo(-1.235, -2)	-1.23

See Also

Round

RoundTo (see page 4756)

46.2.91 Math.SinCos Function

Returns sine and cosine of an angle.

Pascal

```
procedure SinCos(const Theta: Extended; var Sin: Extended; var Cos: Extended);
```

C++

```
SinCos(const Extended Theta, Extended Sin, Extended Cos);
```

File

Math

Description

Call SinCos to obtain the sine and cosine of an angle expressed in radians. Theta specifies the angle. The sine and cosine are returned in the Sin and Cos parameters, respectively.

SinCos is twice as fast as calling Sin and Cos separately for the same angle.

46.2.92 Math.Sinh Function

Returns the hyperbolic sine of an angle.

Pascal

```
function Sinh(const X: Extended): Extended;
```

C++

```
Extended Sinh(const Extended X);
```

File

Math

Description

Sinh calculates the hyperbolic sine of X.

46.2.93 Math.SLNDepreciation Function

Returns the straight-line depreciation allowance of an asset.

Pascal

```
function SLNDepreciation(const Cost: Extended; const Salvage: Extended; Life: Integer):
Extended;
```

C++

```
Extended SLNDepreciation(const Extended Cost, const Extended Salvage, int Life);
```

File

Math

Description

SLNDepreciation calculates the straight-line depreciation allowance for an asset over one period of its life. The function divides the Cost minus the Salvage by the number of years of useful Life of the asset. Cost is the amount initially paid for the asset. Salvage is the value of the asset at the end of its useful life.

To compute accelerated depreciation (allowing higher depreciation values in the first years of the assets life), use the SYDDepreciation (see page 4766) function.

See Also

SYDDepreciation (see page 4766)

46.2.94 Math.StdDev Function

Returns the sample standard deviation for elements in an array.

Pascal

```
function StdDev(const Data: array of Double): Extended;
```

C++

```
Extended StdDev(const array of Double Data);
```

File

Math

Description

StdDev calculates the sample standard deviation (the square root of the sample variance) of all values in the Data array parameter. In C++, Data_Size indicates the index of the last element in the array (one less than the number of elements).

See Also

TotalVariance (see page 4767)

Variance (see page 4768)

46.2.95 Math.Sum Function

Returns the sum of the elements in an array.

Pascal

```
function Sum(const Data: array of Double): Extended;
```

C++

```
Extended Sum(const array of Double Data);
```

File

Math

Description

Sum returns the sum of all the values in the Data array parameter. In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

SumInt (see page 4764)

46.2.96 Math.SumInt Function

Returns the sum of the elements in an integer array.

Pascal

```
function SumInt(const Data: array of Integer): Integer;
```

C++

```
int SumInt(const array of Integer Data);
```

File

Math

Description

Sum (see page 4764) returns the sum of all the values in the Data array parameter. In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

Sum (see page 4764)

46.2.97 Math.SumOfSquares Function

Returns the sum of the squared values from a data array.

Pascal

```
function SumOfSquares(const Data: array of Double): Extended;
```

C++

```
Extended SumOfSquares(const array of Double Data);
```

File

Math

Description

SumOfSquares returns the sum of the squares of the Data values: Data[0]^2 + Data[1]^2 + Data[2]^2... In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

Sum (see page 4764)

SumsAndSquares (see page 4765)

46.2.98 Math.SumsAndSquares Function

Returns the sum of the values and the sum of the squared values in an array.

Pascal

```
procedure SumsAndSquares(const Data: array of Double; var Sum: Extended; var SumOfSquares:
Extended);
```

C++

```
SumsAndSquares(const array of Double Data, Extended Sum, Extended SumOfSquares);
```

File

Math

Description

SumsAndSquares returns the sum of the Data values in Sum (see page 4764) and the sum of the squares in SumOfSquares (see page 4765). The data to sum is specified in the Data array parameter. In C++, Data_Size indicates the index of the last value (one less than the number of values in Data).

This function is faster than calculating the sums separately.

See Also

Sum (see page 4764)

SumOfSquares (see page 4765)

TotalVariance (see page 4767)

46.2.99 Math.SYDDepreciation Function

Calculates depreciation for an asset.

Pascal

```
function SYDDepreciation(const Cost: Extended; const Salvage: Extended; Life: Integer; Period: Integer): Extended;
```

C++

```
Extended SYDDepreciation(const Extended Cost, const Extended Salvage, int Life, int Period);
```

File

Math

Description

SYDDepreciation (for "sum-of-years-digits depreciation") calculates depreciation amounts for an asset using an accelerated depreciation method. This allows for higher depreciation in the earlier years of an asset's life.

Cost is the initial cost of the asset. Salvage is the value of the asset at the end of its life expectancy. Life is the length of the asset's life expectancy. Period is the period for which to calculate the depreciation.

See Also

SLNDepreciation (see page 4763)

46.2.100 Math.Tan Function

Returns the tangent of X.

Pascal

```
function Tan(const X: Extended): Extended;
```

C++

```
Extended Tan(const Extended X);
```

File

Math

Description

Tan returns the tangent of X. Tan(X) = Sin(X) / Cos(X).

See Also

Sin

Cos

ArcTan

46.2.101 Math.Tanh Function

Returns the hyperbolic tangent of X.

Pascal

```
function Tanh(const X: Extended): Extended;
```

C++

```
Extended Tanh(const Extended X);
```

File

Math

Description

Tanh calculates the hyperbolic tangent of X.

See Also

Cosh (see page 4727)

Sinh (see page 4762)

46.2.102 Math.TotalVariance Function

Returns the statistical variance from an array of values.

Pascal

```
function TotalVariance(const Data: array of Double): Extended;
```

C++

```
Extended TotalVariance(const array of Double Data);
```

File

Math

Description

TotalVariance returns the sum of the squares of the difference between each Data value and the mean of the data values.

Total Variance (see page 4768): SUM(i=1,N)[(X(i) - Mean (see page 4746))**2]

See Also

SumOfSquares (see page 4765)

Variance (see page 4768)

StdDev (see page 4763)

Mean (see page 4746)

46.2.103 Math.Variance Function

Calculates statistical sample variance from an array of data.

Pascal

```
function Variance(const Data: array of Double): Extended;
```

C++

```
Extended Variance(const array of Double Data);
```

File

Math

Description

Variance calculates the sample variance of all values in the Data array parameter, using the n-1 method (unbiased). In C++, Data_Size specifies the index of the last value in Data (on less than the number of data values).

See Also

TotalVariance (see page 4767)

StdDev (see page 4763)

Mean (see page 4746)

PopnVariance (see page 4752)

46.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFPUException (see page 4769)	TFPUException defines TFPUExceptionMask (see page 4773) values.
	TFPUPrecisionMode (see page 4770)	TFPUPrecisionMode defines how the FPU handles precision.
	TFPURoundingMode (see page 4771)	TFPURoundingMode defines how the FPU handles rounding issues.
	TPaymentTime (see page 4772)	TPaymentTime is an enumerated type used as a parameter in financial functions.

46.3.1 Math.TFPUException Enumeration

TFPUException defines TFPUExceptionMask (see page 4773) values.

Pascal

```
TFPUException = (
  exInvalidOp,
  exDenormalized,
  exZeroDivide,
  exOverflow,
  exUnderflow,
  exPrecision
);
```

C++

```
enum TFPUException {
  exInvalidOp,
  exDenormalized,
  exZeroDivide,
  exOverflow,
  exUnderflow,
  exPrecision
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. You can prevent exceptions from occuring by using SetExceptionMask (see page 4759) to mask some or all of them.

When an exception is masked, the FPU returns its best attempt at a value – often NaN (see page 4780), Infinity (see page 4776), or NegInfinity (see page 4781). The value depends on the operation and the current rounding mode (see GetRoundMode (see page 4736) and SetRoundMode (see page 4760)).

TFPUExceptionMask (see page 4773) can have none, one, some or all of the following TFPUException values set :

Value	Meaning
exInvalidOp	An invalid operation was attempted.
exDenormalized	A number was reduced in size smaller than can be stored as non-zero. It has been denormalized.

exZeroDivide	An attempt was made to divide by zero.
exOverflow	A number has exceeded the highest positive value supported.
exUnderflow	A number has exceeded the highest negative value supported.
exPrecision	A number has exceeded the number of digits of precision.

See Also

GetExceptionMask (see page 4735)

GetRoundMode (see page 4736)

SetExceptionMask (see page 4759)

SetRoundMode (see page 4760)

TFPUExceptionMask (see page 4773)

46.3.2 Math.TFPUPrecisionMode Enumeration

TFPUPrecisionMode defines how the FPU handles precision.

Pascal

```
TFPUPrecisionMode = (
  pmSingle,
  pmReserved,
  pmDouble,
  pmExtended
);
```

C++

```
enum TFPUPrecisionMode {
  pmSingle,
  pmReserved,
  pmDouble,
  pmExtended
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. GetPrecisionMode (see page 4735) is used to read, and SetPrecisionMode (see page 4760) is used to set the precision control mode, with TFPUPrecisionMode defining this mode.

TFPUPrecisionMode can be one of the following values :

Value	Meaning
pmSingle	Single precision : 7 significant digits, exponent from -38 to +38.
pmReserved	Reserved for future use.
pmDouble	Double precision : 15 significant digits, exponent from -308 to +308.
pmExtended	Extended precision : 19 significant digits, exponent from -4932 to +4932.

See Also

GetPrecisionMode (see page 4735)

GetRoundMode (see page 4736)

SetPrecisionMode (see page 4760)

SetRoundMode (see page 4760)

46.3.3 Math.TFPURoundingMode Enumeration

TFPURoundingMode defines how the FPU handles rounding issues.

Pascal

```
TFPURoundingMode = (
  rmNearest,
  rmDown,
  rmUp,
  rmTruncate
);
```

C++

```
enum TFPURoundingMode {
  rmNearest,
  rmDown,
  rmUp,
  rmTruncate
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. GetRoundMode (see page 4736) is used to read, and SetRoundMode (see page 4760) is used to set the rounding mode, with TFPURoundingMode defining this mode.

TFPURoundingMode can be one of the following values :

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

GetPrecisionMode (see page 4735)

GetRoundMode (see page 4736)

SetPrecisionMode (see page 4760)

SetRoundMode (see page 4760)

46.3.4 Math.TPaymentTime Enumeration

TPaymentTime is an enumerated type used as a parameter in financial functions.

Pascal

```
TPaymentTime = (
  ptEndOfPeriod,
  ptStartOfPeriod
);
```

C++

```
enum TPaymentTime {
  ptEndOfPeriod,
  ptStartOfPeriod
};
```

File

Math

Description

TPaymentTime indicates when interest is calculated. The following table lists the possible values:

Value	Meaning
ptStartOfPeriod	Interest is calculated on the principle balance at the beginning of the payment period
ptEndOfPeriod	Interest is calculated at the end of the payment period

46.4 Types

The following table lists types in this documentation.

Types

Name	Description
TFPUExceptionMask (see page 4773)	TFPUExceptionMask defines a set of TFPUException (see page 4769) values.
TRoundToEXRangeExtended (see page 4774)	This is type Math.TRoundToEXRangeExtended.
TRoundToRange (see page 4774)	TRoundToRange defines the RoundTo (see page 4756) rounding limits.
TValueSign (see page 4774)	TValueSign defines the Sign (see page 4761) returned value.

46.4.1 Math.TFPUExceptionMask Type

TFPUExceptionMask defines a set of TFPUException (see page 4769) values.

Pascal

```
TFPUExceptionMask = set of TFPUException;
```

C++

```
TFPUException TFPUExceptionMask;
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. You can prevent exceptions from occuring by using SetExceptionMask (see page 4759) to mask some or all of them.

When an exception is masked, the FPU returns its best attempt at a value – often NaN (see page 4780), Infinity (see page 4776), or NegInfinity (see page 4781). The value depends on the operation and the current rounding mode (see GetRoundMode (see page 4736) and SetRoundMode (see page 4760)).

TFPUExceptionMask can have none, one, some or all of the following TFPUException (see page 4769) values set :

Value	Meaning
exInvalidOp	An invalid operation was attempted.
exDenormalized	A number was reduced in size smaller than can be stored as non-zero. It has been denormalized.
exZeroDivide	An attempt was made to divide by zero.
exOverflow	A number has exceeded the highest positive value supported.
exUnderflow	A number has exceeded the highest negative value supported.
exPrecision	A number has exceeded the number of digits of precision.

See Also

GetExceptionMask (see page 4735)

GetRoundMode (see page 4736)

SetExceptionMask (see page 4759)

SetRoundMode (see page 4760)

TFPUException (see page 4769)

46.4.2 Math.TRoundToEXRangeExtended Type

Pascal

```
TRoundToEXRangeExtended = -20..20;
```

C++

```
-20..20 TRoundToEXRangeExtended;
```

File

Math

Description

This is type Math.TRoundToEXRangeExtended.

46.4.3 Math.TRoundToRange Type

TRoundToRange defines the RoundTo (see page 4756) rounding limits.

Pascal

```
TRoundToRange = -37..37;
```

C++

```
-37..37 TRoundToRange;
```

File

Math

Description

The RoundTo (see page 4756) routine rounds a floating-point value to a specified digit or power of ten using "Banker's rounding". A negative TRoundToRange value rounds to this number of decimal places. A positive value removes all decimal places, and zeroes this number of significant digits to the left of the decimal point. Banker's rounding always rounds down to an even value last digit when it has a choice.

See Also

RoundTo (see page 4756)

46.4.4 Math.TValueSign Type

TValueSign defines the Sign (see page 4761) returned value.

Pascal

```
TValueSign = -1..1;
```

C++

```
-1..1 TValueSign;
```

File

Math

Description

The Sign (see page 4761) routine determines whether a number is negative, zero or positive. It returns the result in a TValueSign value as follows :

Value	Associated constant	Meaning
-1	NegativeValue (see page 4780)	Negative number.
0	ZeroValue (see page 4782)	Zero.
+1	PositiveValue (see page 4781)	Positive number.

See Also

NegativeValue (see page 4780)

PositiveValue (see page 4781)

ZeroValue (see page 4782)

RoundTo (see page 4756)

46.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Infinity (see page 4776)	Infinity represents positive infinity.
MaxComp (see page 4777)	MaxComp defines the maximum absolute value for a Comp variable.
MaxDouble (see page 4777)	MaxDouble defines the maximum absolute value for a Double variable.
MaxExtended (see page 4777)	MaxExtended defines the maximum absolute value for an Extended variable.
MaxSingle (see page 4778)	MaxSingle defines the maximum absolute value for a Single variable.
MinComp (see page 4778)	MinComp defines the smallest absolute value for a Comp variable.
MinDouble (see page 4779)	MinDouble defines the smallest absolute value for a Double variable.
MinExtended (see page 4779)	MinExtended defines the smallest absolute value for an Extended variable.
MinSingle (see page 4779)	MinSingle defines the smallest absolute value for a Single variable.
NaN (see page 4780)	NaN represents a value that cannot be represented as a valid number.
NegativeValue (see page 4780)	NegativeValue defines a negative number result from using Sign (see page 4761).
NegInfinity (see page 4781)	NegInfinity represents negative infinity.
PositiveValue (see page 4781)	PositiveValue defines a positive number result from using Sign (see page 4761).
ZeroValue (see page 4782)	ZeroValue defines a zero value number result from using Sign (see page 4761).

46.5.1 Math.Infinity Constant

Infinity represents positive infinity.

Pascal

```
Infinity = 1.0 / 0.0;
```

C++

```
Infinity = 1.0 / 0.0;
```

File

Math

Description

Infinity is literally defined as 1.0/0.0 : positive infinity.

Note: Only use Infinity in assignments : use IsInfinite (see page 4741).

See Also

IsInfinite (see page 4741)

NegInfinity (see page 4781)

46.5.2 Math.MaxComp Constant

MaxComp defines the maximum absolute value for a Comp variable.

Pascal

```
MaxComp = 9.223372036854775807e+18;
```

C++

```
MaxComp = 9.223372036854775807e+18;
```

File

Math

Description

MaxComp defines the maximum absolute value for a Comp variable.

See Also

MinComp (see page 4778)

46.5.3 Math.MaxDouble Constant

MaxDouble defines the maximum absolute value for a Double variable.

Pascal

```
MaxDouble = 1.7e+308;
```

C++

```
MaxDouble = 1.7e+308;
```

File

Math

Description

MaxDouble defines the maximum absolute value for a Double variable.

See Also

MinDouble (see page 4779)

46.5.4 Math.MaxExtended Constant

MaxExtended defines the maximum absolute value for an Extended variable.

Pascal

```
MaxExtended = 1.1e+4932;
```

C++

```
MaxExtended = 1.1e+4932;
```

File

Math

Description

MaxExtended defines the maximum absolute value for an Extended variable.

See Also

MinExtended (see page 4779)

46.5.5 Math.MaxSingle Constant

MaxSingle defines the maximum absolute value for a Single variable.

Pascal

```
MaxSingle = 3.4e+38;
```

C++

```
MaxSingle = 3.4e+38;
```

File

Math

Description

MaxSingle defines the maximum absolute value for a Single variable.

See Also

MinSingle (see page 4779)

46.5.6 Math.MinComp Constant

MinComp defines the smallest absolute value for a Comp variable.

Pascal

```
MinComp = -9.223372036854775807e+18;
```

C++

```
MinComp = -9.223372036854775807e+18;
```

File

Math

Description

MinComp defines the smallest absolute value for a Comp variable.

See Also

MaxComp (see page 4777)

46.5.7 Math.MinDouble Constant

MinDouble defines the smallest absolute value for a Double variable.

Pascal

```
MinDouble = 5.0e-324;
```

C++

```
MinDouble = 5.0e-324;
```

File

Math

Description

MinDouble defines the smallest absolute value for a Double variable.

See Also

MaxDouble (see page 4777)

46.5.8 Math.MinExtended Constant

MinExtended defines the smallest absolute value for an Extended variable.

Pascal

```
MinExtended = 3.4e-4932;
```

C++

```
MinExtended = 3.4e-4932;
```

File

Math

Description

MinExtended defines the smallest absolute value for an Extended variable.

See Also

MaxExtended (see page 4777)

46.5.9 Math.MinSingle Constant

MinSingle defines the smallest absolute value for a Single variable.

Pascal

```
MinSingle = 1.5e-45;
```

C++

```
MinSingle = 1.5e-45;
```

File

Math

Description

MinSingle defines the smallest absolute value for a Single variable.

See Also

MaxSingle (see page 4778)

46.5.10 Math.NaN Constant

NaN represents a value that cannot be represented as a valid number.

Pascal

```
NaN = 0.0 / 0.0;
```

C++

```
NaN = 0.0 / 0.0;
```

File

Math

Description

NaN is literally defined as 0.0/0.0 : an undefined value.

Note: Only use NaN in assignments : use IsNaN.

See Also

IsNaN

46.5.11 Math.NegativeValue Constant

NegativeValue defines a negative number result from using Sign (see page 4761).

Pascal

```
NegativeValue = Low(TValueSign);
```

C++

```
NegativeValue = Low(TValueSign);
```

File

Math

Description

Sign (see page 4761) determines the sign of a number, returning the result in a TValueSign (see page 4774) value. TValueSign (see page 4774) will be NegativeValue if the number is negative.

See Also

PositiveValue (see page 4781)

Sign (see page 4761)

TValueSign (see page 4774)

ZeroValue (see page 4782)

46.5.12 Math.NegInfinity Constant

NegInfinity represents negative infinity.

Pascal

```
NegInfinity = -1.0 / 0.0;
```

C++

```
NegInfinity = -1.0 / 0.0;
```

File

Math

Description

NegInfinity is literally defined as -1.0/0.0 : negative infinity.

Note: Only use NegInfinity in assignments : use IsInfinite (see page 4741).

See Also

Infinity (see page 4776)

IsInfinity

IsInfinite (see page 4741)

46.5.13 Math.PositiveValue Constant

PositiveValue defines a positive number result from using Sign (see page 4761).

Pascal

```
PositiveValue = High(TValueSign);
```

C++

```
PositiveValue = High(TValueSign);
```

File

Math

Description

Sign (see page 4761) determines the sign of a number, returning the result in a TValueSign (see page 4774) value. TValueSign (see page 4774) will be PositiveValue if the number is positive.

See Also

NegativeValue (see page 4780)

Sign (see page 4761)

TValueSign (see page 4774)

ZeroValue (see page 4782)

46.5.14 Math.ZeroValue Constant

ZeroValue defines a zero value number result from using Sign (see page 4761).

Pascal

```
ZeroValue = 0;
```

C++

```
ZeroValue = 0;
```

File

Math

Description

Sign (see page 4761) determines the sign of a number, returning the result in a TValueSign (see page 4774) value. TValueSign (see page 4774) will be ZeroValue if the number is zero.

See Also

NegativeValue (see page 4780)

PositiveValue (see page 4781)

Sign (see page 4761)

TValueSign (see page 4774)

47 MConnect Namespace

This is namespace MConnect.

Topics

Name	Description
Classes (see page 4784)	The following table lists classes in this documentation.
Functions (see page 4823)	The following table lists functions in this documentation.

Classes

	Name	Description
	TCOMConnection (see page 4784)	TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.
	TCustomObjectBroker (see page 4787)	TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.
	TDCOMConnection (see page 4790)	TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.
	TDispatchAppServer (see page 4794)	This is class MConnect.TDispatchAppServer.
	TDispatchConnection (see page 4796)	TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.
	TOLEnterpriseConnection (see page 4811)	TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.
	TSharedConnection (see page 4814)	TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

Functions

	Name	Description
	GetMIDASAppServerList (see page 4823)	This is function MConnect.GetMIDASAppServerList.

47.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCOMConnection (see page 4784)	TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.
	TCustomObjectBroker (see page 4787)	TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.
	TDCOMConnection (see page 4790)	TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.
	TDispatchAppServer (see page 4794)	This is class MConnect.TDispatchAppServer.
	TDispatchConnection (see page 4796)	TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.
	TOLEnterpriseConnection (see page 4811)	TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.
	TSharedConnection (see page 4814)	TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

47.1.1 TCOMConnection Class

TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.

Class Hierarchy

Pascal

```
TCOMConnection = class(TDispatchConnection);
```

C++

```
class TCOMConnection : public TDispatchConnection;
```

File

MConnect

Description

Do not create instances of TCOMConnection. Instead, use TCOMConnection as a base class when created a custom component that forms a COM-based connection to an application server.

TCOMConnection adds to the ability to work with COM objects to the generic connection capabilities inherited from TDispatchConnection (see page 4796). It has two descendants: TDCOMConnection (see page 4790), which creates and manages connections using DCOM, and TOLEnterpriseConnection (see page 4811), which is provided for backward compatibility and handles connections using OLEnterprise.

See Also

TClientDataSet

TDCOMConnection (see page 4790)

TOLEnterpriseConnection (see page 4811)

TDataSetProvider (see page 6510)

IAppServer (see page 5222)

Writing Applications Using COM

47.1.1.1 TCOMConnection Members

The following tables list the members exposed by TCOMConnection.

Protected Methods

	Name	Description
	DoConnect (see page 4786)	Connects to an application server.
	DoDisconnect (see page 4798)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.
	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4786)	Sets the Connected (see page 4802) property.

Protected Properties

	Name	Description
	ObjectBroker (see page 4806)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Public Methods

	Name	Description
	Create (see page 4797)	Creates an instance of a dispatch connection component.
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

Published Properties

	Name	Description
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.

47.1.1.2 TCOMConnection Methods

The methods of the TCOMConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 4786)	Connects to an application server.
	SetConnected (see page 4786)	Sets the Connected (see page 4802) property.

47.1.1.2.1 TCOMConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected (see page 4802) property is set to true. DoConnect creates a COM object for the application server (based on the ServerName (see page 4807) or ServerGUID (see page 4807) property), and sets the value of the AppServer property to this COM object.

Descendants of TCOMConnection (see page 4784) override DoConnect to change the way the client application forms a connection to an application server.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

ServerName (see page 4807)

ServerGUID (see page 4807)

47.1.1.2.2 TCOMConnection.SetConnected Method

Sets the Connected (see page 4802) property.

Pascal

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

C++

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

Description

SetConnected is the protected implementation of the Connected (see page 4802) property.

If Value is false, SetConnected generates a BeforeDisconnect (see page 4809) event, closes the connection, and generates an AfterDisconnect (see page 4808) event.

If Value is true, SetConnected performs the following actions:

1. It checks the value of the ServerGUID (see page 4807) or ServerName (see page 4807) property and raises an exception if these do not specify a valid application server.

2. If LoginPrompt (see page 4802) is true, and the connection is not already open, SetConnected generates an OnGetUsername (see page 4810) event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
3. It generates a BeforeConnect (see page 4809) event.
4. It opens the connection.
5. It generates an AfterConnect (see page 4808) event.
6. If LoginPrompt (see page 4802) is true, and the connection opened successfully, SetConnected generates an OnLogin (see page 4810) event.

See Also

Connected

ServerGUID (see page 4807)

ServerName (see page 4807)

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

OnGetUserName

OnLogin

47.1.2 TCustomObjectBroker Class

TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.

Class Hierarchy

Pascal

```
TCustomObjectBroker = class(TComponent);
```

C++

```
class TCustomObjectBroker : public TComponent;
```

File

MConnect

Description

TCustomObjectBroker defines the methods by which a connection component obtains information about available application servers. When a connection component uses an object broker, it does not save any properties that identify a particular server machine. Instead, when it needs to open a connection, it uses the methods of the object broker to get that identifying information. This process allows client applications to locate server systems dynamically at runtime. If there are redundant server systems, this allows client applications to continue functioning when a server goes down.

TCustomObjectBroker does not implement any of the methods that provide information to connection components. Instead,

descendants such as TSimpleObjectBroker implement a mechanism for choosing servers and identifying them to connection components. Application developers can derive more elaborate object brokers that provide additional benefits such as intelligent load balancing.

See Also

TDispatchConnection (see page 4796)

TSimpleObjectBroker (see page 5916)

47.1.2.1 TCustomObjectBroker Members

The following tables list the members exposed by TCustomObjectBroker.

Public Methods

	Name	Description
	GetComputerForGUID (see page 4788)	Returns a string that identifies a server machine for an interface identified by its GUID.
	GetComputerForProgID (see page 4789)	Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.
	GetPortForComputer (see page 4789)	Returns a port number for connecting to a specified remote server.
	SetConnectStatus (see page 4790)	Records whether a connection was established to a specified server.

47.1.2.2 TCustomObjectBroker Methods

The methods of the TCustomObjectBroker class are listed here.

Public Methods

	Name	Description
	GetComputerForGUID (see page 4788)	Returns a string that identifies a server machine for an interface identified by its GUID.
	GetComputerForProgID (see page 4789)	Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.
	GetPortForComputer (see page 4789)	Returns a port number for connecting to a specified remote server.
	SetConnectStatus (see page 4790)	Records whether a connection was established to a specified server.

47.1.2.2.1 TCustomObjectBroker.GetComputerForGUID Method

Returns a string that identifies a server machine for an interface identified by its GUID.

Pascal

```
function GetComputerForGUID(GUID: TGUID): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetComputerForGUID(TGUID GUID) = 0;
```

Description

Call GetComputerForGUID to obtain a string that identifies a server machine on which a given interface is registered. The GUID parameter specifies the global unique identifier of the desired interface.

TCustomObjectBroker (see page 4787) declares GetComputerForGUID as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Connection components call this method, passing in the value of their ServerGUID property. They use the return value to set the ComputerName, URL, Host, or Address property, as appropriate.

See Also

GetComputerForProgID (see page 4789)

ServerGUID (see page 4807)

computername (see page 4793)

Address

Host

URL

47.1.2.2.2 TCustomObjectBroker.GetComputerForProgID Method

Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.

Pascal

```
function GetComputerForProgID(const ProgID): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetComputerForProgID(const  ProgID) = 0;
```

Description

Call GetComputerForProgID to obtain the name or address of a server machine on which a given interface is registered. The ProgID parameter specifies the name of the desired interface. If the server was created using one of the remote data module wizards, this is the value that was entered as the class name for the wizard.

TCustomObjectBroker (see page 4787) declares GetComputerForProgID as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Connection components can call this method, passing in the value of their ServerName property. They can use the return value to set the ComputerName, URL, Host, or Address property, as appropriate.

See Also

GetComputerForGUID (see page 4788)

ServerName (see page 4807)

computername (see page 4793)

Address

Host

URL

47.1.2.2.3 TCustomObjectBroker.GetPortForComputer Method

Returns a port number for connecting to a specified remote server.

Pascal

```
function GetPortForComputer(const ComputerName: string): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetPortForComputer(const UnicodeString ComputerName) = 0;
```

Description

Socket connection components call GetPortForComputer to get a value for their Port property. The ComputerName parameter specifies a Host name or IP address that was returned by GetComputerForGUID (see page 4788) or GetComputerForProgID (see page 4789).

None of the connection components that work on Linux explicitly use a Port number. GetPortForComputer is included in TCustomObjectBroker (see page 4787) to allow for the possibility of connection components that use an explicit Port number. The ComputerName parameter is a Host name or IP address that is returned by the GetComputerForGUID (see page 4788) or GetComputerForProgID (see page 4789) method.

TCustomObjectBroker (see page 4787) declares GetPortForComputer as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Note: This interface assumes that a single socket dispatcher on each server system handles all client requests, even when they are destined for different application servers on the same host.

See Also

GetComputerForProgID (see page 4789)

GetComputerForGUID (see page 4788)

Port

47.1.2.2.4 TCustomObjectBroker.SetConnectStatus Method

Records whether a connection was established to a specified server.

Pascal

```
procedure SetConnectStatus(ComputerName: string; Success: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SetConnectStatus(UnicodeString ComputerName, Boolean Success) = 0;
```

Description

After calling GetComputerForGUID (see page 4788) or GetComputerForProgID (see page 4789), connection components use the returned value to open a connection to the identified server. They then call SetConnectStatus to tell the broker whether the connection was successfully opened.

TCustomObjectBroker (see page 4787) declares SetConnectStatus as an abstract (pure virtual) method. Descendants override this method to keep track of when server machines are down or to monitor the number of connections created for each server.

See Also

GetComputerForGUID (see page 4788)

GetComputerForProgID (see page 4789)

47.1.3 TDCOMConnection Class

TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.

Class Hierarchy

Pascal

```
TDCOMConnection = class(TCOMConnection);
```

C++

```
class TDCOMConnection : public TCOMConnection;
```

File

MConnect

Description

TDCOMConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The DCOM connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer (see page 5222) for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TDCOMConnection uses DCOM to establish the initial connection between the client application and a remote application server. In order to do so, the application server must be registered on the client machine.

Once the connection is established, the client registers any or all of its client datasets with TDCOMConnection, and these client datasets use the IAppServer (see page 5222) interface from the DCOM connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect client applications to remote application servers using TCP/IP sockets, use TSocketConnection instead. To connect to remote application servers using HTTP, use the TWebConnection component. To connect to application servers using SOAP, use TSoapConnection.

See Also

TClientDataSet

TSocketConnection

TRemoteDataModule

TDataSetProvider (see page 6510)

IAppServer (see page 5222)

TWebConnection

TSoapConnection

Specifying a Connection Using DCOM

47.1.3.1 TDCOMConnection Members

The following tables list the members exposed by TDCOMConnection.

Protected Methods

	Name	Description
	DoConnect (see page 4793)	Connects to an application server.
	DoDisconnect (see page 4798)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.

	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4786)	Sets the Connected (see page 4802) property.
	SetConnected (see page 4800)	Sets the Connected (see page 4802) property.

Public Methods

	Name	Description
	Create (see page 4792)	Creates and initializes a DCOM connection component.
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

Published Properties

	Name	Description
	ComputerName (see page 4793)	This is ComputerName, a member of class TDCOMConnection.
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ObjectBroker (see page 4794)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.

47.1.3.2 TDCOMConnection.Create Constructor

Creates and initializes a DCOM connection component.

Pascal

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

C++

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

Description

Call Create to instantiate TDCOMConnection (see page 4790) at runtime. DCOM connection components placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually the form or data module) that is responsible for freeing the DCOM connection component.

47.1.3.3 TDCOMConnection Methods

The methods of the TDCOMConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 4793)	Connects to an application server.

47.1.3.3.1 TDCOMConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected (see page 4802) property is set to true. DoConnect establishes a dual interface with the remote data module on the application server. This interface is used to locate providers on the application server.

To locate the server system, DoConnect uses the ComputerName (see page 4793) property. If the ObjectBroker (see page 4794) property is set, DoConnect gets this value from the object broker. Otherwise, the application must assign this property a value directly.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

computername (see page 4793)

ObjectBroker (see page 4806)

47.1.3.4 TDCOMConnection Properties

The properties of the TDCOMConnection class are listed here.

Published Properties

	Name	Description
	ComputerName (see page 4793)	This is ComputerName, a member of class TDCOMConnection.
	ObjectBroker (see page 4794)	Specifies an Object Broker that allows the connection component to choose from several application servers.

47.1.3.4.1 TDCOMConnection.ComputerName Property

Pascal

```
property ComputerName: string;
```

C++

```
__property UnicodeString ComputerName;
```

Description

This is ComputerName, a member of class TDCOMConnection.

47.1.3.4.2 TDCOMConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect

ServerName (see page 4807)

ServerGUID (see page 4807)

TSimpleObjectBroker (see page 5916)

Brokering Connections

Specifying a Connection Using HTTP

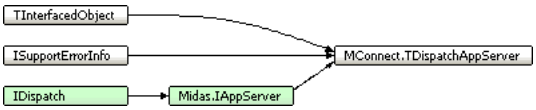
Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

47.1.4 TDispatchAppServer Class

Class Hierarchy



Pascal

```
TDispatchAppServer = class(TInterfacedObject, IAppServer, ISupportErrorInfo);
```

C++

```
class TDispatchAppServer : public TInterfacedObject, public IAppServer, public
ISupportErrorInfo;
```

File

MConnect

Description

This is class MConnect.TDispatchAppServer.

47.1.4.1 TDispatchAppServer Members

The following tables list the members exposed by TDispatchAppServer.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 5223)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 5224)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 5224)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 5225)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 5226)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 5227)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 5228)	Returns information from a specified record of the provider's dataset.
	Create (see page 4795)	This is Create, a member of class TDispatchAppServer.
V	SafeCallException (see page 4795)	Handles exceptions in methods declared using the safecall calling convention.

47.1.4.2 TDispatchAppServer.Create Constructor

Pascal

```
constructor Create(const AppServer: IAppServerDisp);
```

C++

```
__fastcall TDispatchAppServer(const IAppServerDisp AppServer);
```

Description

This is Create, a member of class TDispatchAppServer.

47.1.4.3 TDispatchAppServer Methods

The methods of the TDispatchAppServer class are listed here.

Public Methods

	Name	Description
V	SafeCallException (see page 4795)	Handles exceptions in methods declared using the safecall calling convention.

47.1.4.3.1 TDispatchAppServer.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

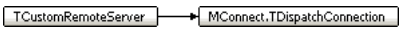
System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject, System::TObject::SafeCallException simply returns E_UNEXPECTED. This is the appropriate response for classes that do no support interfaces

47.1.5 TDispatchConnection Class

TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.

Class Hierarchy



Pascal

```
TDispatchConnection = class(TCustomRemoteServer);
```

C++

```
class TDispatchConnection : public TCustomRemoteServer;
```

File

MConnect

Description

Do not create instances of TDispatchConnection. Instead, use TDispatchConnection as a base class when creating a custom component that forms a connection to a COM-based application server in a multi-tiered database application.

TDispatchConnection introduces the infrastructure for logging on to an application server, obtaining an IAppServer (see page 5222) interface, calling the application server's interface, and managing a remote connection. Additionally, TCustomRemoteServer inherits the ability to work with one or more client datasets.

TDispatchConnection is not capable of connecting to an application server. This ability is added by descendants, which are specialized for different connection types. Once a connection is formed, however, TDispatchConnection makes the application server's IAppServer (see page 5222) interface available to TClientDataSet components.

To enable a client application to connect to a COM-based application server, use a descendant of TDispatchConnection such as TSocketConnection, TWebConnection, or TDCOMConnection (see page 4790).

See Also

TDataSetProvider (see page 6510)

TClientDataSet

TDCOMConnection (see page 4790)

TSocketConnection

IAppServer (see page 5222)

TWebConnection

TSOAPConnection

47.1.5.1 TDispatchConnection Members

The following tables list the members exposed by TDispatchConnection.

Protected Methods

	Name	Description
	DoDisconnect (see page 4798)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.
	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4800)	Sets the Connected (see page 4802) property.

Protected Properties

	Name	Description
	ObjectBroker (see page 4806)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Public Methods

	Name	Description
	Create (see page 4797)	Creates an instance of a dispatch connection component.
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

Published Properties

	Name	Description
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.

47.1.5.2 TDispatchConnection.Create Constructor

Creates an instance of a dispatch connection component.

Pascal

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

C++

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

Description

Call Create to instantiate a dispatch connection component at runtime. Connection components placed in forms or data modules

at design time are created automatically.

After calling the inherited constructor, Create initializes the LoginPrompt (see page 4802) property to false.

See Also

LoginPrompt

Create

47.1.5.3 TDispatchConnection Methods

The methods of the TDispatchConnection class are listed here.

Protected Methods

	Name	Description
	DoDisconnect (see page 4798)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.
	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4800)	Sets the Connected (see page 4802) property.

Public Methods

	Name	Description
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

47.1.5.3.1 TDispatchConnection.DoDisconnect Method

Drops a connection to an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected (see page 4802) property is set to false. It drops the connection and changes the AppServer property to a NULL Variant.

See Also

AfterDisconnect

BeforeDisconnect

Connected (see page 4802)

AppServer

47.1.5.3.2 TDispatchConnection.GetConnected Method

Returns the value of the Connected (see page 4802) property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected implementation of the Connected (see page 4802) property. It uses the AppServer property (set by the DoConnect method of a TDispatchConnection (see page 4796) descendant) to determine whether a connection is established.

If a connection is established, GetConnected returns true. Otherwise, GetConnected returns false.

See Also

Connected (see page 4802)

DoConnect (see page 4786)

47.1.5.3.3 TDispatchConnection.GetServer Method

Returns the IAppServer (see page 5222) interface of the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Client datasets call GetServer to set the value of their AppServer property. This allows them to access providers on the remote application server.

GetServer ensures that there is a connection and then returns an IAppServer (see page 5222) interface that can be used to communicate with providers on the application server. To use other methods of the application server's interface, use the AppServer property instead.

See Also

Connected (see page 4802)

AppServer

AppServer

47.1.5.3.4 TDispatchConnection.GetServerCLSID Method

Returns the GUID for the application server.

Pascal

```
function GetServerCLSID: TGUID;
```

C++

```
__fastcall TGUID GetServerCLSID();
```

Description

GetServerCLSID is used internally to obtain the GUID for the application server.

In most cases, GetServerCLSID returns the value of the ServerGUID (see page 4807) property. However, if the application server was specified using the ServerName (see page 4807) property, and the application server was not registered when the

ServerName (see page 4807) property was set, ServerGUID (see page 4807) will not represent the application server's GUID. GetServerCLSID can return a GUID that only becomes available when a connection is opened.

If neither the ServerName (see page 4807) nor the ServerGUID (see page 4807) property is set, GetServerCLSID raises an exception.

See Also

ServerGUID (see page 4807)

ServerName (see page 4807)

47.1.5.3.5 TDispatchConnection.GetServerList Method

Returns a list of remote data modules to which the component can connect.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName (see page 4807) property. GetServerList returns a variant array that lists all application servers that are on the local machine, registered in the system registry, or on the remote machine identified by the connection component.

See Also

ServerName (see page 4807)

47.1.5.3.6 TDispatchConnection.Notification Method

Responds when other objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically so objects can make internal adjustments when components are created or destroyed. TDispatchConnection (see page 4796) overrides the inherited method to set the ObjectBroker (see page 4806) property to nil (Delphi) or NULL (C++) when it receives notification that that object is about to be destroyed.

See Also

ObjectBroker (see page 4806)

47.1.5.3.7 TDispatchConnection.SetConnected Method

Sets the Connected (see page 4802) property.

Pascal

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

C++

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

Description

SetConnected is the protected implementation of the Connected (see page 4802) property. It establishes a connection to the Application server if Value is true, and disconnects from the application server if Value is false.

If Value is false, SetConnected generates a BeforeDisconnect (see page 4809) event, closes the connection, and generates an AfterDisconnect (see page 4808) event.

If Value is true, SetConnected performs the following actions:

1. If LoginPrompt (see page 4802) is true, and the connection is not already open, SetConnected generates an OnGetUsername (see page 4810) event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
2. It generates a BeforeConnect (see page 4809) event.
3. It opens the connection.
4. It generates an AfterConnect (see page 4808) event.
5. If LoginPrompt (see page 4802) is true, and the connection opened successfully, SetConnected generates an OnLogin (see page 4810) event.

See Also

Connected (see page 4802)

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

OnGetUserName

OnLogin

47.1.5.4 TDispatchConnection Properties

The properties of the TDispatchConnection class are listed here.

Protected Properties

	Name	Description
	ObjectBroker (see page 4806)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Published Properties

	Name	Description
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.

47.1.5.4.1 TDispatchConnection.Connected Property

Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to true to establish a connection to the application server. Set Connected to false to terminate the connection.

When changing the value of Connected, TDispatchConnection (see page 4796) generates a BeforeConnect (see page 4809) and an AfterConnect (see page 4808) event, or a BeforeDisconnect (see page 4809) and an AfterDisconnect (see page 4808) event. In addition, when setting Connected to true, TDispatchConnection (see page 4796) may display a login dialog, depending on the value of LoginPrompt (see page 4802).

The Connected property uses the protected DoConnect and DoDisconnect (see page 4798) methods to connect to or disconnect from an application server. Descendants of TDispatchConnection (see page 4796) override DoConnect and DoDisconnect (see page 4798) to change the way the connection is formed or terminated.

See Also

DoDisconnect (see page 4798)

LoginPrompt

AfterConnect

AfterDisconnect

BeforeConnect

BeforeDisconnect

DoConnect (see page 4786)

47.1.5.4.2 TDispatchConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt;
```

C++

```
__property LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the

connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

C++ Examples:

```
//
//This example demostrates the use of ADO for database conectivity.
//Example assumes that a TDBGrid is placed on the form.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Login details */
    String UserName = "db_user_name";
    String PassWord = "db_pass_word";
    String Server = "my.db.server";

    /* Connection String */
    String ConnString =
        "Provider=SQLOLEDB.1;Persist Security Info=False;";
    ConnString +=
        "User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;";
    ConnString +=
        "Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;";
    ConnString +=
        "Tag with column collation when possible=False";

    /* SQL Query */
    String SQLStr = "SELECT * FROM customer WHERE customer_id = :AnId;";

    /* All ADO variables */
    TADOConnection* ADOConn;
    TADOQuery* ADOQuery;
    TDataSource* DataSrc;
    TParameter* Param;

    /* Create an ADO connection */
    ADOConn = new TADOConnection(this);
```

```
    /* Setup the connection string */
    ADOConn->ConnectionString = Format(ConnString,
        ARRAYOFCONST((UserName, PassWord, Server)));

    /* Disable login prompt */
    ADOConn->LoginPrompt = False;

    try
    {
        ADOConn->Connected = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                    TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the query */
    ADOQuery = new TADOQuery(this);
    ADOQuery->Connection = ADOConn;
    ADOQuery->SQL->Add(SQLStr);

    /* Update the parameter that was parsed from the SQL query: AnId */
    Param = ADOQuery->Parameters->ParamByName("AnId");
    Param->DataType = ftInteger;
    Param->Value = 1;

    /* Set the query to Prepared - will improve performance */
    ADOQuery->Prepared = true;

    try
    {
        ADOQuery->Active = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                    TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the data source */
    DataSrc = new TDataSource(this);
    DataSrc->DataSet = ADOQuery;
    DataSrc->Enabled = true;

    /* Finally initilalize the grid */
    DBGrid1->DataSource = DataSrc;
}
```

Delphi Examples:

```
{
This example demostrates the use of ADO for database conectivity.
Example assumes that a TDBGrid is placed on the form.
}
procedure TForm2.FormCreate(Sender: TObject);
const
  { Connection string }
  ConnString =
  'Provider=SQLOLEDB.1;Persist Security Info=False;' +
  'User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;' +
  'Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;'+
  'Tag with column collation when possible=False';
```

```
    { SQL Query }
    SQLStr = 'SELECT * FROM customer WHERE customer_id = :AnId;';

    { User access }
    UserName = 'db_user_name';
    PassWord = 'db_pass_word';
    Server = 'my.db.server';

  var
    ADOConn  : TADOConnection;
    ADOQuery : TADOQuery;
    DataSrc  : TDataSource;
    Param    : TParameter;

  begin
    { Create an ADO connection }
    ADOConn := TADOConnection.Create(Self);
    { Setup the provider engine }

    { Setup the connection string }
    ADOConn.ConnectionString := Format(ConnString,
      [UserName, PassWord, Server]);

    { Disable login prompt }
    ADOConn.LoginPrompt := False;

    try
      ADOConn.Connected := True;
    except
      on e: EADOError do
      begin
        MessageDlg('Error while connecting', mtError,
                   [mbOK], 0);

        Exit;
      end;
    end;

    { Create the query }
    ADOQuery := TADOQuery.Create(Self);
    ADOQuery.Connection := ADOConn;
    ADOQuery.SQL.Add(SQLStr);

    { Update the parameter that was parsed from the SQL query: AnId }
    Param := ADOQuery.Parameters.ParamByName('AnId');
    Param.DataType := ftInteger;
    Param.Value := 1;

    { Set the query to Prepared - will improve performance }
    ADOQuery.Prepared := true;

    try
      ADOQuery.Active := True;
    except
      on e: EADOError do
      begin
        MessageDlg('Error while doing query', mtError,
                   [mbOK], 0);

        Exit;
      end;
    end;

    { Create the data source }
    DataSrc := TDataSource.Create(Self);
    DataSrc.DataSet := ADOQuery;
```

```
    DataSrc.Enabled := true;

    { Finally initilalize the grid }
    DBGrid1.DataSource := DataSrc;
  end;
```

See Also

OnLogin

BeforeConnect

AfterConnect

OnGetUserName

OnLogin

OnLogin

Controlling Server Login

47.1.5.4.3 TDispatchConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName (see page 4807) or ServerGUID (see page 4807). When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection (see page 4796) define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker (see page 4787).

See Also

DoConnect

ServerName (see page 4807)

ServerGUID (see page 4807)

TSimpleObjectBroker (see page 5916)

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

47.1.5.4.4 TDispatchConnection.ServerGUID Property

Specifies the GUID of the application server with which to connect.

Pascal

```
property ServerGUID: string;
```

C++

```
__property UnicodeString ServerGUID;
```

Description

Use ServerGUID to specify the application server to which the client application should connect by its GUID (global unique identifier). If the value specifies a valid GUID for a registered application server, setting ServerGUID changes the value of ServerName (see page 4807) to the class name of the corresponding COM object.

Using ServerGUID rather than ServerName (see page 4807) to identify the application server is more robust because it does not require the application server to be registered on the client system.

Note: Either ServerName (see page 4807) or ServerGUID must be provided so that the dispatch connection can create and communicate with the appropriate server COM object. This is true even for connections that are not formed using DCOM, because the remote data module on the application server is always implemented as a COM object.

See Also

ServerName (see page 4807)

Connecting to the Application Server

Connecting to the Application Server using DataSnap Components

47.1.5.4.5 TDispatchConnection.ServerName Property

Specifies the name of the application server with which to connect.

Pascal

```
property ServerName: string;
```

C++

```
__property UnicodeString ServerName;
```

Description

Use ServerName to specify the name of the application server to which the client application should connect. At design time, the Object Inspector provides a drop-down list of all available servers if one of the following conditions is met:

For DCOM and OLEnterprise connections, the servers must be registered in the system Registry (see page 6599).

For Socket connections, ScktSrvr.exe must be running on the server machine.

For Web connections, the connection component must have an URL for locating httpsrvr.dll.

If ServerName specifies a valid name for a registered application server, setting ServerName sets the ServerGUID (see page 4807) property to the GUID that corresponds to the named application server.

Note: Either ServerName or ServerGUID (see page 4807) must be provided so that the dispatch connection can create and communicate with the appropriate server COM object. This is true even for connections that are not formed using DCOM, because the remote data module on the application server is always implemented as a COM object. Using ServerGUID (see page 4807) rather than ServerName to identify the application server is more robust because it does not require the application server to be registered on the client system when using a COM-based connection.

See Also

ServerGUID (see page 4807)

Connecting to the Application Server

47.1.5.5 TDispatchConnection Events

The events of the TDispatchConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

47.1.5.5.1 TDispatchConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

47.1.5.5.2 TDispatchConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

47.1.5.5.3 TDispatchConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

47.1.5.5.4 TDispatchConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

47.1.5.5.5 TDispatchConnection.OnGetUsername Event

Pascal

```
property OnGetUsername;
```

C++

```
__property OnGetUsername;
```

Description

This is OnGetUsername, a member of class TDispatchConnection.

47.1.5.5.6 TDispatchConnection.OnLogin Event

Occurs when a communication channel opens to the server.

Pascal

```
property OnLogin;
```

C++

```
__property OnLogin;
```

Description

OnLogin occurs when connections are opened if the LoginPrompt property is true.

For DataSnap SOAP connection components, the following sequence of events occurs when opening a connection to an application server:

1. An OnGetUsername event occurs, where the application can supply an initial user name for the login dialog.
2. The login dialog appears, where the user can supply a user name and password.
3. A BeforeConnect event occurs, the connection is opened, and an AfterConnect event occurs.
4. The OnLogin event occurs, where you can write code to log in to the application server using the username and password that were provided in the remote login dialog.

   Note: Do not use the OnLogin event if the connection component provides a different scheme for logging in. For example, TWebConnection uses the Username and Password properties instead.

   For ADO connection components, the following sequence of events occurs when opening a connection to an application server:

1. A BeforeConnect event occurs.

2. An OnWillConnect event occurs, where the application can supply an initial user name and password for the login dialog, as well as the connection options and status.

3. The login dialog appears, where the user can supply a user name and password.

4. The OnLogin event occurs, where you can write code to respond to the user-supplied values. (For example, you could convert an application-specific user name and password to database user name and password).

5. The connection is opened, and an AfterConnect event occurs.

Note: Some

C connection components that do not connect to an application server in a multitiered database application replace the OnLogin

event defined in TCustomConnection with a different event that occurs before a connection is established.

See Also

Connected

Open

OnGetUserName

BeforeConnect

AfterConnect

LoginPrompt

OnWillConnect

47.1.6 TOLEnterpriseConnection Class

TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.

Class Hierarchy

Pascal

```
TOLEnterpriseConnection = class(TCOMConnection);
```

C++

```
class TOLEnterpriseConnection : public TCOMConnection;
```

File

MConnect

Description

TOLEnterpriseConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server using OLEnterprise. This component is provided for backward compatibility.

TOLEnterpriseConnection establishes the initial connection between the client application and a remote application server using OLEnterprise. Once the connection is established, the client registers any or all of its client datasets with the remote server, and these client datasets use the IAppServer (see page 5222) interface from the OLEnterprise connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection (see page 4790) component instead. To connect client applications to remote application servers using TCP/IP sockets, use TSocketConnection. To connect client applications to remote application servers using HTTP, use TWebConnection. To connect to remote application servers using SOAP, use TSOAPConnection.

See Also

TClientDataSet

TSocketConnection

TDCOMConnection (see page 4790)

TRemoteDataModule

IAppServer (see page 5222)

TWebConnection

47.1.6.1 TOLEnterpriseConnection Members

The following tables list the members exposed by TOLEnterpriseConnection.

Protected Methods

	Name	Description
	DoConnect (see page 4813)	Establishes a connection to the application server.
	DoDisconnect (see page 4798)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.
	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4786)	Sets the Connected (see page 4802) property.
	SetConnected (see page 4800)	Sets the Connected (see page 4802) property.

Protected Properties

	Name	Description
	ObjectBroker (see page 4806)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Public Methods

	Name	Description
	Create (see page 4797)	Creates an instance of a dispatch connection component.
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

Published Properties

	Name	Description
	BrokerName (see page 4813)	Specifies the name of the Business Object Broker that locates the application server.
	ComputerName (see page 4814)	Specifies the name of the machine on which the application server is located.
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.

47.1.6.2 TOLEnterpriseConnection Methods

The methods of the TOLEnterpriseConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 4813)	Establishes a connection to the application server.

47.1.6.2.1 TOLEnterpriseConnection.DoConnect Method

Establishes a connection to the application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected (see page 4802) property is set to true. DoConnect obtains the interface of the remote data module on the application server using OLEnterprise runtime. This interface can then be used to obtain interfaces for the providers on the application server or to make interface calls to the remote data module.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

47.1.6.3 TOLEnterpriseConnection Properties

The properties of the TOLEnterpriseConnection class are listed here.

Published Properties

	Name	Description
	BrokerName (see page 4813)	Specifies the name of the Business Object Broker that locates the application server.
	ComputerName (see page 4814)	Specifies the name of the machine on which the application server is located.

47.1.6.3.1 TOLEnterpriseConnection.BrokerName Property

Specifies the name of the Business Object Broker that locates the application server.

Pascal

```
property BrokerName: string;
```

C++

```
__property UnicodeString BrokerName;
```

Description

Use BrokerName to specify the OLEnterprise Business Object Broker with which the application server is registered. The Business Object Broker locates an available application server with the name specified by the ServerName (see page 4807) property.

If you are not using the Business Object Broker, use the ComputerName (see page 4814) property instead. You must specify a value for either BrokerName or ComputerName (see page 4814):

Note: Setting the value of BrokerName changes the ComputerName (see page 4814) property to blank, and vice versa.

See Also

ServerName (see page 4807)

ComputerName (see page 4814)

47.1.6.3.2 TOLEnterpriseConnection.ComputerName Property

Specifies the name of the machine on which the application server is located.

Pascal

```
property ComputerName: string;
```

C++

```
__property UnicodeString ComputerName;
```

Description

Use ComputerName to specify the machine where the application server resides if you are not using the Business Object Broker. If you are not using the Business Object Broker and ComputerName is blank (the default), TOLEnterpriseConnection (see page 4811) assumes the application server can be found on the user's machine or that the computer name can be derived from the system registry based on the DCOM configuration. If ComputerName is not blank, it overrides any system registry entries.

If you are using the Business Object Broker, use the BrokerName (see page 4813) property instead. BrokerName (see page 4813) and ComputerName are mutually exclusive: Setting the value of ComputerName changes the BrokerName (see page 4813) property to blank, and vice versa.

See Also

ServerName (see page 4807)

BrokerName (see page 4813)

47.1.7 TSharedConnection Class

TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

Class Hierarchy

Pascal

```
TSharedConnection = class(TCustomRemoteServer);
```

C++

```
class TSharedConnection : public TCustomRemoteServer;
```

File

MConnect

Description

TSharedConnection manages a connection to a "child" remote data module that is managed by a single main remote data module on an application server. The client application does not connect directly to the "child" remote data module. Rather, it uses a single connection to the main remote data module on the server machine, and that main remote data module dispatches calls to the IAppServer (see page 5222) interface of other "child" data modules. This allows the client application to use a single connection rather than one connection for each remote data module on the application server.

On the server-side, the interface for the child remote data module is surfaced as a property of the main remote data module's interface. The TSharedConnection component uses this property name to indicate which child remote data module is the actual target of its connection.

TSharedConnection uses a TDispatchConnection (see page 4796) descendant (TDCOMConnection (see page 4790), TWebConnection, or TSocketConnection) to form the connection to the main remote data module on the application server. When you use TSharedConnection to open a connection, it causes the associated dispatch connection to connect to the main remote data module if it has not already done so. When you close the connection maintained by TSharedConnection, however, it severs only the connection to the child remote data module, leaving the connection to the main remote data module open in case other components on the client application are still using it.

See Also

TCustomClientDataSet

TSocketConnection

TWebConnection

TDCOMConnection (see page 4790)

Connecting to an Application Server That Uses Multiple Data Modules

47.1.7.1 TSharedConnection Members

The following tables list the members exposed by TSharedConnection.

Protected Methods

	Name	Description
	ConnectEvent (see page 4817)	This is ConnectEvent, a member of class TSharedConnection.
V	DoConnect (see page 4817)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 4818)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 4818)	Returns the value of the Connected property.
V	Notification (see page 4819)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	Destroy (see page 4816)	Destroys the TSharedConnection (see page 4814) instance.
V	GetServer (see page 4818)	Returns the IAppServer (see page 5222) interface of the child remote data module.

Published Events

	Name	Description
	AfterConnect (see page 4820)	Occurs after a connection is established.
	AfterDisconnect (see page 4821)	Occurs after the connection closes.
	BeforeConnect (see page 4821)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4822)	Occurs immediately before the connection closes.

Published Methods

	Name	Description
	Create (see page 4816)	Creates an instance of a remote server component.

Published Properties

	Name	Description
	ChildName (see page 4819)	Identifies the child remote data module on the application server with which you want to communicate.
	ParentConnection (see page 4820)	Specifies the connection component that establishes a connection to the main remote data module on the application server.

47.1.7.2 TSharedConnection.Create Constructor

Creates an instance of a remote server component.

Pascal

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

C++

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

Description

Create instantiates an instance of a remote server component and initializes its properties.

AOwner is the component that is responsible for freeing the TCustomRemoteServer instance. It becomes the value of the Owner property.

See Also

Create

Owner

47.1.7.3 TSharedConnection.Destroy Destructor

Destroys the TSharedConnection (see page 4814) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSharedConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the reference to the shared connection component is not nil before calling Destroy

See Also

System::TObject::Free

47.1.7.4 TSharedConnection Methods

The methods of the TSharedConnection class are listed here.

Protected Methods

	Name	Description
	ConnectEvent (see page 4817)	This is ConnectEvent, a member of class TSharedConnection.
V	DoConnect (see page 4817)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 4818)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 4818)	Returns the value of the Connected property.
V	Notification (see page 4819)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	GetServer (see page 4818)	Returns the IAppServer (see page 5222) interface of the child remote data module.

47.1.7.4.1 TSharedConnection.ConnectEvent Method

Pascal

```
procedure ConnectEvent(Sender: TObject; Connecting: Boolean);
```

C++

```
__fastcall ConnectEvent(TObject * Sender, Boolean Connecting);
```

Description

This is ConnectEvent, a member of class TSharedConnection.

47.1.7.4.2 TSharedConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

47.1.7.4.3 TSharedConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

47.1.7.4.4 TSharedConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

47.1.7.4.5 TSharedConnection.GetServer Method

Returns the IAppServer (see page 5222) interface of the child remote data module.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Call GetServer to obtain the interface of the remote data module on the application server. GetServer returns the IAppServer (see page 5222) interface of the same remote data module as the one whose interface is the value of the AppServer property. Client applications can communicate directly with the child remote data module through this interface.

See Also

AppServer

47.1.7.4.6 TSharedConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

47.1.7.5 TSharedConnection Properties

The properties of the TSharedConnection class are listed here.

Published Properties

	Name	Description
	ChildName (see page 4819)	Identifies the child remote data module on the application server with which you want to communicate.
	ParentConnection (see page 4820)	Specifies the connection component that establishes a connection to the main remote data module on the application server.

47.1.7.5.1 TSharedConnection.ChildName Property

Identifies the child remote data module on the application server with which you want to communicate.

Pascal

```
property ChildName: String;
```

C++

```
__property UnicodeString ChildName;
```

Description

Set ChildName to the name of the property on the application server's main remote data module that exposes the interface for the desired child remote data module. TSharedConnection (see page 4814) uses this property to obtain the interface for the child remote data module, which it sets as the value of the AppServer property.

See Also

AppServer

ParentConnection (see page 4820)

47.1.7.5.2 TSharedConnection.ParentConnection Property

Specifies the connection component that establishes a connection to the main remote data module on the application server.

Pascal

```
property ParentConnection: TDispatchConnection;
```

C++

```
__property TDispatchConnection ParentConnection;
```

Description

Set ParentConnection to a connection component that connects to the main remote data module on the application server. TSharedConnection (see page 4814) uses this component's connection for communicating to the child remote data module.

The main remote data module with which ParentConnection communicates must expose the interface for the child remote data module as a property on its IAppServer (see page 5222) interface.

See Also

AppServer

ChildName (see page 4819)

AppServer

47.1.7.6 TSharedConnection Events

The events of the TSharedConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 4820)	Occurs after a connection is established.
	AfterDisconnect (see page 4821)	Occurs after the connection closes.
	BeforeConnect (see page 4821)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4822)	Occurs immediately before the connection closes.

47.1.7.6.1 TSharedConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

47.1.7.6.2 TSharedConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

47.1.7.6.3 TSharedConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

47.1.7.6.4 TSharedConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

47.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetMIDASAppServerList (see page 4823)	This is function MConnect.GetMIDASAppServerList.

47.2.1 MConnect.GetMIDASAppServerList Function

Pascal

```
procedure GetMIDASAppServerList(List: TStringList; const RegCheck: string);
```

C++

```
GetMIDASAppServerList(TStringList List, const UnicodeString RegCheck);
```

File

MConnect

Description

This is function MConnect.GetMIDASAppServerList.

48 Menus Namespace

This is namespace Menus.

Topics

Name	Description
Classes (see page 4827)	The following table lists classes in this documentation.
Functions (see page 4951)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4965)	The following table lists structs, records, enums in this documentation.
Types (see page 4970)	The following table lists types in this documentation.
Variables (see page 4975)	The following table lists variables in this documentation.
Constants (see page 4977)	The following table lists constants in this documentation.

Classes

	Name	Description
	EMenuError (see page 4827)	EMenuError is the exception class for errors involving menu items.
	TMainMenu (see page 4828)	TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.
	TMenu (see page 4837)	TMenu is the base type for menu components such as TMainMenu (see page 4828) and TPopupMenu (see page 4935).
	TMenuActionLink (see page 4858)	TMenuActionLink links menu item clients to the corresponding menu action.
	TMenuItem (see page 4869)	TMenuItem describes the properties of an item in a menu.
	TMenuItemEnumerator (see page 4929)	This is class Menus.TMenuItemEnumerator.
	TMenuItemStack (see page 4931)	This is class Menus.TMenuItemStack.
	TPopupList (see page 4932)	TPopupList manages the window that handles popup menu messages.
	TPopupMenu (see page 4935)	TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

Constants

Name	Description
cDialogSuffix (see page 4977)	This is constant Menus.cDialogSuffix.
cHotkeyPrefix (see page 4977)	This is constant Menus.cHotkeyPrefix.
cLineCaption (see page 4977)	This is constant Menus.cLineCaption.

Functions

	Name	Description
	AnsiSameCaption (see page 4951)	Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.

	DrawMenuItem (see page 4952)	This is function Menus.DrawMenuItem.
	GetHotkey (see page 4952)	Returns the last character in a string marked with an ampersand (&).
	IsAltGRPressed (see page 4952)	This is function Menus.IsAltGRPressed.
	NewItem (see page 4953)	Creates and initializes a menu item.
	NewLine (see page 4954)	Creates a menu item for a separator.
	NewMenu (see page 4955)	Creates and initializes a main menu.
	NewPopupMenu (see page 4956)	Creates and initializes a popup menu.
	NewSubMenu (see page 4957)	Creates and initializes a menu item that represents a submenu.
	SameCaption (see page 4958)	This is function Menus.SameCaption.
	ShortCut (see page 4958)	Creates a menu shortcut for a specified key combination.
	ShortCutFromMessage (see page 4959)	This is function Menus.ShortCutFromMessage.
	ShortCutToKey (see page 4960)	Returns the key code and shift state of a menu shortcut.
	ShortCutToText (see page 4961)	Converts a menu shortcut into a string that describes the shortcut.
	StripHotkey (see page 4962)	This is function Menus.StripHotkey.
	TextToShortCut (see page 4963)	Creates a menu shortcut from a text string.

Structs, Records, Enums

	Name	Description
	TFindItemKind (see page 4965)	TFindItemKind indicates the criterion to use when locating menu items.
	TMenuAnimations (see page 4966)	TMenuAnimation (see page 4971) and TMenuAnimations indicate how a popup menu appears on screen.
	TMenuBreak (see page 4966)	TMenuBreak indicates the criterion to use when locating menu items.
	TMenuItemAutoFlag (see page 4967)	TMenuItemAutoFlag and TMenuAutoFlag (see page 4972) Indicate when menu items are automatically maintained.
	TPopupAlignment (see page 4968)	TPopupAlignment represents the position of a popup menu relative to the mouse.
	TTrackButton (see page 4968)	TTrackButton specifies which mouse button activates a tool button menu.

Types

Name	Description
PMenuItem (see page 4970)	This is type Menus.PMenuItem.
TAdvancedMenuDrawItemEvent (see page 4970)	TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuActionLinkClass (see page 4971)	This is type Menus.TMenuActionLinkClass.
TMenuAnimation (see page 4971)	TMenuAnimation and TMenuAnimations (see page 4966) indicate how a popup menu appears on screen.
TMenuAutoFlag (see page 4972)	TMenuItemAutoFlag (see page 4967) and TMenuAutoFlag Indicate when menu items are automatically maintained.
TMenuChangeEvent (see page 4972)	This is type Menus.TMenuChangeEvent.
TMenuDrawItemEvent (see page 4973)	TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuHandle (see page 4973)	This is type Menus.TMenuHandle.
TMenuMeasureItemEvent (see page 4973)	TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

Variables

Name	Description
PopupList (see page 4975)	This is variable Menus.PopupList.
ShortCutItems (see page 4975)	This is variable Menus.ShortCutItems.
ValidMenuHotkeys (see page 4975)	This is variable Menus.ValidMenuHotkeys.

48.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMenuError (see page 4827)	EMenuError is the exception class for errors involving menu items.
	TMainMenu (see page 4828)	TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.
	TMenu (see page 4837)	TMenu is the base type for menu components such as TMainMenu (see page 4828) and TPopupMenu (see page 4935).
	TMenuActionLink (see page 4858)	TMenuActionLink links menu item clients to the corresponding menu action.
	TMenuItem (see page 4869)	TMenuItem describes the properties of an item in a menu.
	TMenuItemEnumerator (see page 4929)	This is class Menus.TMenuItemEnumerator.
	TMenuItemStack (see page 4931)	This is class Menus.TMenuItemStack.
	TPopupList (see page 4932)	TPopupList manages the window that handles popup menu messages.
	TPopupMenu (see page 4935)	TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

48.1.1 EMenuError Class

EMenuError is the exception class for errors involving menu items.

Class Hierarchy

Pascal

```
EMenuError = class(Exception);
```

C++

```
class EMenuError : public Exception;
```

File

Menus

Description

EMenuError is raised if an error occurs while an application is working with menu items. For example, an EMenuError exception occurs if

An application tries to delete an item from an empty menu

A menu item attempts to insert itself into it's own list of items

An application attempts to reference a menu that does not exist

An application specifies an invalid GroupIndex for a menu item

See Also

TMenuItem (see page 4869)

TMenu (see page 4837)

48.1.2 TMainMenu Class

TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.

Class Hierarchy

Pascal

```
TMainMenu = class(TMenu);
```

C++

```
class TMainMenu : public TMenu;
```

File

Menus

Description

Use TMainMenu to provide the main menu for a form. To begin designing a menu, add a main menu to a form, and double-click the component.

TMainMenu introduces properties and methods for merging the drop-down menus of the main menu with the main menu of another form.

TMainMenu introduces properties and methods for assisting in the menu negotiation process of an OLE container.

Tip: To create a menu that users can configure at runtime, use the TActionMainMenuBar component with an action manager instead.

See Also

TMenuItem (see page 4869)

TPopupMenu (see page 4935)

TActionMainMenuBar

TActionManager

48.1.2.1 TMainMenu Members

The following tables list the members exposed by TMainMenu.

Protected Methods

	Name	Description
	AdjustBiDiBehavior (see page 4841)	Applies the BiDiMode (see page 4850) property to the menu when the Windows control is created.
	DoBiDiModeChanged (see page 4842)	Updates the menu when the BiDiMode (see page 4850) property changes.
	DoChange (see page 4842)	Generates an OnChange (see page 4858) event.
	DoGetMenuString (see page 4843)	Returns the caption for an item in the menu.
	GetHandle (see page 4830)	Returns the value of the Handle property.
	IsOwnerDraw (see page 4845)	Indicates whether menu items can contain more than simple captions.
	Loaded (see page 4846)	Calls the DoChange (see page 4842) method when the menu is first loaded into memory.
	MenuChanged (see page 4831)	Responds when the menu changes.

	Notification (see page 4847)	Responds when components are added or deleted.
	SetChildOrder (see page 4848)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	UpdateItems (see page 4849)	Causes all menu items to update themselves.

Public Methods

	Name	Description
	Create (see page 4839)	Creates an instance of TMenu (see page 4837).
	CreateMenuItem (see page 4841)	This is CreateMenuItem, a member of class TMenu.
	Destroy (see page 4840)	Destroys the menu component.
	DispatchCommand (see page 4841)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 4842)	Dispatches a pop-up menu command.
	FindItem (see page 4843)	Locates and returns a menu item.
	GetChildren (see page 4844)	Executes a callback for child menu items.
	GetHelpContext (see page 4844)	Returns the help context ID associated with a menu item.
	GetOle2AcceleratorTable (see page 4830)	Returns the main menu's accelerator table.
	IsRightToLeft (see page 4845)	Indicates whether menu items read from right to left.
	IsShortCut (see page 4845)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	Merge (see page 4831)	Combines the main menu of one form with the main menu of another in non-MDI applications.
	ParentBiDiModeChanged (see page 4847)	This is the overview for the ParentBiDiModeChanged method overload.
	PopulateOle2Menu (see page 4832)	Populates an OLE menu with items from the main menu.
	ProcessMenuChar (see page 4848)	Responds to keyboard input for the menu.
	SetOle2MenuHandle (see page 4832)	Associates the handle for an OLE menu with the main menu.
	Unmerge (see page 4832)	Reverses the merging of two menus in a non-MDI application.

Public Properties

	Name	Description
	Handle (see page 4851)	Provides access to the Windows menu handle for the menu.
	WindowHandle (see page 4857)	Provides access to the handle of the window that uses the menu.

Published Events

	Name	Description
	OnChange (see page 4837)	Occurs when the menu changes.

Published Properties

	Name	Description
	AutoHotkeys (see page 4833)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4834)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoMerge (see page 4834)	Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.
	BiDiMode (see page 4835)	Determines the layout of the menu in Middle East locales.
	Images (see page 4835)	Lists the images that can appear beside individual menu items.
	Items (see page 4852)	Describes the elements of the menu.
	OwnerDraw (see page 4836)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 4836)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.

48.1.2.2 TMainMenu Methods

The methods of the TMainMenu class are listed here.

Protected Methods

	Name	Description
	GetHandle (see page 4830)	Returns the value of the Handle property.
	MenuChanged (see page 4831)	Responds when the menu changes.

Public Methods

	Name	Description
	GetOle2AcceleratorTable (see page 4830)	Returns the main menu's accelerator table.
	Merge (see page 4831)	Combines the main menu of one form with the main menu of another in non-MDI applications.
	PopulateOle2Menu (see page 4832)	Populates an OLE menu with items from the main menu.
	SetOle2MenuHandle (see page 4832)	Associates the handle for an OLE menu with the main menu.
	Unmerge (see page 4832)	Reverses the merging of two menus in a non-MDI application.

48.1.2.2.1 TMainMenu.GetHandle Method

Returns the value of the Handle property.

Pascal

```
function GetHandle: HMENU; override;
```

C++

```
virtual __fastcall HMENU GetHandle();
```

Description

GetHandle is the protected implementation of the Handle property. Override this method to change the implementation of the handle property. For example, TMainMenu overrides GetHandle to adjust for menus added by embedded OLE objects.

See Also

Handle (see page 4851)

48.1.2.2.2 TMainMenu.GetOle2AcceleratorTable Method

Returns the main menu's accelerator table.

Pascal

```
procedure GetOle2AcceleratorTable(var AccelTable: HAccel; var AccelCount: Integer; const
Groups: array of Integer);
```

C++

```
__fastcall GetOle2AcceleratorTable(HAccel AccelTable, int AccelCount, const array of Integer
Groups);
```

Description

Call GetOle2AcceleratorTable to obtain a handle to an accelerator table for all menu items with a GroupIndex value that appears in the Groups parameter. The AccelTable parameter is set to the accelerator table handle, and the AccelCount is set to the number of entries in the table. GetOle2AcceleratorTable is used by OLE containers when merging menus.

See Also

GroupIndex (see page 4914)

ShortCut (see page 4920)

48.1.2.2.3 TMainMenu.MenuChanged Method

Responds when the menu changes.

Pascal

```
procedure MenuChanged(Sender: TObject; Source: TMenuItem; Rebuild: Boolean); override;
```

C++

```
virtual __fastcall MenuChanged(TObject * Sender, TMenuItem Source, Boolean Rebuild);
```

Description

MenuChanged is called automatically when changes occur that affect the structure of the menu. The Sender parameter indicates the object that originates the change. It's value can be nil (Delphi) or NULL (C++). Source indicates which menu item is affected. Rebuild indicates whether the menu must be rebuilt to reflect the change (for example when items are added or deleted).

MenuChanged calls the DoChange method, which generates an OnChange event.

Note: Changes that affect individual items rather than the menu as a whole trigger the UpdateItems method instead of MenuChanged. For example, when the Images property changes, UpdateItems is called rather than MenuChanged.

See Also

Items (see page 4852)

UpdateItems (see page 4849)

DoChange (see page 4842)

48.1.2.2.4 TMainMenu.Merge Method

Combines the main menu of one form with the main menu of another in non-MDI applications.

Pascal

```
procedure Merge(Menu: TMainMenu);
```

C++

```
__fastcall Merge(TMainMenu Menu);
```

Description

Call Merge to merge another main menu with this one. Specify the menu you want merged with this menu as the Menu parameter.

For example, to use the main menu of the first form in an application as the main menu for the application, call Merge to merge the main menu of any additional forms in the application with the main menu of the first form.

Depending on the value of the GroupIndex property of menu items on the main menu, the merged menu items can replace menu items on the menu bar, or insert menu items into the menu bar. See GroupIndex for information on how to do these things.

To merge and unmerge main menus automatically when other forms are displayed, change the value of the AutoMerge (see page 4834) property to true on all forms whose menus should be merged in.

See Also

AutoMerge (see page 4834)

GroupIndex (see page 4914)

Unmerge (see page 4832)

48.1.2.2.5 TMainMenu.PopulateOle2Menu Method

Populates an OLE menu with items from the main menu.

Pascal

```
procedure PopulateOle2Menu(SharedMenu: HMenu; const Groups: array of Integer; var Widths:
array of Longint);
```

C++

```
__fastcall PopulateOle2Menu(HMenu SharedMenu, const array of Integer Groups, array of Longint
Widths);
```

Description

Call PopulateOle2Menu to merge the menu items with GroupIndex values specified by the Groups parameter into the menu that uses the SharedMenu handle. PopulateOle2Menu appends all menu items in the specified groups to the menu, and increments the corresponding entries in the Widths array for every menu item appended. PopulateOle2Menu is used by OLE containers when merging menus.

Note: PopulateOle2Menu populates the menu specified by the SharedMenu parameter but does not set this as the OLE menu associated with the menu.

See Also

GetOle2AcceleratorTable (see page 4830)

SetOle2MenuHandle (see page 4832)

GroupIndex (see page 4914)

48.1.2.2.6 TMainMenu.SetOle2MenuHandle Method

Associates the handle for an OLE menu with the main menu.

Pascal

```
procedure SetOle2MenuHandle(Handle: HMENU);
```

C++

```
__fastcall SetOle2MenuHandle(HMENU Handle);
```

Description

Call SetOle2MenuHandle to temporarily replace the menu of a form with the menu represented by the Handle (see page 4851) parameter. To restore the original menu, call SetOle2MenuHandle with the Handle (see page 4851) parameter set to 0. SetOle2MenuHandle allows an OLE container to use menu items provided by an OLE server.

See Also

GetOle2AcceleratorTable (see page 4830)

PopulateOle2Menu (see page 4832)

48.1.2.2.7 TMainMenu.Unmerge Method

Reverses the merging of two menus in a non-MDI application.

Pascal

```
procedure Unmerge(Menu: TMainMenu);
```

C++

```
__fastcall Unmerge(TMainMenu Menu);
```

Description

Call Unmerge to remove items from another main menu that were added using the Merge (see page 4831) method. The Menu parameter is the merged menu whose items should be removed from the main menu.

See Also

AutoMerge (see page 4834)

Merge (see page 4831)

48.1.2.3 TMainMenu Properties

The properties of the TMainMenu class are listed here.

Published Properties

	Name	Description
	AutoHotkeys (see page 4833)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4834)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoMerge (see page 4834)	Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.
	BiDiMode (see page 4835)	Determines the layout of the menu in Middle East locales.
	Images (see page 4835)	Lists the images that can appear beside individual menu items.
	OwnerDraw (see page 4836)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 4836)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.

48.1.2.3.1 TMainMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its

Items property. Setting one sets the other.

See Also

AutoHotkeys (see page 4900)

Items (see page 4852)

RethinkHotkeys (see page 4894)

48.1.2.3.2 TMainMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoLineReduction (see page 4900)

Items (see page 4852)

RethinkLines (see page 4894)

48.1.2.3.3 TMainMenu.AutoMerge Property

Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.

Pascal

```
property AutoMerge: Boolean;
```

C++

```
__property Boolean AutoMerge;
```

Description

To merge a form's menus with the main menu in the main form, set the AutoMerge property of each main menu you want merged to true. Make sure that the AutoMerge property of the main menu of the main form (the one other menus are merged

into) remains false. How menus merge depends on the value of the GroupIndex property for each menu item.

If the application is an MDI application (the FormStyle properties are set so the main form is a parent form and subsequent forms are child forms), menu merging occurs automatically without using the AutoMerge property.

See Also

FormStyle (see page 2162)

GroupIndex (see page 4914)

Merge (see page 4831)

48.1.2.3.4 TMainMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 4857)

IsRightToLeft (see page 4845)

48.1.2.3.5 TMainMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 4916)

Images

Items (see page 4852)

48.1.2.3.6 TMainMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 4929)

OnDrawItem (see page 4928)

Images (see page 4851)

48.1.2.3.7 TMainMenu.ParentBiDiMode Property

Indicates whether the BiDiMode property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode setting and control the menu's layout independently.

See Also

BiDiMode (see page 4850)

48.1.2.4 TMainMenu Events

The events of the TMainMenu class are listed here.

Published Events

	Name	Description
	OnChange (see page 4837)	Occurs when the menu changes.

48.1.2.4.1 TMainMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

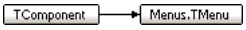
See Also

Items (see page 4852)

48.1.3 TMenu Class

TMenu is the base type for menu components such as TMainMenu (see page 4828) and TPopupMenu (see page 4935).

Class Hierarchy



Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TMenu = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TMenu : public TComponent;
```

File

Menus

Description

Use TMenu as a base class when defining a component that represents a collection of menu items. TMenu introduces properties and methods to create a Windows menu that represents a set of menu items, dispatch commands associated with those menu items, and locate the help context id for any of those menu items.

See Also

TMainMenu (see page 4828)

TMenuItem (see page 4869)

TPopupMenu (see page 4935)

Disabling Menu Items

Adding Images to Menu Items

Adding

Building Menus

Determining the Order of Merged Menu Items: GroupIndex Property

Editing Menu Items in the Object Inspector

Importing Resource Files

Manipulating Menu Items at Runtime

Merging Menus

Moving Menu Items

Naming Conventions for Template Menu Items and Event Handlers

Naming Menus

Naming the Menu Items

Opening the Menu Designer

Saving a Menu as a Template

Specifying Accelerator Keys and Keyboard Shortcuts

Specifying the Active Menu: Menu Property

Switching Between Menus at Design Time

Using Menu Templates

Using the Menu Designer Context Menu

Viewing the Menu

48.1.3.1 TMenu Members

The following tables list the members exposed by TMenu.

Protected Events

	Name	Description
	OnChange (see page 4858)	Occurs when the menu changes.

Protected Methods

	Name	Description
	AdjustBiDiBehavior (see page 4841)	Applies the BiDiMode (see page 4850) property to the menu when the Windows control is created.
	DoBiDiModeChanged (see page 4842)	Updates the menu when the BiDiMode (see page 4850) property changes.
	DoChange (see page 4842)	Generates an OnChange (see page 4858) event.
	DoGetMenuString (see page 4843)	Returns the caption for an item in the menu.
	GetHandle (see page 4844)	Returns the value of the Handle (see page 4851) property.
	IsOwnerDraw (see page 4845)	Indicates whether menu items can contain more than simple captions.
	Loaded (see page 4846)	Calls the DoChange (see page 4842) method when the menu is first loaded into memory.
	MenuChanged (see page 4846)	Responds when the menu changes.
	Notification (see page 4847)	Responds when components are added or deleted.
	SetChildOrder (see page 4848)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	UpdateItems (see page 4849)	Causes all menu items to update themselves.

Public Methods

	Name	Description
	Create (see page 4839)	Creates an instance of TMenu (see page 4837).
	CreateMenuItem (see page 4841)	This is CreateMenuItem, a member of class TMenu.
	Destroy (see page 4840)	Destroys the menu component.
	DispatchCommand (see page 4841)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 4842)	Dispatches a pop-up menu command.
	FindItem (see page 4843)	Locates and returns a menu item.
	GetChildren (see page 4844)	Executes a callback for child menu items.
	GetHelpContext (see page 4844)	Returns the help context ID associated with a menu item.
	IsRightToLeft (see page 4845)	Indicates whether menu items read from right to left.
	IsShortCut (see page 4845)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	ParentBiDiModeChanged (see page 4847)	This is the overview for the ParentBiDiModeChanged method overload.
	ProcessMenuChar (see page 4848)	Responds to keyboard input for the menu.

Public Properties

	Name	Description
	AutoHotkeys (see page 4849)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4850)	Determines whether redundant separator bars are automatically removed from the menu.
	BiDiMode (see page 4850)	Determines the layout of the menu in Middle East locales.
	Handle (see page 4851)	Provides access to the Windows menu handle for the menu.
	Images (see page 4851)	Lists the images that can appear beside individual menu items.
	OwnerDraw (see page 4856)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 4857)	Indicates whether the BiDiMode (see page 4850) property is inherited from the control that uses the menu.
	WindowHandle (see page 4857)	Provides access to the handle of the window that uses the menu.

Published Properties

	Name	Description
	Items (see page 4852)	Describes the elements of the menu.

48.1.3.2 TMenu.Create Constructor

Creates an instance of TMenu (see page 4837).

Pascal

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

C++

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

Description

Call Create to create a menu at runtime. Menus (see page 4825) placed on forms at design time are created automatically. Create calls the inherited Create method, then sets the initial values for the menu component, including creating an Items (see page 4852) property.

The AOwner parameter indicates the component (usually a form) that is responsible for managing the memory associated with the menu.

See Also

Items (see page 4852)

48.1.3.3 TMenu.Destroy Destructor

Destroys the menu component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMenu (see page 4837) reference is not nil, and only then calls Destroy. Destroy frees the Items (see page 4852) property, and then calls the inherited Destroy method.

See Also

System::TObject::Free

Items (see page 4852)

48.1.3.4 TMenu Methods

The methods of the TMenu class are listed here.

Protected Methods

	Name	Description
	AdjustBiDiBehavior (see page 4841)	Applies the BiDiMode (see page 4850) property to the menu when the Windows control is created.
	DoBiDiModeChanged (see page 4842)	Updates the menu when the BiDiMode (see page 4850) property changes.
	DoChange (see page 4842)	Generates an OnChange (see page 4858) event.
	DoGetMenuString (see page 4843)	Returns the caption for an item in the menu.
	GetHandle (see page 4844)	Returns the value of the Handle (see page 4851) property.
	IsOwnerDraw (see page 4845)	Indicates whether menu items can contain more than simple captions.
	Loaded (see page 4846)	Calls the DoChange (see page 4842) method when the menu is first loaded into memory.
	MenuChanged (see page 4846)	Responds when the menu changes.
	Notification (see page 4847)	Responds when components are added or deleted.
	SetChildOrder (see page 4848)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	UpdateItems (see page 4849)	Causes all menu items to update themselves.

Public Methods

	Name	Description
	CreateMenuItem (see page 4841)	This is CreateMenuItem, a member of class TMenu.
	DispatchCommand (see page 4841)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 4842)	Dispatches a pop-up menu command.
	FindItem (see page 4843)	Locates and returns a menu item.
	GetChildren (see page 4844)	Executes a callback for child menu items.
	GetHelpContext (see page 4844)	Returns the help context ID associated with a menu item.
	IsRightToLeft (see page 4845)	Indicates whether menu items read from right to left.
	IsShortCut (see page 4845)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	ParentBiDiModeChanged (see page 4847)	This is the overview for the ParentBiDiModeChanged method overload.
	ProcessMenuChar (see page 4848)	Responds to keyboard input for the menu.

48.1.3.4.1 TMenu.AdjustBiDiBehavior Method

Applies the BiDiMode (see page 4850) property to the menu when the Windows control is created.

Pascal

```
procedure AdjustBiDiBehavior;
```

C++

```
__fastcall AdjustBiDiBehavior();
```

Description

AdjustBiDiBehavior is called automatically when the Windows control for the menu is created. It ensures that the BiDiMode (see page 4850) property is properly reflected in the menu if the application is running in a Middle Eastern locale.

See Also

BiDiMode (see page 4850)

48.1.3.4.2 TMenu.CreateMenuItem Method

Pascal

```
function CreateMenuItem: TMenuItem; dynamic;
```

C++

```
__fastcall TMenuItem CreateMenuItem();
```

Description

This is CreateMenuItem, a member of class TMenu.

48.1.3.4.3 TMenu.DispatchCommand Method

Calls the Click method of the menu item with the specified value as its Command property.

Pascal

```
function DispatchCommand(ACommand: Word): Boolean;
```

C++

```
__fastcall Boolean DispatchCommand(Word ACommand);
```

Description

Call DispatchCommand to simulate a click on the menu item associated with a command ID. If DispatchCommand locates a menu item that matches the command ID, it calls the Click method for that menu item and returns true. If there is no such item, DispatchCommand returns false.

See Also

DispatchPopup (see page 4842)

IsShortCut (see page 4845)

Items (see page 4852)

Click (see page 4881)

Command (see page 4907)

48.1.3.4.4 TMenu.DispatchPopup Method

Dispatches a pop-up menu command.

Pascal

```
function DispatchPopup(AHandle: HMENU): Boolean;
```

C++

```
__fastcall Boolean DispatchPopup(HMENU AHandle);
```

Description

Call DispatchPopup to simulate a click on the menu item associated with the pop-up menu handle. If DispatchPopup locates a menu item that matches the pop-up menu handle, it calls the Click method for that menu item and returns true. If there is no such item, DispatchPopup returns false.

See Also

DispatchCommand (see page 4841)

IsShortCut (see page 4845)

Items (see page 4852)

Click (see page 4881)

48.1.3.4.5 TMenu.DoBiDiModeChanged Method

Updates the menu when the BiDiMode (see page 4850) property changes.

Pascal

```
procedure DoBiDiModeChanged;
```

C++

```
__fastcall DoBiDiModeChanged();
```

Description

Do not call DoBiDiModeChanged. It is called automatically when the menu's BiDiMode (see page 4850) property changes to update the menu to reflect the current setting.

See Also

BiDiMode (see page 4850)

48.1.3.4.6 TMenu.DoChange Method

Generates an OnChange (see page 4858) event.

Pascal

```
procedure DoChange(Source: TMenuItem; Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall DoChange(TMenuItem Source, Boolean Rebuild);
```

Description

DoChange is called automatically when the properties of any menu items change. The Source parameter specifies the menu item (if any) that changed. The Rebuild parameter indicates whether the change requires the menu to regenerate its items.

See Also

OnChange (see page 4858)

MenuChanged (see page 4846)

48.1.3.4.7 TMenu.DoGetMenuString Method

Returns the caption for an item in the menu.

Pascal

```
function DoGetMenuString(Menu: HMENU; ItemID: UINT; Str: PChar; MaxCount: Integer; Flag:
UINT): Integer;
```

C++

```
__fastcall int DoGetMenuString(HMENU Menu, UINT ItemID, const char * Str, int MaxCount, UINT
Flag);
```

Description

DoGetMenuString is used internally to obtain menu item captions, given the Windows menu handle. The Menu parameter indicates the menu's handle. The ItemID parameter specifies the menu item whose caption is desired. If the Flag parameter is MF_BYCOMMAND, ItemID is the Command property of the menu item. If Flag is MF_BYPOSITION, ItemID is the 0-based index of the menu item. The caption is returned in a buffer pointed to by the Str parameter. MaxCount indicates the size of that buffer.

Unlike the Windows API function GetMenuString, DoGetMenuString returns the correct caption for owner-draw menu items.

See Also

Caption (see page 4902)

Command (see page 4907)

48.1.3.4.8 TMenu.FindItem Method

Locates and returns a menu item.

Pascal

```
function FindItem(Value: Integer; Kind: TFindItemKind): TMenuItem;
```

C++

```
__fastcall TMenuItem FindItem(int Value, TFindItemKind Kind);
```

Description

Call FindItem to locate a menu item in the menu or one of its submenus. The menu item is identified by either a pop-up menu handle, command ID, or menu shortcut value. The Kind parameter specifies how the menu item is to be identified.

See Also

TMenuItem (see page 4869)

Command (see page 4907)

Handle (see page 4915)

ShortCut (see page 4920)

48.1.3.4.9 TMenu.GetChildren Method

Executes a callback for child menu items.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called automatically by the streaming system so that child menu items are saved and read with their parent menu.

See Also

GetParentComponent

SetChildOrder

48.1.3.4.10 TMenu.GetHandle Method

Returns the value of the Handle (see page 4851) property.

Pascal

```
function GetHandle: HMENU; virtual;
```

C++

```
virtual __fastcall HMENU GetHandle();
```

Description

GetHandle is the protected implementation of the Handle (see page 4851) property. Override this method to change the implementation of the handle property. For example, TMainMenu (see page 4828) overrides GetHandle to adjust for menus added by embedded OLE objects.

See Also

Handle (see page 4851)

48.1.3.4.11 TMenu.GetHelpContext Method

Returns the help context ID associated with a menu item.

Pascal

```
function GetHelpContext(Value: Integer; ByCommand: Boolean): THelpContext;
```

C++

```
__fastcall THelpContext GetHelpContext(int Value, Boolean ByCommand);
```

Description

Call GetHelpContext to obtain the ID for context-sensitive help on a particular menu item. Identify the menu item as the item that matches the Value property. If ByCommand is true, Value is the Command property of the TMenuItem (see page 4869). If ByCommand is false, Value is the Handle (see page 4851) property.

See Also

FindItem (see page 4843)

Command (see page 4907)

Handle (see page 4915)

HelpContext (see page 4915)

HelpContext

48.1.3.4.12 TMenu.IsOwnerDraw Method

Indicates whether menu items can contain more than simple captions.

Pascal

```
function IsOwnerDraw: Boolean;
```

C++

```
__fastcall Boolean IsOwnerDraw();
```

Description

IsOwnerDraw returns true when the OwnerDraw (see page 4856) property is true or the Images (see page 4851) property is not nil (Delphi) or NULL (C++). This is used internally to determine how the menu should be painted.

See Also

OwnerDraw (see page 4856)

Images (see page 4851)

48.1.3.4.13 TMenu.IsRightToLeft Method

Indicates whether menu items read from right to left.

Pascal

```
function IsRightToLeft: Boolean;
```

C++

```
__fastcall Boolean IsRightToLeft();
```

Description

Call IsRightToLeft to determine the layout of main menu items. IsRightToLeft returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode (see page 4850) property is not bdLeftToRight.

See Also

BiDiMode (see page 4850)

48.1.3.4.14 TMenu.IsShortCut Method

Dispatches a key message if it represents a shortcut key for an item in the menu.

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean; dynamic;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

Call IsShortCut when processing a KeyDown message to identify shortcut keys and trigger the appropriate menu action. IsShortCut decodes the key message passed in the Message parameter. It then encodes the state of the Shift, Control, and Alt keys with the value of the character that was typed to obtain the ShortCut (see page 4958) property of a menu item. If IsShortCut locates a menu item associated with the shortcut key combination, it calls the item's Click method and returns true. If no item in the menu has a shortcut corresponding to the message, IsShortCut returns false.

See Also

DispatchCommand (see page 4841)

DispatchPopup (see page 4842)

FindItem (see page 4843)

Click (see page 4881)

48.1.3.4.15 TMenu.Loaded Method

Calls the DoChange (see page 4842) method when the menu is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded is called automatically immediately after the menu is loaded into memory. It calls the DoChange (see page 4842) method to allow any necessary fixup.

See Also

DoChange (see page 4842)

48.1.3.4.16 TMenu.MenuChanged Method

Responds when the menu changes.

Pascal

```
procedure MenuChanged(Sender: TObject; Source: TMenuItem; Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall MenuChanged(TObject * Sender, TMenuItem Source, Boolean Rebuild);
```

Description

MenuChanged is called automatically when changes occur that affect the structure of the menu. The Sender parameter indicates the object that originates the change. It's value can be nil (Delphi) or NULL (C++). Source indicates which menu item is affected. Rebuild indicates whether the menu must be rebuilt to reflect the change (for example when items are added or deleted).

MenuChanged calls the DoChange (see page 4842) method, which generates an OnChange (see page 4858) event.

Note: Changes that affect individual items rather than the menu as a whole trigger the UpdateItems (see page 4849) method instead of MenuChanged. For example, when the Images (see page 4851) property changes, UpdateItems (see page 4849) is called rather than MenuChanged.

See Also

Items (see page 4852)

UpdateItems (see page 4849)

DoChange (see page 4842)

48.1.3.4.17 TMenu.Notification Method

Responds when components are added or deleted.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

TMenu (see page 4837) overrides the inherited method to set the Images (see page 4851) property to nil (Delphi) or NULL (C++) if the ImageList is deleted.

See Also

Notification

Images (see page 4851)

48.1.3.4.18 ParentBiDiModeChanged Method

This is the overview for the ParentBiDiModeChanged method overload.

Overload List

	Name	Description
	TMenu.ParentBiDiModeChanged () (see page 4847)	Ensures that the menu's layout follows the current settings of the BiDiMode (see page 4850) and ParentBiDiMode (see page 4857) properties.
	TMenu.ParentBiDiModeChanged (TObject) (see page 4847)	Ensures that the menu's layout follows the current settings of the BiDiMode (see page 4850) and ParentBiDiMode (see page 4857) properties.

48.1.3.4.18.1 TMenu.ParentBiDiModeChanged Method ()

Ensures that the menu's layout follows the current settings of the BiDiMode (see page 4850) and ParentBiDiMode (see page 4857) properties.

Pascal

```
procedure ParentBiDiModeChanged; overload;
procedure ParentBiDiModeChanged(AControl: TObject); overload;
```

C++

```
__fastcall ParentBiDiModeChanged();
__fastcall ParentBiDiModeChanged(TObject * AControl);
```

Description

ParentBiDiMode (see page 4857) changed is called when the menu's BiDiMode (see page 4850) property must be updated to reflect the bi-directional mode of its associated control.

See Also

BiDiMode (see page 4850)

ParentBiDiMode (see page 4857)

DoBiDiModeChanged (see page 4842)

48.1.3.4.19 TMenu.ProcessMenuChar Method

Responds to keyboard input for the menu.

Pascal

```
procedure ProcessMenuChar(var Message: TWMMenuChar);
```

C++

```
__fastcall ProcessMenuChar(TWMMenuChar Message);
```

Description

ProcessMenuChar handles keyboard messages from Windows. It checks whether the character typed is an accelerator character from one of the menu's items. If so, ProcessMenuChar executes the menu item (or, if the item is disabled, selects it). If the key is not an accelerator, ProcessMenuChar checks for captions that begin with the character. If there is only one, it is treated like an accelerator. Otherwise, the first item beginning with that character is selected.

When calling ProcessMenuChar programmatically, the User, MenuFlag, and Menu fields of Message parameter must be supplied.

See Also

Caption (see page 4902)

48.1.3.4.20 TMenu.SetChildOrder Method

Ensures that the menu item indexes reflect the order in which they are loaded and saved.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

SetChildOrder is called automatically when menus are loaded into memory. The Child parameter specifies the child object that was just loaded. The Order parameter specifies the order in which it was loaded. SetChildOrder assigns the MenuIndex for each child to match the order in which the items are loaded.

See Also

MenuIndex (see page 4919)

48.1.3.4.21 TMenu.UpdateItems Method

Causes all menu items to update themselves.

Pascal

```
procedure UpdateItems;
```

C++

```
__fastcall UpdateItems();
```

Description

UpdateItems is called automatically when changes occur that must be reflected by individual menu items. Changes that affect the menu as a whole trigger the MenuChanged (see page 4846) method instead.

See Also

MenuChanged (see page 4846)

48.1.3.5 TMenu Properties

The properties of the TMenu class are listed here.

Public Properties

	Name	Description
	AutoHotkeys (see page 4849)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4850)	Determines whether redundant separator bars are automatically removed from the menu.
	BiDiMode (see page 4850)	Determines the layout of the menu in Middle East locales.
R	Handle (see page 4851)	Provides access to the Windows menu handle for the menu.
	Images (see page 4851)	Lists the images that can appear beside individual menu items.
	OwnerDraw (see page 4856)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 4857)	Indicates whether the BiDiMode (see page 4850) property is inherited from the control that uses the menu.
	WindowHandle (see page 4857)	Provides access to the handle of the window that uses the menu.

Published Properties

	Name	Description
R	Items (see page 4852)	Describes the elements of the menu.

48.1.3.5.1 TMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu (see page 4837)

tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items (see page 4852) property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its Items (see page 4852) property. Setting one sets the other.

See Also

AutoHotkeys (see page 4900)

Items (see page 4852)

RethinkHotkeys (see page 4894)

48.1.3.5.2 TMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items (see page 4852) property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items (see page 4852) property. Setting one sets the other.

See Also

AutoLineReduction (see page 4900)

Items (see page 4852)

RethinkLines (see page 4894)

48.1.3.5.3 TMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 4857)

IsRightToLeft (see page 4845)

48.1.3.5.4 TMenu.Handle Property

Provides access to the Windows menu handle for the menu.

Pascal

```
property Handle: HMENU;
```

C++

```
__property HMENU Handle;
```

Description

Use Handle to call a Windows API function that requires a menu handle. Handle is the handle for the menu that contains only the items described by the Items (see page 4852) property. If no other menus have been merged with this menu, this is the same as the WindowHandle (see page 4857) property.

See Also

WindowHandle (see page 4857)

48.1.3.5.5 TMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 4916)

Images

Items (see page 4852)

48.1.3.5.6 TMenu.Items Property

Describes the elements of the menu.

Pascal

```
property Items: TMenuItem;
```

C++

```
__property TMenuItem Items;
```

Description

Use Items to access information about the elements in the menu. Items is a single TMenuItem (see page 4869) object that describes the elements of the menu in its own Items property.

Note: Because Items is the default property of TMenuItem (see page 4869), the Items property of TMenu (see page 4837) can be treated as an indexed array of menu items that describe the individual items in the menu. That is, instead of writing

Note: FirstItem := Menu1.Items.Items[0];

Note: you can use

Note: FirstItem := Menu1.Items[0];

Set the Items property at design time by clicking on the Items property in the Property Inspector. The Menu Designer automatically inserts a menu item. As each menu item is entered, outlined regions appear at the possible locations of additional menu items. Click in those regions to specify additional menu items. At runtime, create a TMenuItem (see page 4869) that describes the elements of the menu programmatically, and set the Items property of the menu.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items->Items[1]->Add(NewItem);
```

```
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
```

```
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
```

```
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

TMenuItem (see page 4869)

Items (see page 4917)

48.1.3.5.7 TMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images (see page 4851) and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 4929)

OnDrawItem (see page 4928)

Images (see page 4851)

48.1.3.5.8 TMenu.ParentBiDiMode Property

Indicates whether the BiDiMode (see page 4850) property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode (see page 4850) setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode (see page 4850) setting and control the menu's layout independently.

See Also

BiDiMode (see page 4850)

48.1.3.5.9 TMenu.WindowHandle Property

Provides access to the handle of the window that uses the menu.

Pascal

```
property WindowHandle: HWND;
```

C++

```
__property HWND WindowHandle;
```

Description

Read WindowHandle to get the window handle of the control that receives WM_COMMAND (see page 5148) messages when the user clicks on an item in the menu. When menu items are selected, use this handle in an OnClick event handler to send messages to the control that uses the menu or to make Windows API calls that affect the control.

Set WindowHandle to associate the menu with a control.

See Also

DispatchCommand (see page 4841)

Command (see page 4907)

OnClick (see page 4922)

48.1.3.6 TMenu Events

The events of the TMenu class are listed here.

Protected Events

	Name	Description
	OnChange (see page 4858)	Occurs when the menu changes.

48.1.3.6.1 TMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

See Also

Items (see page 4852)

48.1.4 TMenuActionLink Class

TMenuActionLink links menu item clients to the corresponding menu action.

Class Hierarchy

Pascal

```
TMenuActionLink = class(TActionLink);
```

C++

```
class TMenuActionLink : public TActionLink;
```

File

Menus

Description

TMenuActionLink is designed to link actions to menu item clients. TMenuActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of and events of menu item clients.

TMenuActionLink overrides methods to support linking the following properties and events with menu item client properties or events, most of which have the same name:

Caption

Checked

Enabled

HelpContext

Hint

ImageIndex

ShortCut (see page 4958)

Visible

OnExecute

48.1.4.1 TMenuActionLink Members

The following tables list the members exposed by TMenuActionLink.

Protected Fields

	Name	Description
	FClient (see page 4860)	This is FClient, a member of class TMenuActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 4860)	Assigns a client menu item that is linked to the associated action.
	IsAutoCheckLinked (see page 4861)	Indicates whether the AutoCheck state for the action and the client are linked.
	IsCaptionLinked (see page 4861)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 4862)	Indicates whether the checked state for the action and the client are linked.
	IsEnabledLinked (see page 4862)	Indicates whether the enabled state for the action and the client are linked.
	IsGroupIndexLinked (see page 4862)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 4863)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHintLinked (see page 4863)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 4864)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 4864)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsShortCutLinked (see page 4864)	Indicates whether the ShortCut (see page 4958) property for the action and the client are linked.
	IsVisibleLinked (see page 4865)	Indicates whether the visible state for the action and the client are linked.
	SetAutoCheck (see page 4865)	Sets the AutoCheck property of a menu item if the client is linked to the action for this property.
	SetCaption (see page 4866)	Sets the caption of a menu item if the client is linked to the action for this property.
	SetChecked (see page 4866)	Sets the checked state of a menu item if the client is linked to the action for this property.
	SetEnabled (see page 4866)	Sets the enabled state of a menu item if the client is linked to the action for this property.
	SetHelpContext (see page 4867)	Sets the Help context of a menu item that is the client of this action link.
	SetHint (see page 4867)	Sets the hint of a menu item if the client is linked to the action for this property.
	SetImageIndex (see page 4867)	Sets the image index of a menu item if the client is linked to the action for this property.
	SetOnExecute (see page 4868)	Sets the execution event handler for a menu item if the client is linked to the action for this property.
	SetShortCut (see page 4868)	Sets the shortcut of a menu item if the client is linked to the action for this property.
	SetVisible (see page 4869)	Sets the visible state of a menu item if the client is linked to the action for this property.

48.1.4.2 TMenuActionLink Fields

The fields of the TMenuActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 4860)	This is FClient, a member of class TMenuActionLink.

48.1.4.2.1 TMenuActionLink.FClient Field

Pascal

```
FClient: TMenuItem;
```

C++

```
TMenuItem FClient;
```

Description

This is FClient, a member of class TMenuActionLink.

48.1.4.3 TMenuActionLink Methods

The methods of the TMenuActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 4860)	Assigns a client menu item that is linked to the associated action.
	IsAutoCheckLinked (see page 4861)	Indicates whether the AutoCheck state for the action and the client are linked.
	IsCaptionLinked (see page 4861)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 4862)	Indicates whether the checked state for the action and the client are linked.
	IsEnabledLinked (see page 4862)	Indicates whether the enabled state for the action and the client are linked.
	IsGroupIndexLinked (see page 4862)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 4863)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHintLinked (see page 4863)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 4864)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 4864)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsShortCutLinked (see page 4864)	Indicates whether the ShortCut (see page 4958) property for the action and the client are linked.
	IsVisibleLinked (see page 4865)	Indicates whether the visible state for the action and the client are linked.
	SetAutoCheck (see page 4865)	Sets the AutoCheck property of a menu item if the client is linked to the action for this property.
	SetCaption (see page 4866)	Sets the caption of a menu item if the client is linked to the action for this property.
	SetChecked (see page 4866)	Sets the checked state of a menu item if the client is linked to the action for this property.
	SetEnabled (see page 4866)	Sets the enabled state of a menu item if the client is linked to the action for this property.
	SetHelpContext (see page 4867)	Sets the Help context of a menu item that is the client of this action link.
	SetHint (see page 4867)	Sets the hint of a menu item if the client is linked to the action for this property.
	SetImageIndex (see page 4867)	Sets the image index of a menu item if the client is linked to the action for this property.
	SetOnExecute (see page 4868)	Sets the execution event handler for a menu item if the client is linked to the action for this property.
	SetShortCut (see page 4868)	Sets the shortcut of a menu item if the client is linked to the action for this property.
	SetVisible (see page 4869)	Sets the visible state of a menu item if the client is linked to the action for this property.

48.1.4.3.1 TMenuActionLink.AssignClient Method

Assigns a client menu item that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TMenuActionLink (see page 4858) overrides AssignClient to ensure that the client for the link is a menu item.

See Also

Action (see page 4899)

Create

48.1.4.3.2 TMenuActionLink.IsAutoCheckLinked Method

Indicates whether the AutoCheck state for the action and the client are linked.

Pascal

```
function IsAutoCheckLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsAutoCheckLinked();
```

Description

IsAutoCheckLinked indicates whether the AutoCheck property is linked to the appropriate client property. IsAutoCheckLinked returns true if the client requests that the action control this property, false otherwise.

See Also

AutoCheck

AutoCheck (see page 4899)

SetAutoCheck (see page 4865)

48.1.4.3.3 TMenuActionLink.IsCaptionLinked Method

Indicates whether the caption for the action and the client are linked.

Pascal

```
function IsCaptionLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

IsCaptionLinked indicates whether the Caption property is linked to the appropriate client property. IsCaptionLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Caption

Caption (see page 4902)

SetCaption (see page 4866)

48.1.4.3.4 TMenuActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked

Checked (see page 4906)

SetChecked (see page 4866)

48.1.4.3.5 TMenuActionLink.IsEnabledLinked Method

Indicates whether the enabled state for the action and the client are linked.

Pascal

```
function IsEnabledLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsEnabledLinked();
```

Description

IsEnabledLinked indicates whether the Enabled property is linked to the appropriate client property. IsEnabledLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Enabled

Enabled (see page 4912)

SetEnabled (see page 4866)

48.1.4.3.6 TMenuActionLink.IsGroupIndexLinked Method

Indicates whether the GroupIndex property for the action controls client behavior.

Pascal

```
function IsGroupIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsGroupIndexLinked();
```

Description

TActionLink descendants can call IsGroupIndexLinked to determine whether they need to propagate changes to the GroupIndex property of the action so that they are reflected by the client object. When IsGroupIndexLinked returns true, the action controls

the GroupIndex property: Changes to the GroupIndex property of the action are reflected in the client object. When IsGroupIndexLinked returns false, the client object is not influenced by the GroupIndex property of its action.

As implemented in TActionLink, IsGroupIndexLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

GroupIndex

SetGroupIndex

Action

48.1.4.3.7 TMenuActionLink.IsHelpContextLinked Method

Indicates whether the HelpContext properties for the action and client controls are linked.

Pascal

```
function IsHelpContextLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHelpContextLinked();
```

Description

TActionLink descendants can call IsHelpContextLinked to determine whether they need to propagate changes to the HelpContext property of the action so that they are reflected by the client object. When IsHelpContextLinked returns true, the action controls the HelpContext property: Changes to the HelpContext property of the action are reflected in the client object. When IsHelpContextLinked returns false, the client object is not influenced by the HelpContext property of its action.

As implemented in TActionLink, IsHelpContextLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

HelpContext

SetHelpContext

Action

IsHelpLinked

48.1.4.3.8 TMenuActionLink.IsHintLinked Method

Indicates whether the Hint property for the action and the client are linked.

Pascal

```
function IsHintLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

IsHintLinked indicates whether the Hint property is linked to the appropriate client property. IsHintLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Hint

Hint (see page 4916)

SetHint (see page 4867)

48.1.4.3.9 TMenuActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

IsImageIndexLinked indicates whether the ImageIndex property is linked to the appropriate client property. IsImageIndexLinked returns true if the client requests that the action control this property, false otherwise.

See Also

ImageIndex

ImageIndex (see page 4916)

SetImageIndex (see page 4867)

48.1.4.3.10 TMenuActionLink.IsOnExecuteLinked Method

Indicates whether the OnExecute event handler for the action and the client are linked.

Pascal

```
function IsOnExecuteLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

IsOnExecuteLinked indicates whether the OnExecute event handler is linked to the appropriate client event. IsOnExecuteLinked returns true if the client requests that the action control this event, false otherwise.

See Also

OnExecute

OnClick (see page 4922)

SetOnExecute (see page 4868)

48.1.4.3.11 TMenuActionLink.IsShortCutLinked Method

Indicates whether the ShortCut (see page 4958) property for the action and the client are linked.

Pascal

```
function IsShortCutLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsShortCutLinked();
```

Description

IsShortCutLinked indicates whether the ShortCut (see page 4958) property is linked to the appropriate client property. IsVisibleLinked (see page 4865) returns true if the client requests that the action control this property, false otherwise.

See Also

ShortCut

ShortCut (see page 4920)

SetShortCut (see page 4868)

48.1.4.3.12 TMenuActionLink.IsVisibleLinked Method

Indicates whether the visible state for the action and the client are linked.

Pascal

```
function IsVisibleLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVisibleLinked();
```

Description

IsVisibleLinked indicates whether the Visible property is linked to the appropriate client property. IsVisibleLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Visible

Visible (see page 4921)

SetVisible (see page 4869)

48.1.4.3.13 TMenuActionLink.SetAutoCheck Method

Sets the AutoCheck property of a menu item if the client is linked to the action for this property.

Pascal

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

C++

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

Description

SetAutoCheck sets the AutoCheck property of a menu item, and is called only if the client is linked for this event. Value specifies the AutoCheck new value of AutoCheck.

See Also

AutoCheck

AutoCheck (see page 4899)

IsAutoCheckLinked (see page 4861)

48.1.4.3.14 TMenuActionLink.SetCaption Method

Sets the caption of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const UnicodeString Value);
```

Description

SetCaption sets the caption of a menu item, and is called only if the client is linked for this event. Value is the string to which the caption should be set.

See Also

Caption

Caption (see page 4902)

IsCaptionLinked (see page 4861)

48.1.4.3.15 TMenuActionLink.SetChecked Method

Sets the checked state of a menu item if the client is linked to the action for this property.

Pascal

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

C++

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

Description

SetChecked sets the checked state of a menu item, and is called only if the client is linked for this event. Value specifies whether the checked state should be set.

See Also

Checked

Checked (see page 4906)

IsCheckedLinked (see page 4862)

48.1.4.3.16 TMenuActionLink.SetEnabled Method

Sets the enabled state of a menu item if the client is linked to the action for this property.

Pascal

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

C++

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

Description

SetEnabled sets the enabled state of a menu item, and is called only if the client is linked for this event. Value specifies whether the enabled state should be set.

See Also

Enabled

Enabled (see page 4912)

IsEnabledLinked (see page 4862)

48.1.4.3.17 TMenuActionLink.SetHelpContext Method

Sets the Help context of a menu item that is the client of this action link.

Pascal

```
procedure SetHelpContext(Value: THelpContext); override;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

SetHelpContext sets the help context of a client menu item, and is called only if the client is linked for this property. Value specifies what the help context should be.

See Also

HelpContext

HelpContext (see page 4915)

IsHelpLinked

48.1.4.3.18 TMenuActionLink.SetHint Method

Sets the hint of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetHint(const Value: string); override;
```

C++

```
virtual __fastcall SetHint(const UnicodeString Value);
```

Description

SetHint sets the hint of a menu item, and is called only if the client is linked for this event. Value is the string that to which the hint should be set.

See Also

Hint

Hint (see page 4916)

IsHintLinked (see page 4863)

48.1.4.3.19 TMenuActionLink.SetImageIndex Method

Sets the image index of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

SetImageIndex sets the image index of a menu item, and is called only if the client is linked for this event. Value specifies what the image index should be.

See Also

ImageIndex

ImageIndex (see page 4916)

IsImageIndexLinked (see page 4864)

48.1.4.3.20 TMenuActionLink.SetOnExecute Method

Sets the execution event handler for a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); override;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

SetOnExecute sets the appropriate execution event handler of a menu item, and is called only if the client is linked for this event. Value specifies the procedure that should be used for this event.

See Also

OnExecute

OnClick (see page 4922)

IsOnExecuteLinked (see page 4864)

48.1.4.3.21 TMenuActionLink.SetShortCut Method

Sets the shortcut of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetShortCut(Value: TShortCut); override;
```

C++

```
virtual __fastcall SetShortCut(TShortCut Value);
```

Description

SetShortCut sets the shortcut of a menu item, and is called only if the client is linked for this event. Value specifies what the shortcut should be.

See Also

ShortCut

ShortCut (see page 4920)

IsShortCutLinked (see page 4864)

48.1.4.3.22 TMenuActionLink.SetVisible Method

Sets the visible state of a menu item if the client is linked to the action for this property.

Pascal

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

C++

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

Description

SetVisible sets the visible state of a menu item, and is called only if the client is linked for this event. Value specifies whether the visible state should be set.

See Also

Visible

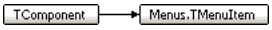
Visible (see page 4921)

IsVisibleLinked (see page 4865)

48.1.5 TMenuItem Class

TMenuItem describes the properties of an item in a menu.

Class Hierarchy



Pascal

```
TMenuItem = class(TComponent);
```

C++

```
class TMenuItem : public TComponent;
```

File

Menus

Description

Use TMenuItem to specify the appearance and behavior of an item in a menu. Each TMainMenu (see page 4828) or TPopupMenu (see page 4935) component can contain multiple menu items. When a menu is defined at design time, the Menu Designer automatically creates menu item objects for each command on the menu.

See Also

TMainMenu (see page 4828)

TPopupMenu (see page 4935)

NewItem (see page 4953)

NewLine (see page 4954)

NewSubMenu (see page 4957)

48.1.5.1 TMenuItem Members

The following tables list the members exposed by TMenuItem.

Protected Methods

	Name	Description
	ActionChange (see page 4875)	This is ActionChange, a member of class TMenuItem.
V	AdvancedDrawItem (see page 4880)	This is AdvancedDrawItem, a member of class TMenuItem.
V	AssignTo (see page 4880)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 4881)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoDrawText (see page 4882)	This is DoDrawText, a member of class TMenuItem.
V	DrawItem (see page 4883)	This is DrawItem, a member of class TMenuItem.
	GetActionLinkClass (see page 4883)	This is GetActionLinkClass, a member of class TMenuItem.
	GetAutoHotkeys (see page 4884)	This is GetAutoHotkeys, a member of class TMenuItem.
	GetAutoLineReduction (see page 4884)	This is GetAutoLineReduction, a member of class TMenuItem.
	GetCount (see page 4884)	This is GetCount, a member of class TMenuItem.
	GetHandle (see page 4885)	This is GetHandle, a member of class TMenuItem.
	GetItem (see page 4886)	This is GetItem, a member of class TMenuItem.
	GetMenuIndex (see page 4886)	This is GetMenuIndex, a member of class TMenuItem.
	InsertNewLine (see page 4889)	This is InsertNewLine, a member of class TMenuItem.
V	Loaded (see page 4891)	Initializes the component after the form file has been read into memory.
V	MeasureItem (see page 4891)	This is MeasureItem, a member of class TMenuItem.
V	MenuChanged (see page 4892)	This is MenuChanged, a member of class TMenuItem.
V	Notification (see page 4893)	Forwards notification messages to all owned components.
	SetBreak (see page 4895)	This is SetBreak, a member of class TMenuItem.
	SetCaption (see page 4895)	This is SetCaption, a member of class TMenuItem.
	SetChecked (see page 4895)	This is SetChecked, a member of class TMenuItem.
V	SetChildOrder (see page 4896)	This is SetChildOrder, a member of class TMenuItem.
	SetDefault (see page 4896)	This is SetDefault, a member of class TMenuItem.
	SetEnabled (see page 4896)	This is SetEnabled, a member of class TMenuItem.
	SetGroupIndex (see page 4896)	This is SetGroupIndex, a member of class TMenuItem.
	SetImageIndex (see page 4897)	This is SetImageIndex, a member of class TMenuItem.
	SetMenuIndex (see page 4897)	This is SetMenuIndex, a member of class TMenuItem.
	SetRadioItem (see page 4897)	This is SetRadioItem, a member of class TMenuItem.
	SetShortCut (see page 4897)	This is SetShortCut, a member of class TMenuItem.
	SetVisible (see page 4898)	This is SetVisible, a member of class TMenuItem.
	UpdateItems (see page 4898)	This is UpdateItems, a member of class TMenuItem.

Protected Properties

	Name	Description
	ActionLink (see page 4899)	This is ActionLink, a member of class TMenuItem.

Public Methods

	Name	Description
	Add (see page 4875)	This is the overview for the Add method overload.
	Clear (see page 4881)	Removes and frees all menu items listed in the Items (see page 4917) property.
V	Click (see page 4881)	Simulates a mouse click.
V	Create (see page 4872)	Creates an instance of TMenuItem (see page 4869).
	Delete (see page 4882)	Removes a menu item from the Items (see page 4917) property array.
V	Destroy (see page 4873)	Destroys an instance of TMenuItem (see page 4869).
	Find (see page 4883)	Locates a menu item in the Items (see page 4917) property array given its caption.

	GetChildren (see page 4884)	This is GetChildren, a member of class TMenuItem.
	GetEnumerator (see page 4884)	Returns a TComponent enumerator.
	GetImageList (see page 4885)	Returns the Image list that supplies an image for the menu item.
	GetParentComponent (see page 4886)	Returns the parent component of the menu item.
	GetParentMenu (see page 4886)	Returns the main menu of which the menu item is part.
	HasParent (see page 4887)	Indicates that the menu item has a parent that is responsible for loading and saving its data.
	IndexOf (see page 4887)	Returns the position of a menu item within the Items (see page 4917) array.
	InitiateAction (see page 4887)	Calls the action link's Update method if the menu item is associated with an action link.
	Insert (see page 4888)	Inserts a menu item into a specified position in the Items (see page 4917) array.
	InsertNewLineAfter (see page 4889)	Inserts a separator bar after a specified item from the Items (see page 4917) property array.
	InsertNewLineBefore (see page 4890)	Inserts a separator bar before a specified item from the Items (see page 4917) property array.
	IsLine (see page 4890)	Indicates whether the menu item represents a separator bar.
	NewBottomLine (see page 4892)	Inserts a separator bar at the end of the Items (see page 4917) property array.
	NewTopLine (see page 4892)	Inserts a separator bar at the beginning of the Items (see page 4917) property array.
	Remove (see page 4893)	Removes a menu item from the Items (see page 4917) property array
	RethinkHotkeys (see page 4894)	Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.
	RethinkLines (see page 4894)	Removes superfluous separator lines from the Items (see page 4917) property array.
	SetParentComponent (see page 4897)	This is SetParentComponent, a member of class TMenuItem.

Public Properties

	Name	Description
	Command (see page 4907)	Specifies the Windows Command ID associated with the menu item.
	Count (see page 4907)	Indicates the number of subitems of the menu item.
	Handle (see page 4915)	Indicates the Windows menu handle of the drop-down menu associated with the menu item.
	Items (see page 4917)	Lists the menu items in the submenu of the menu item.
	MenuIndex (see page 4919)	Indicates the index of the menu item within its parent menu.
	Parent (see page 4919)	Identifies the parent menu item of this menu item.

Published Events

	Name	Description
	OnAdvancedDrawItem (see page 4922)	Occurs when an owner-draw menu needs to be drawn.
	OnClick (see page 4922)	Occurs when the user clicks menu item.
	OnDrawItem (see page 4928)	Occurs when an owner-draw menu needs to be drawn.
	OnMeasureItem (see page 4929)	Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

Published Properties

	Name	Description
	Action (see page 4899)	Designates the action associated with the menu item.
	AutoCheck (see page 4899)	Indicates whether the menu item's checked state toggles automatically when the item is clicked.
	AutoHotkeys (see page 4900)	Determines whether the accelerator keys for submenu items can be reset automatically.
	AutoLineReduction (see page 4900)	Determines whether redundant separator bars are automatically removed from the submenu.
	Bitmap (see page 4901)	Designates a bitmap that is drawn for the menu item.
	Break (see page 4901)	Determines whether the menu item starts a new column in the menu.
	Caption (see page 4902)	Specifies the text of the menu item.
	Checked (see page 4906)	Specifies whether a check mark should appear beside the Caption (see page 4902).

	Default (see page 4912)	Specifies whether the menu item is invoked when the parent item is double clicked.
	Enabled (see page 4912)	Specifies whether the menu item is enabled.
	GroupIndex (see page 4914)	Identifies the logical group to which the menu item belongs.
	HelpContext (see page 4915)	Specifies the help context ID associated with the menu item.
	Hint (see page 4916)	Specifies the text string that can appear when the user moves the mouse pointer over a menu item.
	ImageIndex (see page 4916)	Indicates which image maintained by the parent menu appears next to the menu item.
	RadioItem (see page 4920)	Specifies whether the menu item is mutually exclusive with other menu items in its group.
	ShortCut (see page 4920)	Specifies the key combination users can type to quickly access the menu item.
	SubMenuImages (see page 4921)	Lists the images that can appear beside individual items in a submenu of this menu item.
	Visible (see page 4921)	Specifies whether the menu item appears in the menu.

48.1.5.2 TMenuItem.Create Constructor

Creates an instance of TMenuItem (see page 4869).

Pascal

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

C++

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

Description

Use Create to create a menu item at runtime. Menu items defined at design time using the Menu designer are created automatically.

AOwner is the component that is responsible for freeing the menu item. It becomes the value of the Owner property.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items->Items[1]->Add(NewItem);
  }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
```

See Also

Owner

NewItem (see page 4953)

48.1.5.3 TMenuItem.Destroy Destructor

Destroys an instance of TMenuItem (see page 4869).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMenuItem();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMenuItem (see page 4869) reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

48.1.5.4 TMenuItem Methods

The methods of the TMenuItem class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 4875)	This is ActionChange, a member of class TMenuItem.
V	AdvancedDrawItem (see page 4880)	This is AdvancedDrawItem, a member of class TMenuItem.
V	AssignTo (see page 4880)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 4881)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoDrawText (see page 4882)	This is DoDrawText, a member of class TMenuItem.
V	DrawItem (see page 4883)	This is DrawItem, a member of class TMenuItem.
	GetActionLinkClass (see page 4883)	This is GetActionLinkClass, a member of class TMenuItem.
	GetAutoHotkeys (see page 4884)	This is GetAutoHotkeys, a member of class TMenuItem.
	GetAutoLineReduction (see page 4884)	This is GetAutoLineReduction, a member of class TMenuItem.
	GetCount (see page 4884)	This is GetCount, a member of class TMenuItem.
	GetHandle (see page 4885)	This is GetHandle, a member of class TMenuItem.
	GetItem (see page 4886)	This is GetItem, a member of class TMenuItem.
	GetMenuIndex (see page 4886)	This is GetMenuIndex, a member of class TMenuItem.
	InsertNewLine (see page 4889)	This is InsertNewLine, a member of class TMenuItem.
V	Loaded (see page 4891)	Initializes the component after the form file has been read into memory.
V	MeasureItem (see page 4891)	This is MeasureItem, a member of class TMenuItem.
V	MenuChanged (see page 4892)	This is MenuChanged, a member of class TMenuItem.
V	Notification (see page 4893)	Forwards notification messages to all owned components.
	SetBreak (see page 4895)	This is SetBreak, a member of class TMenuItem.
	SetCaption (see page 4895)	This is SetCaption, a member of class TMenuItem.
	SetChecked (see page 4895)	This is SetChecked, a member of class TMenuItem.
V	SetChildOrder (see page 4896)	This is SetChildOrder, a member of class TMenuItem.
	SetDefault (see page 4896)	This is SetDefault, a member of class TMenuItem.
	SetEnabled (see page 4896)	This is SetEnabled, a member of class TMenuItem.
	SetGroupIndex (see page 4896)	This is SetGroupIndex, a member of class TMenuItem.
	SetImageIndex (see page 4897)	This is SetImageIndex, a member of class TMenuItem.
	SetMenuIndex (see page 4897)	This is SetMenuIndex, a member of class TMenuItem.
	SetRadioItem (see page 4897)	This is SetRadioItem, a member of class TMenuItem.
	SetShortCut (see page 4897)	This is SetShortCut, a member of class TMenuItem.
	SetVisible (see page 4898)	This is SetVisible, a member of class TMenuItem.
	UpdateItems (see page 4898)	This is UpdateItems, a member of class TMenuItem.

Public Methods

	Name	Description
	Add (see page 4875)	This is the overview for the Add method overload.
	Clear (see page 4881)	Removes and frees all menu items listed in the Items (see page 4917) property.
V	Click (see page 4881)	Simulates a mouse click.
	Delete (see page 4882)	Removes a menu item from the Items (see page 4917) property array.
	Find (see page 4883)	Locates a menu item in the Items (see page 4917) property array given its caption.
V	GetChildren (see page 4884)	This is GetChildren, a member of class TMenuItem.
	GetEnumerator (see page 4884)	Returns a TComponent enumerator.
	GetImageList (see page 4885)	Returns the Image list that supplies an image for the menu item.
V	GetParentComponent (see page 4886)	Returns the parent component of the menu item.
	GetParentMenu (see page 4886)	Returns the main menu of which the menu item is part.

	HasParent (see page 4887)	Indicates that the menu item has a parent that is responsible for loading and saving its data.
	IndexOf (see page 4887)	Returns the position of a menu item within the Items (see page 4917) array.
	InitiateAction (see page 4887)	Calls the action link's Update method if the menu item is associated with an action link.
	Insert (see page 4888)	Inserts a menu item into a specified position in the Items (see page 4917) array.
	InsertNewLineAfter (see page 4889)	Inserts a separator bar after a specified item from the Items (see page 4917) property array.
	InsertNewLineBefore (see page 4890)	Inserts a separator bar before a specified item from the Items (see page 4917) property array.
	IsLine (see page 4890)	Indicates whether the menu item represents a separator bar.
	NewBottomLine (see page 4892)	Inserts a separator bar at the end of the Items (see page 4917) property array.
	NewTopLine (see page 4892)	Inserts a separator bar at the beginning of the Items (see page 4917) property array.
	Remove (see page 4893)	Removes a menu item from the Items (see page 4917) property array
	RethinkHotkeys (see page 4894)	Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.
	RethinkLines (see page 4894)	Removes superfluous separator lines from the Items (see page 4917) property array.
	SetParentComponent (see page 4897)	This is SetParentComponent, a member of class TMenuItem.

48.1.5.4.1 TMenuItem.ActionChange Method

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); dynamic;
```

C++

```
__fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

This is ActionChange, a member of class TMenuItem.

48.1.5.4.2 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TMenuItem.Add (array of TMenuItem) (see page 4875)	Adds one or more menu items to the end of the Items (see page 4917) property array.
	TMenuItem.Add (TMenuItem) (see page 4875)	Adds one or more menu items to the end of the Items (see page 4917) property array.

48.1.5.4.2.1 TMenuItem.Add Method (TMenuItem)

Adds one or more menu items to the end of the Items (see page 4917) property array.

Pascal

```
procedure Add(Item: TMenuItem); overload;
procedure Add(const AItems: array of TMenuItem); overload;
```

C++

```
__fastcall Add(TMenuItem Item);
__fastcall Add(const array of TMenuItem AItems);
```

Description

Use Add to add new menu items to the dropdown menu for this menu item. If this menu item is the value of the Items (see page 4917) property of a main menu or pop-up menu, Add adds the items to the end of the menu.

Specify a single new menu item to add as the value of the Item parameter, or an array of new menu items as the value of the AItems parameter. AItems_Size is the index of the last item in the AItems array (one less than the number of items).

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items->Items[1]->Add(NewItem);
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
  var
    NewItem: TMenuItem;
    I : integer;
  begin
    { first create the separator }
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Name:= 'Separator';
    NewItem.Caption := '-';
    { add the new item to the Windows menu }
    MainMenu1.Items[1].Add(NewItem);
    { now create and add a menu item for each form }
    for  I := 0 to Screen.FormCount-1 do
    begin
      NewItem := TMenuItem.Create(MainMenu1.Items[1]);
      NewItem.Caption := Screen.Forms[I].Name + 'Item';
      NewItem.Name := Screen.Forms[I].Name;
      MainMenu1.Items[1].Add(NewItem);
     end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I : Integer;
  begin
    // Count down, not up!
    for  I := MainMenu1.Items[1].Count - 1 downto 0 do
    begin
      MainMenu1.Items[1].Items[I].Free;
    end;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}
```

```
procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

Delete (see page 4882)

Insert (see page 4888)

Items (see page 4917)

Remove (see page 4893)

Items (see page 4852)

NewItem (see page 4953)

NewSubMenu (see page 4957)

48.1.5.4.3 TMenuItem.AdvancedDrawItem Method

Pascal

```
procedure AdvancedDrawItem(ACanvas: TCanvas; ARect: TRect; State: TOwnerDrawState; TopLevel:
Boolean); virtual;
```

C++

```
virtual __fastcall AdvancedDrawItem(TCanvas ACanvas, TRect ARect, TOwnerDrawState State,
Boolean TopLevel);
```

Description

This is AdvancedDrawItem, a member of class TMenuItem.

48.1.5.4.4 TMenuItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

48.1.5.4.5 TMenuItem.Clear Method

Removes and frees all menu items listed in the Items (see page 4917) property.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to free all the items listed in the Items (see page 4917) property array.

See Also

Items (see page 4917)

Delete (see page 4882)

48.1.5.4.6 TMenuItem.Click Method

Simulates a mouse click.

Pascal

```
procedure Click; virtual;
```

C++

```
virtual __fastcall Click();
```

Description

Click generates an OnClick (see page 4922) event, as if the user had clicked the menu item. Unlike a real mouse click, Click does not generate a WM_COMMAND (see page 5148) message to the window associated with the menu.

Component writers can override click to provide a different response when the user selects the menu item.

See Also

Command (see page 4907)

OnClick (see page 4922)

48.1.5.4.7 TMenuItem.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor

DefineBinaryProperty

DefineProperties

DefineProperty

48.1.5.4.8 TMenuItem.Delete Method

Removes a menu item from the Items (see page 4917) property array.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to remove a menu item and all its submenus from a menu. The Index parameter is the index of the item to be deleted, where 0 specifies the first item in the menu, 1 specifies the second item, and so on.

Delete does not free the removed item's memory. To remove an item from the menu and free its memory, call the Free method in the item you want to remove.

To temporarily hide a menu item, without deleting it, use the Visible (see page 4921) property.

See Also

Add (see page 4875)

Insert (see page 4888)

Items (see page 4917)

Remove (see page 4893)

Visible (see page 4921)

System::TObject::Free

48.1.5.4.9 TMenuItem.DoDrawText Method

Pascal

```
procedure DoDrawText(ACanvas: TCanvas; const ACaption: string; var Rect: TRect; Selected:
Boolean; Flags: Longint);
```

C++

```
__fastcall DoDrawText(TCanvas ACanvas, const UnicodeString ACaption, TRect Rect, Boolean
Selected, Longint Flags);
```

Description

This is DoDrawText, a member of class TMenuItem.

48.1.5.4.10 TMenuItem.DrawItem Method

Pascal

```
procedure DrawItem(ACanvas: TCanvas; ARect: TRect; Selected: Boolean); virtual;
```

C++

```
virtual __fastcall DrawItem(TCanvas ACanvas, TRect ARect, Boolean Selected);
```

Description

This is DrawItem, a member of class TMenuItem.

48.1.5.4.11 TMenuItem.Find Method

Locates a menu item in the Items (see page 4917) property array given its caption.

Pascal

```
function Find(ACaption: string): TMenuItem;
```

C++

```
__fastcall TMenuItem Find(UnicodeString ACaption);
```

Description

Use Find to locate the menu item in the Items (see page 4917) property array with the caption specified by the ACaption parameter. Find ignores accelerator characters when comparing item captions to the ACaption parameter.

Find returns the first menu item in the Items (see page 4917) property array with the specified caption. If no item is found, Find returns nil (Delphi) or NULL (C++).

See Also

Items (see page 4917)

Caption (see page 4902)

48.1.5.4.12 TMenuItem.GetActionLinkClass Method

Pascal

```
function GetActionLinkClass: TMenuActionLinkClass; dynamic;
```

C++

```
__fastcall TMenuActionLinkClass GetActionLinkClass();
```

Description

This is GetActionLinkClass, a member of class TMenuItem.

48.1.5.4.13 TMenuItem.GetAutoHotkeys Method

Pascal

```
function GetAutoHotkeys: Boolean;
```

C++

```
__fastcall Boolean GetAutoHotkeys();
```

Description

This is GetAutoHotkeys, a member of class TMenuItem.

48.1.5.4.14 TMenuItem.GetAutoLineReduction Method

Pascal

```
function GetAutoLineReduction: Boolean;
```

C++

```
__fastcall Boolean GetAutoLineReduction();
```

Description

This is GetAutoLineReduction, a member of class TMenuItem.

48.1.5.4.15 TMenuItem.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TMenuItem.

48.1.5.4.16 TMenuItem.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TMenuItem.

48.1.5.4.17 TMenuItem.GetEnumerator Method

Returns a TComponent enumerator.

Pascal

```
function GetEnumerator: TMenuItemEnumerator;
```

C++

```
__fastcall TMenuItemEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TComponentEnumerator reference, which enumerates the components contained within a specified containing component.

To process all these sub components, call the TComponentEnumerator GetCurrent method within a While MoveNext do loop.

See Also

TComponentEnumerator

48.1.5.4.18 TMenuItem.GetHandle Method

Pascal

```
function GetHandle: HMENU;
```

C++

```
__fastcall HMENU GetHandle();
```

Description

This is GetHandle, a member of class TMenuItem.

48.1.5.4.19 TMenuItem.GetImageList Method

Returns the Image list that supplies an image for the menu item.

Pascal

```
function GetImageList: TCustomImageList;
```

C++

```
__fastcall TCustomImageList GetImageList();
```

Description

Use GetImageList to access the list of images that contains the image for this menu item. The ImageIndex (see page 4916) property identifies a specific image from the list that GetImageList returns.

GetImageList first checks the Parent (see page 4919) of this menu item for a SubMenuImages (see page 4921) property. If the parent menu item does not maintain a list of images, it checks that menu item's parent, and so on until reaching a top-level menu item (a menu item where Parent (see page 4919) is not set). If none of the parent menu items has a SubMenuImages (see page 4921) property, GetImageList checks the top-level menu for an Images property.

GetImageList returns the first image list it finds. If it doesn't find an image list, GetImageList returns nil (Delphi) or NULL (C++).

See Also

ImageIndex (see page 4916)

SubMenuImages (see page 4921)

Images (see page 4851)

Parent (see page 4919)

48.1.5.4.20 TMenuItem.GetItem Method

Pascal

```
function GetItem(Index: Integer): TMenuItem;
```

C++

```
__fastcall TMenuItem GetItem(int Index);
```

Description

This is GetItem, a member of class TMenuItem.

48.1.5.4.21 TMenuItem.GetMenuIndex Method

Pascal

```
function GetMenuIndex: Integer;
```

C++

```
__fastcall int GetMenuIndex();
```

Description

This is GetMenuIndex, a member of class TMenuItem.

48.1.5.4.22 TMenuItem.GetParentComponent Method

Returns the parent component of the menu item.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

GetParentComponent is called by the streaming system that loads and saves components. It ensures that objects are loaded and saved with their parent objects. GetParentComponent is overridden in descendant classes to return the appropriate parent for their particular type of component. TMenuItem (see page 4869) overrides it to identify its associated menu, if it has one, as its parent.

48.1.5.4.23 TMenuItem.GetParentMenu Method

Returns the main menu of which the menu item is part.

Pascal

```
function GetParentMenu: TMenu;
```

C++

```
__fastcall TMenu GetParentMenu();
```

Description

GetParentMenu returns the menu that contains the menu item.

See Also

System::TObject::Free

48.1.5.4.24 TMenuItem.HasParent Method

Indicates that the menu item has a parent that is responsible for loading and saving its data.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TMenuItem (see page 4869) overrides HasParent to return true, indicating that menu items have a parent. Applications seldom call the HasParent method. It is used by the streaming system that loads and saves forms to determine when another object is responsible for writing a component to a stream.

See Also

Parent (see page 4919)

HasParent

48.1.5.4.25 TMenuItem.IndexOf Method

Returns the position of a menu item within the Items (see page 4917) array.

Pascal

```
function IndexOf(Item: TMenuItem): Integer;
```

C++

```
__fastcall int IndexOf(TMenuItem Item);
```

Description

Call IndexOf to locate a menu item in the submenu headed by this menu item. IndexOf returns 0 for the first menu item, 1 for the second menu item, and so on. If the menu item is not in the submenu, IndexOf returns -1.

Note: IndexOf may not correspond to the position of the item in the menu the user sees if the menu includes some items that are not visible.

See Also

Items (see page 4917)

MenuIndex (see page 4919)

Visible (see page 4921)

48.1.5.4.26 TMenuItem.InitiateAction Method

Calls the action link's Update method if the menu item is associated with an action link.

Pascal

```
procedure InitiateAction; virtual;
```

C++

```
virtual __fastcall InitiateAction();
```

Description

InitiateAction calls the action link, if there is one for the menu item, to update the association between the action and the menu item.

See Also

Items (see page 4917)

MenuIndex (see page 4919)

Visible (see page 4921)

48.1.5.4.27 TMenuItem.Insert Method

Inserts a menu item into a specified position in the Items (see page 4917) array.

Pascal

```
procedure Insert(Index: Integer; Item: TMenuItem);
```

C++

```
__fastcall Insert(int Index, TMenuItem Item);
```

Description

Use Insert to insert a new menu item to the dropdown menu for this menu item. If this menu item is the value of the Items (see page 4917) property of a main menu or pop-up menu, Insert inserts an item into that menu. Specify the new menu item to add as the value of the Item parameter, and the position in the Items (see page 4917) array as the value of the Index parameter.

C++ Examples:

```
/*
This code inserts a menu item after the first item in a menu
named FileMenu:
*/
void __fastcall TForm1::InsertMenuItemClick(TObject *Sender)
{
  TMenuItem *NewItem = new TMenuItem(FileMenu);
  try
  {
    NewItem->Caption = "Form1";
    FileMenu->Insert(1, NewItem);
  }
  catch (Exception &e)
  {
    delete NewItem;
    throw;
  }
}
```

Delphi Examples:

```
{
This code inserts a menu item after the first item in a menu
named FileMenu:
}
procedure TForm1.InsertMenuItemClick(Sender: TObject);
var
  NewItem: TMenuItem;
begin
  NewItem := TMenuItem.Create(FileMenu);
```

```
    try
      NewItem.Caption := 'Do this';
      FileMenu.Insert(1, NewItem);
    except
      NewItem.Free; // Throw away the menu item if there is an error.
      raise; // reraise the exception
    end;
    // Don't free the menu item on success.  Need it for the menu!
  end;

  // Free all the menu items that we have created dynamically.
  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I : Integer;
  begin
    for I := FileMenu.Count - 1 downto 0 do  // Index backwards because Count changes as you
  remove items.
      if FileMenu.Items[I].Name = '' then  // Items created dynamically have not name.
        FileMenu.Items[I].Free;
  end;
```

See Also

Add (see page 4875)

Delete (see page 4882)

Items (see page 4917)

Remove (see page 4893)

Items (see page 4852)

48.1.5.4.28 TMenuItem.InsertNewLine Method

Pascal

```
function InsertNewLine(ABefore: Boolean; AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLine(Boolean ABefore, TMenuItem AItem);
```

Description

This is InsertNewLine, a member of class TMenuItem.

48.1.5.4.29 TMenuItem.InsertNewLineAfter Method

Inserts a separator bar after a specified item from the Items (see page 4917) property array.

Pascal

```
function InsertNewLineAfter(AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLineAfter(TMenuItem AItem);
```

Description

Use InsertNewLineAfter to insert a separator bar after the menu item specified by AItem. If AItem is not an item in the Items (see page 4917) property array, InsertNewLineAfter raises an exception.

InsertNewLineAfter returns the index of the newly inserted separator item in the Items (see page 4917) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 4900) to true so that

superfluous separators can be automatically removed.

See Also

Insert (see page 4888)

InsertNewLineBefore (see page 4890)

Items (see page 4917)

NewBottomLine (see page 4892)

NewTopLine (see page 4892)

AutoLineReduction (see page 4900)

48.1.5.4.30 TMenuItem.InsertNewLineBefore Method

Inserts a separator bar before a specified item from the Items (see page 4917) property array.

Pascal

```
function InsertNewLineBefore(AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLineBefore(TMenuItem AItem);
```

Description

Use InsertNewLineBefore to insert a separator bar before the menu item specified by AItem. If AItem is not an item in the Items (see page 4917) property array, InsertNewLineBefore raises an exception.

InsertNewLineBefore returns the index of the newly inserted separator item in the Items (see page 4917) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 4900) to true so that superfluous separators can be automatically removed.

See Also

Insert (see page 4888)

InsertNewLineAfter (see page 4889)

Items (see page 4917)

NewBottomLine (see page 4892)

NewTopLine (see page 4892)

AutoLineReduction (see page 4900)

48.1.5.4.31 TMenuItem.IsLine Method

Indicates whether the menu item represents a separator bar.

Pascal

```
function IsLine: Boolean;
```

C++

```
__fastcall Boolean IsLine();
```

Description

Call IsLine to determine whether the menu item is a separator bar. IsLine checks the Caption (see page 4902) of the menu

item and returns true if the value of Caption (see page 4902) is '-'. Menu items with this Caption (see page 4902) appear as separator bars.

See Also

Caption (see page 4902)

48.1.5.4.32 TMenuItem.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

48.1.5.4.33 TMenuItem.MeasureItem Method

Pascal

```
procedure MeasureItem(ACanvas: TCanvas; var Width: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(TCanvas ACanvas, int Width, int Height);
```

Description

This is MeasureItem, a member of class TMenuItem.

48.1.5.4.34 TMenuItem.MenuChanged Method

Pascal

```
procedure MenuChanged(Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall MenuChanged(Boolean Rebuild);
```

Description

This is MenuChanged, a member of class TMenuItem.

48.1.5.4.35 TMenuItem.NewBottomLine Method

Inserts a separator bar at the end of the Items (see page 4917) property array.

Pascal

```
function NewBottomLine: Integer;
```

C++

```
__fastcall int NewBottomLine();
```

Description

Use NewBottomLine to insert a separator bar at the end of the items property array. This is useful when building up submenus dynamically at runtime.

NewBottomLine returns the index of the newly inserted separator item in the Items (see page 4917) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 4900) to true so that superfluous separators can be automatically removed.

See Also

Add (see page 4875)

InsertNewLineBefore (see page 4890)

Items (see page 4917)

InsertNewLineAfter (see page 4889)

NewTopLine (see page 4892)

AutoLineReduction (see page 4900)

48.1.5.4.36 TMenuItem.NewTopLine Method

Inserts a separator bar at the beginning of the Items (see page 4917) property array.

Pascal

```
function NewTopLine: Integer;
```

C++

```
__fastcall int NewTopLine();
```

Description

Use NewTopLine to insert a separator bar at the beginning of the items property array. This is useful when building up submenus dynamically at runtime.

NewTopLine returns the index of the newly inserted separator item in the Items (see page 4917) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 4900) to true so that superfluous separators can be automatically removed.

See Also

Add (see page 4875)

InsertNewLineBefore (see page 4890)

Items (see page 4917)

InsertNewLineAfter (see page 4889)

NewBottomLine (see page 4892)

AutoLineReduction (see page 4900)

48.1.5.4.37 TMenuItem.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

48.1.5.4.38 TMenuItem.Remove Method

Removes a menu item from the Items (see page 4917) property array

Pascal

```
procedure Remove(Item: TMenuItem);
```

C++

```
__fastcall Remove(TMenuItem Item);
```

Description

Call Remove to remove a menu item and all its submenus from a menu. The Item parameter is the item to be removed. If Item is not a menu item in the Items (see page 4917) array, Remove raises an exception.

Remove does not free the item's memory. To remove an item from the menu and free its memory, call the Free method in the item you want to remove.

To temporarily hide a menu item, without deleting it, use the Visible (see page 4921) property.

See Also

Add (see page 4875)

Delete (see page 4882)

EMenuError (see page 4827)

Insert (see page 4888)

Items (see page 4917)

Visible (see page 4921)

48.1.5.4.39 TMenuItem.RethinkHotkeys Method

Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.

Pascal

```
function RethinkHotkeys: Boolean;
```

C++

```
__fastcall Boolean RethinkHotkeys();
```

Description

Call RethinkHotkeys to adjust the accelerator keys for the items in the Items (see page 4917) property and in any submenu of those items. RethinkHotkeys checks for items that do not have an accelerator key or that have duplicate accelerator keys. It then replaces any duplicate accelerators with unique characters and adds accelerator keys to items that don't have them. RethinkHotkeys gives first priority to existing accelerator key mappings, but may change them if necessary to ensure uniqueness.

RethinkHotkeys lets a menu adjust the accelerator keys when the AutoHotkeys (see page 4900) property is false. For any menu or submenu that has AutoHotkeys (see page 4900) set to true, this adjustment occurs automatically.

RethinkHotkeys returns true if RethinkHotkeys changed the Caption (see page 4902) property of any menu items, false otherwise.

See Also

AutoHotkeys (see page 4900)

AutoHotkeys (see page 4849)

Caption (see page 4902)

Items (see page 4917)

48.1.5.4.40 TMenuItem.RethinkLines Method

Removes superfluous separator lines from the Items (see page 4917) property array.

Pascal

```
function RethinkLines: Boolean;
```

C++

```
__fastcall Boolean RethinkLines();
```

Description

Call RethinkLines to remove superfluous separators from the items in the Items (see page 4917) property and in any submenu of those items. RethinkLines removes separators that appear at the very beginning or end of a submenu and separators that immediately follow another separator.

RethinkLines lets a menu remove superfluous separators when the AutoLineReduction (see page 4900) property is false. For any menu or submenu that has AutoLineReduction (see page 4900) set to true, this adjustment occurs automatically.

RethinkLines returns true if RethinkLines removed any menu items, false otherwise.

See Also

AutoLineReduction (see page 4900)

AutoLineReduction (see page 4850)

IsLine (see page 4890)

Items (see page 4917)

48.1.5.4.41 TMenuItem.SetBreak Method

Pascal

```
procedure SetBreak(Value: TMenuBreak);
```

C++

```
__fastcall SetBreak(TMenuBreak Value);
```

Description

This is SetBreak, a member of class TMenuItem.

48.1.5.4.42 TMenuItem.SetCaption Method

Pascal

```
procedure SetCaption(const Value: string);
```

C++

```
__fastcall SetCaption(const UnicodeString Value);
```

Description

This is SetCaption, a member of class TMenuItem.

48.1.5.4.43 TMenuItem.SetChecked Method

Pascal

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

C++

```
__fastcall SetChecked(Boolean Value);
```

Description

This is SetChecked, a member of class TMenuItem.

48.1.5.4.44 TMenuItem.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

This is SetChildOrder, a member of class TMenuItem.

48.1.5.4.45 TMenuItem.SetDefault Method

Pascal

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

C++

```
__fastcall SetDefault(Boolean Value);
```

Description

This is SetDefault, a member of class TMenuItem.

48.1.5.4.46 TMenuItem.SetEnabled Method

Pascal

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

C++

```
__fastcall SetEnabled(Boolean Value);
```

Description

This is SetEnabled, a member of class TMenuItem.

48.1.5.4.47 TMenuItem.SetGroupIndex Method

Pascal

```
procedure SetGroupIndex(Value: Byte);
```

C++

```
__fastcall SetGroupIndex(Byte Value);
```

Description

This is SetGroupIndex, a member of class TMenuItem.

48.1.5.4.48 TMenuItem.SetImageIndex Method

Pascal

```
procedure SetImageIndex(Value: TImageIndex);
```

C++

```
__fastcall SetImageIndex(TImageIndex Value);
```

Description

This is SetImageIndex, a member of class TMenuItem.

48.1.5.4.49 TMenuItem.SetMenuIndex Method

Pascal

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

C++

```
__fastcall SetMenuIndex(int Value);
```

Description

This is SetMenuIndex, a member of class TMenuItem.

48.1.5.4.50 TMenuItem.SetParentComponent Method

Pascal

```
procedure SetParentComponent(Value: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * Value);
```

Description

This is SetParentComponent, a member of class TMenuItem.

48.1.5.4.51 TMenuItem.SetRadioItem Method

Pascal

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

C++

```
__fastcall SetRadioItem(Boolean Value);
```

Description

This is SetRadioItem, a member of class TMenuItem.

48.1.5.4.52 TMenuItem.SetShortCut Method

Pascal

```
procedure SetShortCut(Value: TShortCut);
```

C++

```
__fastcall SetShortCut(TShortCut Value);
```

Description

This is SetShortCut, a member of class TMenuItem.

48.1.5.4.53 TMenuItem.SetVisible Method

Pascal

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

C++

```
__fastcall SetVisible(Boolean Value);
```

Description

This is SetVisible, a member of class TMenuItem.

48.1.5.4.54 TMenuItem.UpdateItems Method

Pascal

```
procedure UpdateItems;
```

C++

```
__fastcall UpdateItems();
```

Description

This is UpdateItems, a member of class TMenuItem.

48.1.5.5 TMenuItem Properties

The properties of the TMenuItem class are listed here.

Protected Properties

	Name	Description
	ActionLink (see page 4899)	This is ActionLink, a member of class TMenuItem.

Public Properties

	Name	Description
R	Command (see page 4907)	Specifies the Windows Command ID associated with the menu item.
R	Count (see page 4907)	Indicates the number of subitems of the menu item.
R	Handle (see page 4915)	Indicates the Windows menu handle of the drop-down menu associated with the menu item.
R	Items (see page 4917)	Lists the menu items in the submenu of the menu item.
	MenuIndex (see page 4919)	Indicates the index of the menu item within its parent menu.
R	Parent (see page 4919)	Identifies the parent menu item of this menu item.

Published Properties

	Name	Description
	Action (see page 4899)	Designates the action associated with the menu item.
	AutoCheck (see page 4899)	Indicates whether the menu item's checked state toggles automatically when the item is clicked.
	AutoHotkeys (see page 4900)	Determines whether the accelerator keys for submenu items can be reset automatically.
	AutoLineReduction (see page 4900)	Determines whether redundant separator bars are automatically removed from the submenu.

	Bitmap (see page 4901)	Designates a bitmap that is drawn for the menu item.
	Break (see page 4901)	Determines whether the menu item starts a new column in the menu.
	Caption (see page 4902)	Specifies the text of the menu item.
	Checked (see page 4906)	Specifies whether a check mark should appear beside the Caption (see page 4902).
	Default (see page 4912)	Specifies whether the menu item is invoked when the parent item is double clicked.
	Enabled (see page 4912)	Specifies whether the menu item is enabled.
	GroupIndex (see page 4914)	Identifies the logical group to which the menu item belongs.
	HelpContext (see page 4915)	Specifies the help context ID associated with the menu item.
	Hint (see page 4916)	Specifies the text string that can appear when the user moves the mouse pointer over a menu item.
	ImageIndex (see page 4916)	Indicates which image maintained by the parent menu appears next to the menu item.
	RadioItem (see page 4920)	Specifies whether the menu item is mutually exclusive with other menu items in its group.
	ShortCut (see page 4920)	Specifies the key combination users can type to quickly access the menu item.
	SubMenuImages (see page 4921)	Lists the images that can appear beside individual items in a submenu of this menu item.
	Visible (see page 4921)	Specifies whether the menu item appears in the menu.

48.1.5.5.1 TMenuItem.Action Property

Designates the action associated with the menu item.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action designates the action object that is associated with the menu item. Actions can be associated with a menu item by using the action list editor at design-time. Actions are a way of centralizing user responses to commands.

48.1.5.5.2 TMenuItem.ActionLink Property

Pascal

```
property ActionLink: TMenuActionLink;
```

C++

```
__property TMenuActionLink ActionLink;
```

Description

This is ActionLink, a member of class TMenuItem.

48.1.5.5.3 TMenuItem.AutoCheck Property

Indicates whether the menu item's checked state toggles automatically when the item is clicked.

Pascal

```
property AutoCheck: Boolean;
```

C++

```
__property Boolean AutoCheck;
```

Description

When AutoCheck is true, then every time the menu item is clicked, the value of the Checked (see page 4906) property toggles automatically before the OnClick (see page 4922) event occurs. When AutoCheck is false, the application must explicitly set the value of the Checked (see page 4906) property (for example, in an OnClick (see page 4922) event handler).

See Also

Checked (see page 4906)

OnClick (see page 4922)

48.1.5.5.4 TMenuItem.AutoHotkeys Property

Determines whether the accelerator keys for submenu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuItemAutoFlag;
```

C++

```
__property TMenuItemAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the accelerator keys for the items in the Items (see page 4917) property can be reset if necessary to ensure that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenuItem (see page 4869) tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, it checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic or if AutoHotkeys is maParent and the parent menu item adjusts accelerator key mappings. Otherwise, the menu does not alter the items listed in the Items (see page 4917) property, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

Note: When AutoHotkeys disallows automatic adjustment of accelerator keys, an application can still adjust the menu items by calling the RethinkHotkeys (see page 4894) method. AutoHotkeys merely controls whether this adjustment occurs automatically.

See Also

AutoHotkeys (see page 4849)

Items (see page 4917)

RethinkHotkeys (see page 4894)

48.1.5.5.5 TMenuItem.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the submenu.

Pascal

```
property AutoLineReduction: TMenuItemAutoFlag;
```

C++

```
__property TMenuItemAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu item should automatically ensure that the submenu defined by its Items (see page 4917) property does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, the menu item removes superfluous separator bars if AutoLineReduction is maAutomatic or AutoLineReduction is maParent and the parent menu item removes superfluous separator bars. Otherwise, the menu item does not remove separators from the Items (see page 4917) property, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

Note: When AutoLineReduction disallows automatic removal of separator bars, an application can still remove superfluous separators by calling the RethinkLines (see page 4894) method. AutoLineReduction merely controls whether this adjustment occurs automatically.

See Also

AutoLineReduction (see page 4850)

Items (see page 4917)

RethinkLines (see page 4894)

48.1.5.5.6 TMenuItem.Bitmap Property

Designates a bitmap that is drawn for the menu item.

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Use Bitmap to provide a custom image that appears next to the caption in the menu item. On owner-draw menus, Bitmap can be used from an OnDrawItem (see page 4928) event handler to represent the entire menu item or a portion of the menu item.

Note: If the ImageIndex (see page 4916) property is set and the parent menu has a non-nil (Delphi) or NULL (C++) Images property, the image identified by ImageIndex (see page 4916) is used instead of Bitmap.

See Also

ImageIndex (see page 4916)

Images (see page 4851)

Caption (see page 4902)

OwnerDraw (see page 4856)

OnDrawItem (see page 4928)

48.1.5.5.7 TMenuItem.Break Property

Determines whether the menu item starts a new column in the menu.

Pascal

```
property Break: TMenuBreak;
```

C++

```
__property TMenuBreak Break;
```

Description

Set Break to break a long menu into columns at the item described by this menu item. The value of Break indicates whether the menu is broken into columns, and if so, whether a bar appears between the columns of the parent menu.

The default is mbNone (no breaking into columns).

See Also

Caption (see page 4902)

Parent (see page 4919)

48.1.5.5.8 TMenuItem.Caption Property

Specifies the text of the menu item.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Set Caption to the string the user sees for this item in the menu. To underline a character in a string, include an ampersand (&) before the character. This type of character is called an accelerator character. If Caption includes an accelerator character, the user can select the menu item by pressing Alt while typing the underlined character.

Tip: To automatically assign or adjust the accelerator characters in menu item captions, use the RethinkHotkeys (see page 4894) method.

Use the Caption property to identify the menu item as a line that separates the menu into parts. Specify a hyphen character (-) as the value of Caption for the menu item to indicate that the menu item is a separator. To separate the menu into multiple columns, use the Break (see page 4901) property instead.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";   // The owner will cleanup these menu items.
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items->Items[1]->Add(NewItem);
  // now create and add a menu item for each form
```

```
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this); // The owner will cleanup these menu items.
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items->Items[1]->Add(NewItem);
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
```

```
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
```

```
  end;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;
```

```
{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

Break (see page 4901)

Checked (see page 4906)

Caption

RethinkHotkeys (see page 4894)

48.1.5.5.9 TMenuItem.Checked Property

Specifies whether a check mark should appear beside the Caption (see page 4902).

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to allow a menu item to work like a check box. If Checked is true, the menu item appears checked. If Checked is false, the menu item remains unchecked.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Caption (see page 4902)

RadioItem (see page 4920)

Checked

AutoCheck (see page 4899)

48.1.5.5.10 TMenuItem.Command Property

Specifies the Windows Command ID associated with the menu item.

Pascal

```
property Command: Word;
```

C++

```
__property Word Command;
```

Description

Use Command in an application that directly handles WM_COMMAND (see page 5148) messages. When the user selects the menu item, Windows sends a WM_COMMAND (see page 5148) to the window associated with the menu. The ItemID of the message contains the value of the Command property of the menu item that was selected.

See Also

OnClick (see page 4922)

DispatchCommand (see page 4841)

WindowHandle (see page 4857)

48.1.5.5.11 TMenuItem.Count Property

Indicates the number of subitems of the menu item.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of subitems listed in the Items (see page 4917) property array. When the user clicks on a menu item that has subitems, a dropdown menu appears which displays those subitems. Each subitem can, in turn, contain additional subitems. The Count property counts only the immediate subitems of the menu item.

C++ Examples:

```
/*
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
*/
void __fastcall TForm1::DisableSubItemsClick(TObject *Sender)
{
  for (int i = 0; i < File1->Count; i++)
  {
    File1->Items[i]->Enabled = false;
```

```
  }
}

void __fastcall TForm1::Edit1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Edit command");
}

void __fastcall TForm1::File1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the File command");
}

void __fastcall TForm1::FileSub11Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub1 command");
}

void __fastcall TForm1::FileSub21Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub2 command");
}

void __fastcall TForm1::FileSub31Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub3 command");
}

void __fastcall TForm1::Options1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Options command");
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
```

```
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;
```

```
procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

Items (see page 4917)

MenuIndex (see page 4919)

48.1.5.5.12 TMenuItem.Default Property

Specifies whether the menu item is invoked when the parent item is double clicked.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

Set Default to specify whether a menu item is the default item in a submenu. Default menu items are executed when the parent item is double-clicked, allowing users to avoid navigating the submenus that contain them.

The default item appears in boldface. A submenu can have only one default item. If the Items (see page 4917) list of the parent menu item already contains a default menu item, setting the Default property to true sets the Default property of the previous default item to false.

See Also

Click (see page 4881)

Parent (see page 4919)

48.1.5.5.13 TMenuItem.Enabled Property

Specifies whether the menu item is enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to enable or disable a menu item. If Enabled is true, the Click (see page 4881) method is called when the user selects the item with the mouse. If Enabled is false, the menu item appears dimmed and the user cannot select it. However, calling the Click (see page 4881) method works even when Enabled is false.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

C++ Examples:

```
/*
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
*/
void __fastcall TForm1::DisableSubItemsClick(TObject *Sender)
{
  for (int i = 0; i < File1->Count; i++)
```

```
  {
    File1->Items[i]->Enabled = false;
  }
}

void __fastcall TForm1::Edit1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Edit command");
}

void __fastcall TForm1::File1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the File command");
}

void __fastcall TForm1::FileSub11Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub1 command");
}

void __fastcall TForm1::FileSub21Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub2 command");
}

void __fastcall TForm1::FileSub31Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub3 command");
}

void __fastcall TForm1::Options1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Options command");
}
```

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
```

```
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
```

See Also

Click (see page 4881)

Visible (see page 4921)

Enabled

48.1.5.5.14 TMenuItem.GroupIndex Property

Identifies the logical group to which the menu item belongs.

Pascal

```
property GroupIndex: Byte;
```

C++

```
__property Byte GroupIndex;
```

Description

Use GroupIndex to control how menus are merged. In applications that contain multiple forms, items from the menus of secondary forms can be merged into the application's main menu as those forms become active. MDI applications always merge the menus of child windows with the main menu of the parent window.

By default, all menu items in a menu bar (main menu) have the same GroupIndex value. To use GroupIndex to merge menus, set the GroupIndex value of each successive menu item to a value equal to or greater than that of the previous menu item.

When a menu item in a menu bar that is merged into the application's main menu has the same GroupIndex value as a menu item in the main menu, the menu item in the application's main menu is replaced by the corresponding menu item. If multiple menu items in the application's main menu have the same GroupIndex value, they can be replaced only by separate menu items from the other form's menu that also have repeated GroupIndex values. Thus, the first main menu item with a given value is replaced by the first item in the second form with that value, the second item in the main form is replaced by the second item in the second form, and so on.

When a menu item in a menu bar that is merged into the application's main menu has a GroupIndex that falls between the GroupIndex values of two adjacent items on the application's main menu, that menu item is inserted between the two adjacent items. If the item's GroupIndex value is greater than all other GroupIndex values in the application's main menu, the item appears at the end of the main menu. If the GroupIndex value is less than all other GroupIndex values in the application's main menu, the menu item appears at the beginning of the main menu bar.

When an object created by an OLE 2.0 server application is activated, the server can try to merge its menus with the menus of the container application. The GroupIndex property is used to replace up to three main menu items with items from the server, in the manner described above. The server application uses the following predefined GroupIndex values to locate menu items to replace:

Group	Index	Description
Edit	1	Server menu items for editing the active OLE object
View	3	Server menu items for modifying the view of the OLE object.
Help	5	Server menu items for accessing the server's online Help.

Note: See the documentation for the OLE server for information about whether it attempts menu merge during in-place activation.

GroupIndex can also be used to define groups of menu items that act like radio buttons. To allow menu items to behave like radio buttons, set the RadioItem (see page 4920) property to all menu items in the group to true, and assign each menu item the same value of GroupIndex. Checking any menu item in the group will then uncheck the others.

See Also

MenuIndex (see page 4919)

RadioItem (see page 4920)

AutoMerge (see page 4834)

Merge (see page 4831)

PopulateOle2Menu (see page 4832)

48.1.5.5.15 TMenuItem.Handle Property

Indicates the Windows menu handle of the drop-down menu associated with the menu item.

Pascal

```
property Handle: HMENU;
```

C++

```
__property HMENU Handle;
```

Description

Use Handle when calling a Windows API function that requires a menu handle. Handle is only valid if the Count (see page 4907) property is greater than 0.

See Also

Count (see page 4907)

Items (see page 4917)

48.1.5.5.16 TMenuItem.HelpContext Property

Specifies the help context ID associated with the menu item.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

Use HelpContext to associate the menu item with a screen in the Help system of the application. Each screen in the Help system

should have a unique context ID. When a menu item is selected using the keyboard, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

HelpFile (see page 1974)

HelpContext

HelpContext

48.1.5.5.17 TMenuItem.Hint Property

Specifies the text string that can appear when the user moves the mouse pointer over a menu item.

Pascal

```
property Hint: string;
```

C++

```
__property UnicodeString Hint;
```

Description

Set Hint to a string that provides more information about the meaning of the menu item than the Caption (see page 4902). The hint text appears in the **Status Bar** when the user pauses with the mouse over the menu item if Help Hints are enabled (that is, if the Form's and the Application's Show Hint properties are true). It is also available for the code in the application's On Hint event handler.

The value of Hint can specify both a short value for the Help Hint window and a longer string to be used by the OnHint event handler. To provide both a short and a long hint, set Hint to the short string, followed by a vertical bar (|), followed by the long string.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

48.1.5.5.18 TMenuItem.ImageIndex Property

Indicates which image maintained by the parent menu appears next to the menu item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to designate an image that should appear next to the caption when the menu item is drawn. On top-level menu items, ImageIndex specifies a zero-offset index into the Images property of the parent menu (TMenu (see page 4837) or TPopupMenu (see page 4935)). On submenu items, ImageIndex is the zero-offset index into the parent item's SubMenuImages (see page 4921) property.

If the parent menu or item does not maintain a list of images, use the Bitmap (see page 4901) property to provide an image for a single menu item. However, any image specified by ImageIndex takes precedence over the Bitmap (see page 4901) property. Bitmap (see page 4901) is only used when ImageIndex has a negative value or the parent's Images or SubMenuImages (see page 4921) property is nil (Delphi) or NULL (C++).

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Images (see page 4851)

Bitmap (see page 4901)

ImageIndex

SubMenuImages (see page 4921)

48.1.5.5.19 TMenuItem.Items Property

Lists the menu items in the submenu of the menu item.

Pascal

```
property Items [Index: Integer]: TMenuItem;
```

C++

```
__property TMenuItem Items[int Index];
```

Description

Use Items to access to a subitem by its position in the list of subitems. The value of Index is the position of the subitem within the Items array. For example, if an application has a File drop-down menu that contains the menu items New, Open, and Save, in that order, the expression

```
FileMenu.Items[2]
FileMenu->Items[2]
```

refers to the Save command.

Note: In Delphi, Items is the default property for TMenuItem (see page 4869). This means you can leave off the property name. Thus, instead of writing FileMenu.Items[2], you can write FileMenu[2].

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

C++ Examples:

```
/*
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
*/
void __fastcall TForm1::DisableSubItemsClick(TObject *Sender)
{
  for (int i = 0; i < File1->Count; i++)
  {
    File1->Items[i]->Enabled = false;
  }
}

void __fastcall TForm1::Edit1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Edit command");
}

void __fastcall TForm1::File1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the File command");
}
```

```
void __fastcall TForm1::FileSub11Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub1 command");
}

void __fastcall TForm1::FileSub21Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub2 command");
}

void __fastcall TForm1::FileSub31Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the FileSub3 command");
}

void __fastcall TForm1::Options1Click(TObject *Sender)
{
  ListBox1->Items->Add("This is the Options command");
}
```

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
```

See Also

Count (see page 4907)

MenuIndex (see page 4919)

Parent (see page 4919)

operator_[]

48.1.5.5.20 TMenuItem.MenuIndex Property

Indicates the index of the menu item within its parent menu.

Pascal

```
property MenuIndex: Integer;
```

C++

```
__property int MenuIndex;
```

Description

Use MenuIndex to locate the menu item in the Items (see page 4917) property of its parent menu item. Changing this value moves the menu item within its menu.

Note: MenuIndex may not correspond to the position of the item in the menu the user sees. The menu can include items that are not visible.

See Also

Count (see page 4907)

GroupIndex (see page 4914)

Items (see page 4917)

Parent (see page 4919)

Visible (see page 4921)

48.1.5.5.21 TMenuItem.Parent Property

Identifies the parent menu item of this menu item.

Pascal

```
property Parent: TMenuItem;
```

C++

```
__property TMenuItem Parent;
```

Description

Read Parent to determine the menu item that represents the dropdown menu that contains this menu item. If the menu item is a top level menu item in a main menu or pop-up menu, Parent is the value of that menu's Items (see page 4917) property.

See Also

Items (see page 4917)

MenuIndex (see page 4919)

Items (see page 4852)

48.1.5.5.22 TMenuItem.RadioItem Property

Specifies whether the menu item is mutually exclusive with other menu items in its group.

Pascal

```
property RadioItem: Boolean;
```

C++

```
__property Boolean RadioItem;
```

Description

Use RadioItem to make the menu item behave like a radio button with a group of other menu items. The group of menu items is the set of all menu items in a pop-up or drop-down menu that have the same value of GroupIndex (see page 4914). When RadioItem is true, only one menu item in the group that contains the menu item can be checked at a time. The single selected menu item in the group is indicated by a round dot next to the Caption (see page 4902).

Note: RadioItem does not automatically select the item when the user clicks it at runtime: you must still explicitly set the Checked (see page 4906) property to true in response to the OnClick (see page 4922) event. RadioItem simply changes the appearance of the Checked (see page 4906) state (from a check mark to a round dot) and automatically unchecks other menu items in the same group when you set the Checked (see page 4906) property to true.

RadioItem does not work for the top-level items in a main menu.

Note: Under Windows NT 3.51, the round dot indicator only appears if the "new shell" has been installed.

See Also

Caption (see page 4902)

Checked (see page 4906)

GroupIndex (see page 4914)

TMainMenu (see page 4828)

OnClick (see page 4922)

48.1.5.5.23 TMenuItem.ShortCut Property

Specifies the key combination users can type to quickly access the menu item.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

Use ShortCut to allow the user to type a key combination instead of selecting the item using the menu. This key combination appears to the right of the menu item in the menu.

When setting menu shortcuts at design time, the Object Inspector provides a long list of key combinations to choose from. To create menu shortcuts at runtime, use the global ShortCut or TextToShortCut (see page 4963) function. For example, the following line creates a shortcut, Ctrl+C and assigns it to a menu item named CloseCommand.

```
CloseCommand.ShortCut := ShortCut(Word('C'), [ssCtrl]);
CloseCommand->ShortCut = ShortCut(Word('C'), TShiftState() << ssCtrl);
```

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

ShortCut (see page 4958)

ShortCutToKey (see page 4960)

ShortCutToText (see page 4961)

TextToShortCut (see page 4963)

IsShortCut (see page 4845)

ShortCut

48.1.5.5.24 TMenuItem.SubMenuImages Property

Lists the images that can appear beside individual items in a submenu of this menu item.

Pascal

```
property SubMenuImages: TCustomImageList;
```

C++

```
__property TCustomImageList SubMenuImages;
```

Description

Use SubMenuImages to supply a set of images that can appear beside items listed in the Items (see page 4917) property. Individual menu items set their ImageIndex (see page 4916) property to indicate which image in the list appears to the left of the menu item caption.

Tip: If the application uses action lists, this property can be the value of the command list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 4916)

Images

Items (see page 4917)

48.1.5.5.25 TMenuItem.Visible Property

Specifies whether the menu item appears in the menu.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to prevent the user from seeing an item in the menu. For example, remove menu items that do not apply to the current component of a pop-up menu in an OnPopup event handler.

If Visible is true, the menu item appears. If Visible is false, the menu item is hidden.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Enabled (see page 4912)

OnPopup (see page 4949)

PopupComponent (see page 4946)

Visible

48.1.5.6 TMenuItem Events

The events of the TMenuItem class are listed here.

Published Events

	Name	Description
	OnAdvancedDrawItem (see page 4922)	Occurs when an owner-draw menu needs to be drawn.
	OnClick (see page 4922)	Occurs when the user clicks menu item.
	OnDrawItem (see page 4928)	Occurs when an owner-draw menu needs to be drawn.
	OnMeasureItem (see page 4929)	Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

48.1.5.6.1 TMenuItem.OnAdvancedDrawItem Event

Occurs when an owner-draw menu needs to be drawn.

Pascal

```
property OnAdvancedDrawItem: TAdvancedMenuDrawItemEvent;
```

C++

```
__property TAdvancedMenuDrawItemEvent OnAdvancedDrawItem;
```

Description

Write an OnAdvancedDrawItem event handler to render the image of a menu item in an owner-draw menu.

In the event handler, use the Bitmap (see page 4901) property or the ImageIndex (see page 4916) property to access a bitmap that represents the menu item.

Note: OnAdvancedDrawItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

See Also

OnMeasureItem (see page 4929)

Bitmap (see page 4901)

ImageIndex (see page 4916)

Images (see page 4851)

OwnerDraw (see page 4856)

OnDrawItem (see page 4928)

48.1.5.6.2 TMenuItem.OnClick Event

Occurs when the user clicks menu item.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write an OnClick event handler to implement the desired behavior for when the user selects the menu item. An advantage of using the OnClick event handler over directly handling WM_COMMAND (see page 5148) messages sent when the user selects the menu item, is that only the OnClick event occurs when the Click (see page 4881) method is called.

Note: This event handler is not stored with the menu item if it is supplied by a TAction object.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
```

```
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
```

```
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
```

```
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
```

```
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

Click (see page 4881)

Command (see page 4907)

OnExecute

48.1.5.6.3 TMenuItem.OnDrawItem Event

Occurs when an owner-draw menu needs to be drawn.

Pascal

```
property OnDrawItem: TMenuDrawItemEvent;
```

C++

```
__property TMenuDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to render the image of a menu item in an owner-draw menu. In the event handler, use the Bitmap (see page 4901) property or the ImageIndex (see page 4916) property to access a bitmap that represents the menu item.

Note: OnDrawItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

Tip: For more precise information about the menu item's state, use the OnAdvancedDrawItem (see page 4922) event instead.

See Also

OnMeasureItem (see page 4929)

Bitmap (see page 4901)

ImageIndex (see page 4916)

Images (see page 4851)

OwnerDraw (see page 4856)

OnAdvancedDrawItem (see page 4922)

48.1.5.6.4 TMenuItem.OnMeasureItem Event

Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

Pascal

```
property OnMeasureItem: TMenuMeasureItemEvent;
```

C++

```
__property TMenuMeasureItemEvent OnMeasureItem;
```

Description

Write an OnMeasureItem event handler to indicate the size of owner-draw menu items. If the menu item is using a bitmap image, the event handler can use the Bitmap (see page 4901) property or the ImageIndex (see page 4916) property to access the bitmap to determine its size.

Note: OnMeasureItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

See Also

OnDrawItem (see page 4928)

Bitmap (see page 4901)

ImageIndex (see page 4916)

Images (see page 4851)

OwnerDraw (see page 4856)

TextExtent (see page 2705)

48.1.6 TMenuItemEnumerator Class

Class Hierarchy

Pascal

```
TMenuItemEnumerator = class;
```

C++

```
class TMenuItemEnumerator;
```

File

Menus

Description

This is class Menus.TMenuItemEnumerator.

48.1.6.1 TMenuItemEnumerator Members

The following tables list the members exposed by TMenuItemEnumerator.

Public Methods

	Name	Description
	Create (see page 4930)	This is Create, a member of class TMenuItemEnumerator.
	GetCurrent (see page 4930)	This is GetCurrent, a member of class TMenuItemEnumerator.
	MoveNext (see page 4930)	This is MoveNext, a member of class TMenuItemEnumerator.

Public Properties

	Name	Description
	Current (see page 4931)	This is Current, a member of class TMenuItemEnumerator.

48.1.6.2 TMenuItemEnumerator.Create Constructor

Pascal

```
constructor Create(AMenuItem: TMenuItem);
```

C++

```
__fastcall TMenuItemEnumerator(TMenuItem AMenuItem);
```

Description

This is Create, a member of class TMenuItemEnumerator.

48.1.6.3 TMenuItemEnumerator Methods

The methods of the TMenuItemEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 4930)	This is GetCurrent, a member of class TMenuItemEnumerator.
	MoveNext (see page 4930)	This is MoveNext, a member of class TMenuItemEnumerator.

48.1.6.3.1 TMenuItemEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TMenuItem;
```

C++

```
__fastcall TMenuItem GetCurrent();
```

Description

This is GetCurrent, a member of class TMenuItemEnumerator.

48.1.6.3.2 TMenuItemEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TMenuItemEnumerator.

48.1.6.4 TMenuItemEnumerator Properties

The properties of the TMenuItemEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 4931)	This is Current, a member of class TMenuItemEnumerator.

48.1.6.4.1 TMenuItemEnumerator.Current Property

Pascal

```
property Current: TMenuItem;
```

C++

```
__property TMenuItem Current;
```

Description

This is Current, a member of class TMenuItemEnumerator.

48.1.7 TMenuItemStack Class

Class Hierarchy

Pascal

```
TMenuItemStack = class(TStack);
```

C++

```
class TMenuItemStack : public TStack;
```

File

Menus

Description

This is class Menus.TMenuItemStack.

48.1.7.1 TMenuItemStack Members

The following tables list the members exposed by TMenuItemStack.

Public Methods

	Name	Description
	ClearItem (see page 4932)	This is ClearItem, a member of class TMenuItemStack.

48.1.7.2 TMenuItemStack Methods

The methods of the TMenuItemStack class are listed here.

Public Methods

	Name	Description
	ClearItem (see page 4932)	This is ClearItem, a member of class TMenuItemStack.

48.1.7.2.1 TMenuItemStack.ClearItem Method

Pascal

```
procedure ClearItem(AItem: TMenuItem);
```

C++

```
__fastcall ClearItem(TMenuItem AItem);
```

Description

This is ClearItem, a member of class TMenuItemStack.

48.1.8 TPopupList Class

TPopupList manages the window that handles popup menu messages.

Class Hierarchy

Pascal

```
TPopupList = class(TList);
```

C++

```
class TPopupList : public TList;
```

File

Menus

Description

TPopupList is the type of the global PopupList (see page 4975) variable, which provides centralized message handling for all popup menus in an application.

See Also

TPopupMenu (see page 4935)

PopupList (see page 4975)

48.1.8.1 TPopupList Members

The following tables list the members exposed by TPopupList.

Protected Fields

	Name	Description
	FWindow (see page 4933)	This is FWindow, a member of class TPopupList.

Protected Methods

	Name	Description
	MainWndProc (see page 4934)	Wraps the WndProc (see page 4935) method with an exception-handling layer.
V	WndProc (see page 4935)	Provides centralized message handling for all popup menus in the application.

Public Methods

	Name	Description
	Add (see page 4934)	Inserts a new popup menu at the end of the list.
	Remove (see page 4934)	Removes a popup menu from the list.

Public Properties

	Name	Description
R	Window (see page 4935)	Provides access to the Window handle of the hidden window that processes popup menu messages.

48.1.8.2 TPopupList Fields

The fields of the TPopupList class are listed here.

Protected Fields

	Name	Description
	FWindow (see page 4933)	This is FWindow, a member of class TPopupList.

48.1.8.2.1 TPopupList.FWindow Field

Pascal

```
FWindow: HWND;
```

C++

```
HWND FWindow;
```

Description

This is FWindow, a member of class TPopupList.

48.1.8.3 TPopupList Methods

The methods of the TPopupList class are listed here.

Protected Methods

	Name	Description
	MainWndProc (see page 4934)	Wraps the WndProc (see page 4935) method with an exception-handling layer.
V	WndProc (see page 4935)	Provides centralized message handling for all popup menus in the application.

Public Methods

	Name	Description
	Add (see page 4934)	Inserts a new popup menu at the end of the list.
	Remove (see page 4934)	Removes a popup menu from the list.

48.1.8.3.1 TPopupList.Add Method

Inserts a new popup menu at the end of the list.

Pascal

```
procedure Add(Popup: TPopupMenu);
```

C++

```
__fastcall Add(TPopupMenu Popup);
```

Description

Do not call the Add method. Popup menus automatically add themselves to the list when they are created.

Add increments Count and, if necessary, allocates memory by increasing the value of Capacity. When the first popup menu is added to the list, Add creates the hidden window whose handle is available as the Window (see page 4935) property.

See Also

Remove (see page 4934)

Window (see page 4935)

48.1.8.3.2 TPopupList.MainWndProc Method

Wraps the WndProc (see page 4935) method with an exception-handling layer.

Pascal

```
procedure MainWndProc(var Message: TMessage);
```

C++

```
__fastcall MainWndProc(TMessage Message);
```

Description

Do not use the protected MainWndProc method. It is used internally when creating the hidden window that responds to popup menu messages. To subclass the window procedure that handles messages sent to popup menus, override the WndProc (see page 4935) method instead.

See Also

WndProc (see page 4935)

48.1.8.3.3 TPopupList.Remove Method

Removes a popup menu from the list.

Pascal

```
procedure Remove(Popup: TPopupMenu);
```

C++

```
__fastcall Remove(TPopupMenu Popup);
```

Description

Do not call the Remove method. It is called automatically by popup menus from their destructors, to remove themselves from the list.

See Also

Add (see page 4934)

48.1.8.3.4 TPopupList.WndProc Method

Provides centralized message handling for all popup menus in the application.

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to subclass the window procedure that handles messages directed at the popup menus in an application. WndProc handles messages by dispatching commands, displaying help, painting menu item, and so on.

48.1.8.4 TPopupList Properties

The properties of the TPopupList class are listed here.

Public Properties

	Name	Description
	Window (see page 4935)	Provides access to the Window handle of the hidden window that processes popup menu messages.

48.1.8.4.1 TPopupList.Window Property

Provides access to the Window handle of the hidden window that processes popup menu messages.

Pascal

```
property Window: HWND;
```

C++

```
__property HWND Window;
```

Description

Use Window for Windows API calls that require a window handle. The hidden window that processes messages for the popup menus in an application is not created until the first popup menu is added to the list. If the list contains no popup menus (that is, if no popup menus are created by the application), Window is 0.

48.1.9 TPopupMenu Class

TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

Class Hierarchy

Pascal

```
TPopupMenu = class(TMenu);
```

C++

```
class TPopupMenu : public TMenu;
```

File

Menus

Description

Use TPopupMenu to define the pop-up menu that appears when the user clicks on a control with the right mouse button. To make a pop-up menu available, assign the TPopupMenu object to the control's PopupMenu property.

Tip: To create a popup menu that users can configure at runtime, use the TStandardMenuPopup component with an action manager instead.

Note: If the popup menu's ParentBiDiMode (see page 4946) is true, the popup menu's BiDiMode (see page 4943) is set to the BiDiMode (see page 4943) of the control that activates it. If a control cannot be found, the BiDiMode (see page 4943) is set to the value of the Application object's BiDiMode (see page 4943). The popup menu's BiDiMode (see page 4943) affects all of its menu items.

See Also

TMainMenu (see page 4828)

TMenuItem (see page 4869)

NewPopupMenu (see page 4956)

TStandardMenuPopup

TActionManager

Providing a Pop-up Menu

Setting Up Action Bands

48.1.9.1 TPopupMenu Members

The following tables list the members exposed by TPopupMenu.

Protected Methods

	Name	Description
	AdjustBiDiBehavior (see page 4841)	Applies the BiDiMode (see page 4850) property to the menu when the Windows control is created.
	DoBiDiModeChanged (see page 4842)	Updates the menu when the BiDiMode (see page 4850) property changes.
V	DoChange (see page 4842)	Generates an OnChange (see page 4858) event.
	DoGetMenuString (see page 4843)	Returns the caption for an item in the menu.
V	DoPopup (see page 4939)	Generates an OnPopup (see page 4949) event.
V	GetHandle (see page 4844)	Returns the value of the Handle (see page 4851) property.
	IsOwnerDraw (see page 4845)	Indicates whether menu items can contain more than simple captions.

	Loaded (see page 4846)	Calls the DoChange (see page 4842) method when the menu is first loaded into memory.
	MenuChanged (see page 4846)	Responds when the menu changes.
	Notification (see page 4847)	Responds when components are added or deleted.
	SetChildOrder (see page 4848)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	SetPopupPoint (see page 4940)	This is SetPopupPoint, a member of class TPopupMenu.
	UpdateItems (see page 4849)	Causes all menu items to update themselves.
	UseRightToLeftAlignment (see page 4940)	Specifies whether the control's alignment is in a right-to-left mode.

Public Methods

	Name	Description
	Create (see page 4938)	Creates an instance of TPopupMenu (see page 4935).
	CreateMenuItem (see page 4841)	This is CreateMenuItem, a member of class TMenu.
	Destroy (see page 4938)	Destroys an instance of TPopupMenu (see page 4935).
	DispatchCommand (see page 4841)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 4842)	Dispatches a pop-up menu command.
	FindItem (see page 4843)	Locates and returns a menu item.
	GetChildren (see page 4844)	Executes a callback for child menu items.
	GetHelpContext (see page 4844)	Returns the help context ID associated with a menu item.
	IsRightToLeft (see page 4845)	Indicates whether menu items read from right to left.
	IsShortCut (see page 4845)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	ParentBiDiModeChanged (see page 4847)	This is the overview for the ParentBiDiModeChanged method overload.
	Popup (see page 4939)	Displays the pop-up menu onscreen.
	ProcessMenuChar (see page 4848)	Responds to keyboard input for the menu.

Public Properties

	Name	Description
	Handle (see page 4851)	Provides access to the Windows menu handle for the menu.
	PopupComponent (see page 4946)	Indicates the component that last displayed the popup menu in response to a right mouse click.
	PopupPoint (see page 4948)	Indicates the location where the pop-up menu appears.
	WindowHandle (see page 4857)	Provides access to the handle of the window that uses the menu.

Published Events

	Name	Description
	OnChange (see page 4949)	Occurs when the menu changes.
	OnPopup (see page 4949)	Occurs just before the pop-up menu appears.

Published Properties

	Name	Description
	Alignment (see page 4941)	Determines where the pop-up menu appears when the user clicks the right mouse button.
	AutoHotkeys (see page 4941)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4942)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoPopup (see page 4942)	Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.
	BiDiMode (see page 4943)	Determines the layout of the menu in Middle East locales.
	HelpContext (see page 4944)	Specifies a context ID for the entire pop-up menu.
	Images (see page 4944)	Lists the images that can appear beside individual menu items.
	Items (see page 4852)	Describes the elements of the menu.
	MenuAnimation (see page 4945)	Specifies how the menu appears when it is first displayed.
	OwnerDraw (see page 4945)	Specifies whether the menu items are drawn by the application.

	ParentBiDiMode (see page 4946)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	TrackButton (see page 4948)	Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

48.1.9.2 TPopupMenu.Create Constructor

Creates an instance of TPopupMenu (see page 4935).

Pascal

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

C++

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

Description

Call Create to create to create a pop-up menu at runtime. Pop-up menus placed in forms or data modules at design time are created automatically.

See Also

Create (see page 4839)

48.1.9.3 TPopupMenu.Destroy Destructor

Destroys an instance of TPopupMenu (see page 4935).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPopupMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPopupMenu (see page 4935) reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

48.1.9.4 TPopupMenu Methods

The methods of the TPopupMenu class are listed here.

Protected Methods

	Name	Description
	DoPopup (see page 4939)	Generates an OnPopup (see page 4949) event.
	SetPopupPoint (see page 4940)	This is SetPopupPoint, a member of class TPopupMenu.
	UseRightToLeftAlignment (see page 4940)	Specifies whether the control's alignment is in a right-to-left mode.

Public Methods

	Name	Description
	Popup (see page 4939)	Displays the pop-up menu onscreen.

48.1.9.4.1 TPopupMenu.DoPopup Method

Generates an OnPopup (see page 4949) event.

Pascal

```
procedure DoPopup(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DoPopup(TObject * Sender);
```

Description

DoPopup is called automatically just before the pop-up menu appears to generate an OnPopup (see page 4949) event. Override DoPopup in a descendant class to change the event.

See Also

OnPopup (see page 4949)

Popup (see page 4939)

48.1.9.4.2 TPopupMenu.Popup Method

Displays the pop-up menu onscreen.

Pascal

```
procedure Popup(X: Integer; Y: Integer); virtual;
```

C++

```
virtual __fastcall Popup(int X, int Y);
```

Description

Call Popup to bring up the pop-up menu. The menu appears at the screen coordinates indicated by the values (in pixels) of X and Y.

C++ Examples:

```
/*
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = false;
}

void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
          int X, int Y)
{
  PopupMenu1->Popup(Left + X, Top + Y);
}
```

Delphi Examples:

```
{
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

Alignment (see page 4941)

AutoPopup (see page 4942)

OnPopup (see page 4949)

48.1.9.4.3 TPopupMenu.SetPopupPoint Method

Pascal

```
procedure SetPopupPoint(APopupPoint: TPoint);
```

C++

```
__fastcall SetPopupPoint(TPoint APopupPoint);
```

Description

This is SetPopupPoint, a member of class TPopupMenu.

48.1.9.4.4 TPopupMenu.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the control's alignment is in a right-to-left mode. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns the setting from the control that activated the popup. If the popup was not activated by a control, UseRightToLeftAlignment returns the setting used in the global Application object.

48.1.9.5 TPopupMenu Properties

The properties of the TPopupMenu class are listed here.

Public Properties

	Name	Description
	PopupComponent (see page 4946)	Indicates the component that last displayed the popup menu in response to a right mouse click.
R	PopupPoint (see page 4948)	Indicates the location where the pop-up menu appears.

Published Properties

	Name	Description
	Alignment (see page 4941)	Determines where the pop-up menu appears when the user clicks the right mouse button.
	AutoHotkeys (see page 4941)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 4942)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoPopup (see page 4942)	Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.
	BiDiMode (see page 4943)	Determines the layout of the menu in Middle East locales.
	HelpContext (see page 4944)	Specifies a context ID for the entire pop-up menu.
	Images (see page 4944)	Lists the images that can appear beside individual menu items.
	MenuAnimation (see page 4945)	Specifies how the menu appears when it is first displayed.
	OwnerDraw (see page 4945)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 4946)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	TrackButton (see page 4948)	Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

48.1.9.5.1 TPopupMenu.Alignment Property

Determines where the pop-up menu appears when the user clicks the right mouse button.

Pascal

```
property Alignment: TPopupAlignment;
```

C++

```
__property TPopupAlignment Alignment;
```

Description

Set Alignment to control the placement of the popup menu. Alignment controls whether the menu appears to the left of the mouse, to the right of the mouse, or centered over the mouse.

By default, the menu appears with its top left corner under the mouse (paLeft).

See Also

AutoPopup (see page 4942)

OnPopup (see page 4949)

48.1.9.5.2 TPopupMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure

that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoHotkeys (see page 4900)

Items (see page 4852)

RethinkHotkeys (see page 4894)

48.1.9.5.3 TPopupMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoLineReduction (see page 4900)

Items (see page 4852)

RethinkLines (see page 4894)

48.1.9.5.4 TPopupMenu.AutoPopup Property

Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.

Pascal

```
property AutoPopup: Boolean;
```

C++

```
__property Boolean AutoPopup;
```

Description

Set AutoPopup to true to cause the pop-up menu to appear automatically when the user clicks the right mouse button on a control that has the pop-up menu specified as its PopupMenu property. The application key present on keyboards specifically designed for 32-bit Windows will also show this menu when pressed.

Set AutoPopup to false if the appearance of the pop-up menu should be controlled programmatically. To display a pop-up menu when AutoPopup is false, use the Popup (see page 4939) method.

C++ Examples:

```
/*
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = false;
}

void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
          int X, int Y)
{
  PopupMenu1->Popup(Left + X, Top + Y);
}
```

Delphi Examples:

```
{
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

Popup (see page 4939)

OnPopup (see page 4949)

PopupMenu

48.1.9.5.5 TPopupMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 4857)

IsRightToLeft (see page 4845)

48.1.9.5.6 TPopupMenu.HelpContext Property

Specifies a context ID for the entire pop-up menu.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

Use HelpContext to associate a help screen with the entire pop-up menu. Each screen in the Help system should have a unique context ID. When a component is selected in the application, pressing F1 displays the Help screen associated with the value of HelpContext.

To associate a help screen with the individual items in the pop-up menu, use the HelpContext property of the associated TMenuItem (see page 4869) objects.

See Also

GetHelpContext (see page 4844)

HelpFile (see page 1974)

HelpContext (see page 4915)

48.1.9.5.7 TPopupMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu

items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 4916)

Images

Items (see page 4852)

48.1.9.5.8 TPopupMenu.MenuAnimation Property

Specifies how the menu appears when it is first displayed.

Pascal

```
property MenuAnimation: TMenuAnimation;
```

C++

```
__property TMenuAnimation MenuAnimation;
```

Description

Use MenuAnimation to specify how the popup menu appears on the screen. The menu can simply appear as a whole, or can slide into view like a window shade being drawn from any edge.

Note: MenuAnimation only has an effect when running on Windows 98, NT 5.0, or later.

See Also

Popup (see page 4939)

48.1.9.5.9 TPopupMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 4929)

OnDrawItem (see page 4928)

Images (see page 4851)

48.1.9.5.10 TPopupMenu.ParentBiDiMode Property

Indicates whether the BiDiMode property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode setting and control the menu's layout independently.

See Also

BiDiMode (see page 4850)

48.1.9.5.11 TPopupMenu.PopupComponent Property

Indicates the component that last displayed the popup menu in response to a right mouse click.

Pascal

```
property PopupComponent: TComponent;
```

C++

```
__property TComponent * PopupComponent;
```

Description

Read PopupComponent to determine which control is currently using the popup menu. In applications where multiple controls share the same pop-up menu, use PopupComponent to determine which of them displayed the menu.

Set PopupComponent to associate a control with the menu before calling the Popup (see page 4939) method programmatically to bring up the pop-up menu.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
```

```
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
```

See Also

AutoPopup (see page 4942)

OnPopup (see page 4949)

Popup (see page 4939)

PopupMenu

48.1.9.5.12 TPopupMenu.PopupPoint Property

Indicates the location where the pop-up menu appears.

Pascal

```
property PopupPoint: TPoint;
```

C++

```
__property TPoint PopupPoint;
```

Description

This public property is used internally to locate the popup menu. Its value is set by the Popup (see page 4939) method.

See Also

Popup (see page 4939)

48.1.9.5.13 TPopupMenu.TrackButton Property

Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

Pascal

```
property TrackButton: TTrackButton;
```

C++

```
__property TTrackButton TrackButton;
```

Description

Use TrackButton to specify which mouse button activates the popup menu when it is associated with a toolbar button.

48.1.9.6 TPopupMenu Events

The events of the TPopupMenu class are listed here.

Published Events

	Name	Description
	OnChange (see page 4949)	Occurs when the menu changes.
	OnPopup (see page 4949)	Occurs just before the pop-up menu appears.

48.1.9.6.1 TPopupMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

See Also

Items (see page 4852)

48.1.9.6.2 TPopupMenu.OnPopup Event

Occurs just before the pop-up menu appears.

Pascal

```
property OnPopup: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPopup;
```

Description

Write an OnPopup event handler to take specific action just before the pop-up menu appears. For example, use an OnPopup event handler to set the Checked, Enabled, or Visible property of individual items in the menu so that they are appropriate to the PopupComponent (see page 4946).

C++ Examples:

```
/*
The following code enables the Copy item from the pop-up
menu if the edit control has a nonempty selection.
Note: A portion of the edit control text must be highlighted.
*/
```

```
void __fastcall TForm1::PopupMenu1Popup(TObject *Sender)
{
  Copy1->Enabled = (Edit1->SelLength > 0);
}

void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    ListBox1->Items->Add("'Copy' menu");
}

void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
          int X, int Y)
{
  PopupMenu1->Popup(X, Y);
}
```

Delphi Examples:

```
{
The following code enables the Copy item from the pop-up
menu if the edit control has a nonempty selection.
Note: A portion of the edit control text must be highlighted.
}
procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
  Copy1.Enabled := Edit1.SelLength > 0;
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
    ListBox1.Items.Add('"Copy" menu');
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

PopupComponent (see page 4946)

PopupMenu

48.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiSameCaption (see page 4951)	Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.
	DrawMenuItem (see page 4952)	This is function Menus.DrawMenuItem.
	GetHotkey (see page 4952)	Returns the last character in a string marked with an ampersand (&).
	IsAltGRPressed (see page 4952)	This is function Menus.IsAltGRPressed.
	NewItem (see page 4953)	Creates and initializes a menu item.
	NewLine (see page 4954)	Creates a menu item for a separator.
	NewMenu (see page 4955)	Creates and initializes a main menu.
	NewPopupMenu (see page 4956)	Creates and initializes a popup menu.
	NewSubMenu (see page 4957)	Creates and initializes a menu item that represents a submenu.
	SameCaption (see page 4958)	This is function Menus.SameCaption.
	ShortCut (see page 4958)	Creates a menu shortcut for a specified key combination.
	ShortCutFromMessage (see page 4959)	This is function Menus.ShortCutFromMessage.
	ShortCutToKey (see page 4960)	Returns the key code and shift state of a menu shortcut.
	ShortCutToText (see page 4961)	Converts a menu shortcut into a string that describes the shortcut.
	StripHotkey (see page 4962)	This is function Menus.StripHotkey.
	TextToShortCut (see page 4963)	Creates a menu shortcut from a text string.

48.2.1 Menus.AnsiSameCaption Function

Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.

Pascal

```
function AnsiSameCaption(const Text1: string; const Text2: string): Boolean;
```

C++

```
Boolean AnsiSameCaption(const UnicodeString Text1, const UnicodeString Text2);
```

File

Menus

Description

AnsiSameCaption compares Text1 to Text2, without case sensitivity and ignoring any ampersands (&) that are included to create accelerator characters. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

StringReplace

IsAccel

AnsiSameText

WideSameCaption

48.2.2 Menus.DrawMenuItem Function

Pascal

```
procedure DrawMenuItem(MenuItem: TMenuItem; ACanvas: TCanvas; ARect: TRect; State:
TOwnerDrawState);
```

C++

```
DrawMenuItem(TMenuItem MenuItem, TCanvas ACanvas, TRect ARect, TOwnerDrawState State);
```

File

Menus

Description

This is function Menus.DrawMenuItem.

48.2.3 Menus.GetHotkey Function

Returns the last character in a string marked with an ampersand (&).

Pascal

```
function GetHotkey(const Text: string): string;
```

C++

```
UnicodeString GetHotkey(const UnicodeString Text);
```

File

Menus

Description

GetHotKey searches Text, starting at the end, for a character that is identified as an accelerator key by a preceding ampersand (&). If it locates a character marked with an ampersand, GetHotKey returns that character. Otherwise, it returns an empty string.

See Also

AnsiSameCaption (see page 4951)

StripHotKey

IsAccel

48.2.4 Menus.IsAltGRPressed Function

Pascal

```
function IsAltGRPressed: boolean;
```

C++

```
boolean IsAltGRPressed();
```

File

Menus

Description

This is function Menus.IsAltGRPressed.

48.2.5 Menus.NewItem Function

Creates and initializes a menu item.

Pascal

```
function NewItem(const ACaption: string; AShortCut: TShortCut; AChecked: Boolean; AEnabled:
Boolean; AOnClick: TNotifyEvent; hCtx: THelpContext; const AName: string): TMenuItem;
```

C++

```
TMenuItem NewItem(const UnicodeString ACaption, TShortCut AShortCut, Boolean AChecked, Boolean
AEnabled, TNotifyEvent AOnClick, THelpContext hCtx, const UnicodeString AName);
```

File

Menus

Description

Call NewItem to create and initialize a menu item when constructing a menu programmatically.

The ACaption parameter specifies the string that should appear for the menu item.

The AShortCut parameter specifies any associated shortcut.

The AChecked parameter indicates whether the menu item appears with a check mark.

The AEnabled parameter specifies whether the menu item is enabled or grayed.

The AOnClick parameter specifies an event handler that executes when the user clicks the menu item.

The hCtx parameter specifies the help context ID for the menu item.

The AName parameter specifies the name of the menu item, which can be used to refer to the menu item in code.

The menu item returned by NewItem can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (see page 4955), NewPopupMenu (see page 4956), or NewSubMenu (see page 4957) function.

Warning: The item returned by NewItem does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 4869) do not free memory.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.  Click on the menu item
or type "Ctrl+N" to execute MyOnExecute.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
```

```
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
```

See Also

NewSubMenu (see page 4957)

NewLine (see page 4954)

NewMenu (see page 4955)

NewPopupMenu (see page 4956)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

48.2.6 Menus.NewLine Function

Creates a menu item for a separator.

Pascal

```
function NewLine: TMenuItem;
```

C++

```
TMenuItem NewLine();
```

File

Menus

Description

Call NewLine to create a separator that divides the items in a menu into logical groups. Once the menu item for the separator has been created using NewLine, it can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (

see page 4955), NewPopupMenu (see page 4956), or NewSubMenu (see page 4957) function.

Warning: The item returned by NewLine does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 4869) do not free memory.

See Also

NewItem (see page 4953)

NewPopupMenu (see page 4956)

NewSubMenu (see page 4957)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

48.2.7 Menus.NewMenu Function

Creates and initializes a main menu.

Pascal

```
function NewMenu(Owner: TComponent; const AName: string; const Items: array of TMenuItem):
TMainMenu;
```

C++

```
TMainMenu NewMenu(TComponent * Owner, const UnicodeString AName, const array of TMenuItem
Items);
```

File

Menus

Description

Call NewMenu to create and initialize a main menu programmatically. NewMenu returns the TMainMenu (see page 4828) component that represents the menu.

The Owner parameter specifies the component that is responsible for freeing the menu (typically the form).

The AName parameter specifies the name of the menu, which is used to refer to it in code.

The Items parameter is an array of menu items that make up the top level menu items. To create the menu items for the Items parameter, use the NewItem (see page 4953) function.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.  Click on the menu item
or type "Ctrl+N" to execute MyOnExecute.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
```

```
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
```

See Also

NewPopupMenu (see page 4956)

NewSubMenu (see page 4957)

NewItem (see page 4953)

48.2.8 Menus.NewPopupMenu Function

Creates and initializes a popup menu.

Pascal

```
function NewPopupMenu(Owner: TComponent; const AName: string; Alignment: TPopupAlignment;
AutoPopup: Boolean; const Items: array of TMenuItem): TPopupMenu;
```

C++

```
TPopupMenu NewPopupMenu(TComponent * Owner, const UnicodeString AName, TPopupAlignment
Alignment, Boolean AutoPopup, const array of TMenuItem Items);
```

File

Menus

Description

Call NewPopupMenu to create and initialize a popup menu programmatically. NewPopupMenu returns the TPopupMenu (see page 4935) component that represents the popup menu, but does not associate that popup menu with any control.

The Owner parameter specifies the component that is responsible for freeing the menu.

The AName parameter specifies the name of the popup menu, which is used to refer to it in code.

The Alignment parameter determines where the menu appears relative to the mouse when the user right clicks on the control that uses the menu.

The AutoPopup parameter specifies whether the menu should appear automatically when the user right clicks the associated control.

The Items parameter is an array of menu items that make up the top level menu items. To create the menu items for the Items parameter, use the NewItem (see page 4953) function.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

See Also

NewMenu (see page 4955)

NewSubMenu (see page 4957)

NewItem (see page 4953)

48.2.9 Menus.NewSubMenu Function

Creates and initializes a menu item that represents a submenu.

Pascal

```
function NewSubMenu(const ACaption: string; hCtx: THelpContext; const AName: string; const
Items: array of TMenuItem; AEnabled: Boolean = True): TMenuItem;
```

C++

```
TMenuItem NewSubMenu(const UnicodeString ACaption, THelpContext hCtx, const UnicodeString
AName, const array of TMenuItem Items, Boolean AEnabled = True);
```

File

Menus

Description

Call NewSubMenu to create a menu item that represents a submenu when constructing a menu programmatically at runtime.

The ACaption parameter indicates the string that appears in the parent menu for the submenu.

The hCtx parameter specifies the help context ID for the menu item.

The AName parameter specifies the name of the menu item, which can be used to refer to the menu item in code.

The Items parameter specifies an array of menu items that make up the submenu and that appear when the user clicks the submenu item in the parent menu.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

The AEnabled parameter specifies whether the menu item that appears in the parent menu is enabled.

Once a submenu has been created using NewSubMenu, it can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (see page 4955), NewPopupMenu (see page 4956), or NewSubMenu function.

Warning: The item returned by NewSubMenu does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 4869) do not free memory.

See Also

NewItem (see page 4953)

TMenuItem (see page 4869)

NewLine (see page 4954)

NewMenu (see page 4955)

NewPopupMenu (see page 4956)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

TMenuItem (see page 4869)

48.2.10 Menus.SameCaption Function

Pascal

```
function SameCaption(const Text1: string; const Text2: string): Boolean;
```

C++

```
Boolean SameCaption(const UnicodeString Text1, const UnicodeString Text2);
```

File

Menus

Description

This is function Menus.SameCaption.

48.2.11 Menus.ShortCut Function

Creates a menu shortcut for a specified key combination.

Pascal

```
function ShortCut(Key: Word; Shift: TShiftState): TShortCut;
```

C++

```
TShortCut ShortCut(Word Key, TShiftState Shift);
```

File

Menus

Description

Call ShortCut to create a menu shortcut at runtime.

Specify a Key value using a virtual key code (Menus (see page 4825) version) or Qt key code (QMenus version).

Specify the values for the Shift, Control, and Alt keys by using a set of type TShiftState. For example, to specify the Shift key, use the set [ssShift]. To specify a Shift and Ctrl key combination, use the set [ssShift, ssCtrl].

Assign the value returned by ShortCut as the value of the ShortCut property of a menu item.

Note: TextToShortCut (see page 4963) can also create a shortcut. TextToShortCut (see page 4963) converts a string to a shortcut, so it can be used to let the user specify the shortcut. However, TextToShortCut (see page 4963) executes much more slowly than ShortCut. Use ShortCut whenever possible.

C++ Examples:

```
/*
This example creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
*/
void __fastcall TForm1::OpenCommandClick(TObject *Sender)
{
  MessageDlg(
    "The Open command has been selected.",
    mtInformation,
    TMsgDlgButtons() << mbOK,
    0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OpenCommand->ShortCut =
    ShortCut(Word('O'), TShiftState() << ssCtrl);
}
```

Delphi Examples:

```
{
This example creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenCommand.ShortCut := Menus.ShortCut(Word('O'), [ssCtrl]);
end;

procedure TForm1.OpenCommandClick(Sender: TObject);
begin
  MessageDlg('The Open command has been selected.', mtInformation, [mbOk], 0);
end;
```

See Also

ShortCutToKey (see page 4960)

ShortCutToText (see page 4961)

TextToShortCut (see page 4963)

TMenuItem (see page 4869)

48.2.12 Menus.ShortCutFromMessage Function

Pascal

```
function ShortCutFromMessage(Message: TWMKey): TShortCut;
```

C++

```
TShortCut ShortCutFromMessage(TWMKey Message);
```

File

Menus

Description

This is function Menus.ShortCutFromMessage.

48.2.13 Menus.ShortCutToKey Function

Returns the key code and shift state of a menu shortcut.

Pascal

```
procedure ShortCutToKey(ShortCut: TShortCut; var Key: Word; var Shift: TShiftState);
```

C++

```
ShortCutToKey(TShortCut ShortCut, Word Key, TShiftState Shift);
```

File

Menus

Description

Call ShortCutToKey to parse a menu shortcut into its virtual key code and shift state parts.

ShortCut (see page 4958) is the value of the menu shortcut.

Key returns the virtual key code (Menus (see page 4825) version) or Qt key code (QMenus version) for the key.

Shirt returns the associated combination of Shift, Ctrl, and Alt keys.

C++ Examples:

```
/*
The following code redefines the ShortCut of CloseCommand if
the original short cut used the [ssCtrl] shift state.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Word TheKey;

  TShiftState TheShiftState;

  ShortCutToKey(CloseCommand->ShortCut, TheKey, TheShiftState);
  if (TheShiftState.Contains(ssCtrl))
    CloseCommand->ShortCut = ShortCut(Word('C'), TShiftState() << ssShift);}

void __fastcall TForm1::CloseCommandClick(TObject *Sender)
{
  MessageDlg(
    "The Close command has been selected.",
    mtInformation,
    TMsgDlgButtons() << mbOK,
    0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CloseCommand->ShortCut =
    ShortCut(Word('C'), TShiftState() << ssCtrl);
}
```

Delphi Examples:

```
{
The following code redefines the ShortCut of CloseCommand if
```

```
the original short cut used the [ssCtrl] shift state.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TheKey: Word;
  TheShiftState: TShiftState;
begin
  Menus.ShortCutToKey(CloseCommand.ShortCut, TheKey, TheShiftState);
  if TheShiftState = [ssCtrl] then
   CloseCommand.ShortCut := ShortCut(Word('C'), [ssShift]);
end;

procedure TForm1.CloseCommandClick(Sender: TObject);
begin
  MessageDlg('The Close command has been selected.', mtInformation, [mbOk], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CloseCommand.ShortCut := Menus.ShortCut(Word('C'), [ssCtrl]);
end;
```

See Also

ShortCut (see page 4958)

ShortCutToText (see page 4961)

TextToShortCut (see page 4963)

TMenuItem (see page 4869)

48.2.14 Menus.ShortCutToText Function

Converts a menu shortcut into a string that describes the shortcut.

Pascal

```
function ShortCutToText(ShortCut: TShortCut): string;
```

C++

```
UnicodeString ShortCutToText(TShortCut ShortCut);
```

File

Menus

Description

Call ShortCutToText to obtain a text description of the shortcut associated with a menu item. Use this function when the application needs to display a description of a shortcut to the user.

C++ Examples:

```
/*
This line creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->Text = ShortCutToText(OpenCommand->ShortCut);}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  OpenCommand->ShortCut =
    ShortCut(Word('O'), TShiftState() << ssCtrl);
}

void __fastcall TForm1::OpenCommandClick(TObject *Sender)
{
  MessageDlg(
    "The Open command has been selected.",
    mtInformation,
    TMsgDlgButtons() << mbOK,
    0);
}
```

Delphi Examples:

```
{
This line creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := Menus.ShortCutToText(OpenCommand.ShortCut);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenCommand.ShortCut := Menus.ShortCut(Word('O'), [ssCtrl]);
end;

procedure TForm1.OpenCommandClick(Sender: TObject);
begin
  MessageDlg('The Open command has been selected.', mtInformation, [mbOk], 0);
end;
```

See Also

ShortCut (see page 4958)

ShortCutToKey (see page 4960)

TextToShortCut (see page 4963)

TMenuItem (see page 4869)

48.2.15 Menus.StripHotkey Function

Pascal

```
function StripHotkey(const Text: string): string;
```

C++

```
UnicodeString StripHotkey(const UnicodeString Text);
```

File

Menus

Description

This is function Menus.StripHotkey.

48.2.16 Menus.TextToShortCut Function

Creates a menu shortcut from a text string.

Pascal

```
function TextToShortCut(Text: string): TShortCut;
```

C++

```
TShortCut TextToShortCut(UnicodeString Text);
```

File

Menus

Description

Call TextToShortCut to create a menu shortcut from a descriptive string. For example, an application can allow users to specify the value of a shortcut in an edit box control, and then call TextToShortCut to create a menu shortcut for the value entered by the user.

Note: TextToShortCut executes slowly. When possible, use the ShortCut (see page 4958) function to create a menu shortcut instead.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.  Click on the menu item
or type "Ctrl+N" to execute MyOnExecute.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
```

```
end;
```

See Also

ShortCut (see page 4958)

ShortCutToText (see page 4961)

TMenuItem (see page 4869)

48.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFindItemKind (see page 4965)	TFindItemKind indicates the criterion to use when locating menu items.
	TMenuAnimations (see page 4966)	TMenuAnimation (see page 4971) and TMenuAnimations indicate how a popup menu appears on screen.
	TMenuBreak (see page 4966)	TMenuBreak indicates the criterion to use when locating menu items.
	TMenuItemAutoFlag (see page 4967)	TMenuItemAutoFlag and TMenuAutoFlag (see page 4972) Indicate when menu items are automatically maintained.
	TPopupAlignment (see page 4968)	TPopupAlignment represents the position of a popup menu relative to the mouse.
	TTrackButton (see page 4968)	TTrackButton specifies which mouse button activates a tool button menu.

48.3.1 Menus.TFindItemKind Enumeration

TFindItemKind indicates the criterion to use when locating menu items.

Pascal

```
TFindItemKind = (
  fkCommand,
  fkHandle,
  fkShortCut
);
```

C++

```
enum TFindItemKind {
  fkCommand,
  fkHandle,
  fkShortCut
};
```

File

Menus

Description

TFindItemKind values indicate the criterion to use when locating a menu item. The following table gives the possible values:

Value	Meaning
fkID	Identify the item by its Windows command ID (parameter to WM_COMMAND (see page 5148) message).
fkHandle	Identify the item by its pop-up menu handle.
fkShortCut	Identify the item by its menu shortcut key code.

48.3.2 Menus.TMenuAnimations Enumeration

TMenuAnimation (see page 4971) and TMenuAnimations indicate how a popup menu appears on screen.

Pascal

```
TMenuAnimations = (
  maLeftToRight,
  maRightToLeft,
  maTopToBottom,
  maBottomToTop,
  maNone
);
```

C++

```
enum TMenuAnimations {
  maLeftToRight,
  maRightToLeft,
  maTopToBottom,
  maBottomToTop,
  maNone
};
```

File

Menus

Description

TMenuAnimations indicates the direction in which a menu can "slide" into view. TMenuAnimation (see page 4971) is a set of TMenuAnimations values, which can be used to specify more than one direction (for a popup menu that slides in at a diagonal). Note that some TMenuAnimations values are mutually exclusive (for example maLeftToRight and maRightToLeft).

Value	Meaning
maLeftToRight	The popup menu appears like a window shade that pulls from left to right.
maRightToLeft	The popup menu appears like a window shade that pulls from right to left.
maTopToBottom	The popup menu appears like a window shade that pulls from top to bottom.
maBottomToTop	The popup menu appears like a window shade that pulls from bottom to top.
maNone	The popup menu appears as a whole with no animation effects.

See Also

TPopupMenu

48.3.3 Menus.TMenuBreak Enumeration

TMenuBreak indicates the criterion to use when locating menu items.

Pascal

```
TMenuBreak = (
  mbNone,
  mbBreak,
  mbBarBreak
```

```
);
```

C++

```
enum TMenuBreak {
  mbNone,
  mbBreak,
  mbBarBreak
};
```

File

Menus

Description

TMenuBreak values indicate the criterion to use when locating a menu item. The following table gives the possible values:

Value	Meaning
mbNone	No menu breaking occurs.
mbBarBreak	The Parent menu breaks into another column with the menu item appearing at the top of the new column. A bar separates the new and the old columns.
mbBreak	The Parent menu breaks into another column with the menu item appearing at the top of the new column. Only space separates the new and the old columns.

48.3.4 Menus.TMenuItemAutoFlag Enumeration

TMenuItemAutoFlag and TMenuAutoFlag (see page 4972) Indicate when menu items are automatically maintained.

Pascal

```
TMenuItemAutoFlag = (
  maAutomatic,
  maManual,
  maParent
);
```

C++

```
enum TMenuItemAutoFlag {
  maAutomatic,
  maManual,
  maParent
};
```

File

Menus

Description

TMenuItemAutoFlag indicates when a submenu is checked for problems and updated. TMenuAutoFlag (see page 4972) provides the same information for top-level menu items. The following table lists the possible values:

Values	Meaning
maAutomatic	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected when the menu is displayed at runtime.
maManual	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected only when the application specifically calls a method of the menu item to do so.

maParent	Automatic checking and correction of the immediate submenu follows the same rule as the menu item's parent. This value is not applicable to menus, which have no parent menu item.

See Also

AutoHotkeys

AutoLineReduction

48.3.5 Menus.TPopupAlignment Enumeration

TPopupAlignment represents the position of a popup menu relative to the mouse.

Pascal

```
TPopupAlignment = (
  paLeft,
  paRight,
  paCenter
);
```

C++

```
enum TPopupAlignment {
  paLeft,
  paRight,
  paCenter
};
```

File

Menus

Description

These are the possible values:

Value	Meaning
paLeft	The pop-up menu appears with its top left corner under the mouse.
paCenter	The pop-up menu appears with the top center of the menu under the mouse.
paRight	The pop-up menu appears with its top right corner under the mouse.

48.3.6 Menus.TTrackButton Enumeration

TTrackButton specifies which mouse button activates a tool button menu.

Pascal

```
TTrackButton = (
  tbRightButton,
  tbLeftButton
);
```

C++

```
enum TTrackButton {
```

```
    tbRightButton,
    tbLeftButton
};
```

File

Menus

Description

TTrackButton can have one of the following values:

Value	Meaning
tbRightButton	The right mouse button activates the tool button menu.
tbLeftButton	The left mouse button activates the tool button menu.

48.4 Types

The following table lists types in this documentation.

Types

Name	Description
PMenuItem (see page 4970)	This is type Menus.PMenuItem.
TAdvancedMenuDrawItemEvent (see page 4970)	TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuActionLinkClass (see page 4971)	This is type Menus.TMenuActionLinkClass.
TMenuAnimation (see page 4971)	TMenuAnimation and TMenuAnimations (see page 4966) indicate how a popup menu appears on screen.
TMenuAutoFlag (see page 4972)	TMenuItemAutoFlag (see page 4967) and TMenuAutoFlag Indicate when menu items are automatically maintained.
TMenuChangeEvent (see page 4972)	This is type Menus.TMenuChangeEvent.
TMenuDrawItemEvent (see page 4973)	TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuHandle (see page 4973)	This is type Menus.TMenuHandle.
TMenuMeasureItemEvent (see page 4973)	TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

48.4.1 Menus.PMenuItem Type

Pascal

```
PMenuItem = ^TMenuItem;
```

C++

```
^TMenuItem PMenuItem;
```

File

Menus

Description

This is type Menus.PMenuItem.

48.4.2 Menus.TAdvancedMenuDrawItemEvent Type

TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.

Pascal

```
TAdvancedMenuDrawItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; ARect: TRect;
State: TOwnerDrawState) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; ARect: TRect; State: TOwnerDrawState) (
TAdvancedMenuDrawItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 4869) instance that represents the menu item to be drawn.

ACanvas provides a drawing surface on which to draw the menu item.

ARect indicates the boundaries (in pixels) of the menu item on ACanvas.

State is a set indicating various aspects of the menu items state such as whether it is selected, checked, grayed, and so on.

48.4.3 Menus.TMenuActionLinkClass Type

Pascal

```
TMenuActionLinkClass = class of TMenuActionLink;
```

C++

```
TMenuActionLink TMenuActionLinkClass;
```

File

Menus

Description

This is type Menus.TMenuActionLinkClass.

48.4.4 Menus.TMenuAnimation Type

TMenuAnimation and TMenuAnimations (see page 4966) indicate how a popup menu appears on screen.

Pascal

```
TMenuAnimation = set of TMenuAnimations;
```

C++

```
TMenuAnimations TMenuAnimation;
```

File

Menus

Description

TMenuAnimations (see page 4966) indicates the direction in which a menu can "slide" into view. TMenuAnimation is a set of TMenuAnimations (see page 4966) values, which can be used to specify more than one direction (for a popup menu that slides in at a diagonal). Note that some TMenuAnimations (see page 4966) values are mutually exclusive (for example maLeftToRight and maRightToLeft).

Value	Meaning
maLeftToRight	The popup menu appears like a window shade that pulls from left to right.
maRightToLeft	The popup menu appears like a window shade that pulls from right to left.

maTopToBottom	The popup menu appears like a window shade that pulls from top to bottom.
maBottomToTop	The popup menu appears like a window shade that pulls from bottom to top.
maNone	The popup menu appears as a whole with no animation effects.

See Also

TPopupMenu

48.4.5 Menus.TMenuAutoFlag Type

TMenuItemAutoFlag (see page 4967) and TMenuAutoFlag Indicate when menu items are automatically maintained.

Pascal

```
TMenuAutoFlag = maAutomatic..maManual;
```

C++

```
maAutomatic..maManual TMenuAutoFlag;
```

File

Menus

Description

TMenuItemAutoFlag (see page 4967) indicates when a submenu is checked for problems and updated. TMenuAutoFlag provides the same information for top-level menu items. The following table lists the possible values:

Values	Meaning
maAutomatic	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected when the menu is displayed at runtime.
maManual	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected only when the application specifically calls a method of the menu item to do so.
maParent	Automatic checking and correction of the immediate submenu follows the same rule as the menu item's parent. This value is not applicable to menus, which have no parent menu item.

See Also

AutoHotkeys

AutoLineReduction

48.4.6 Menus.TMenuChangeEvent Type

Pascal

```
TMenuChangeEvent = procedure (Sender: TObject; Source: TMenuItem; Rebuild: Boolean) of object;
```

C++

```
(Sender: TObject; Source: TMenuItem; Rebuild: Boolean) ( TMenuChangeEvent)();
```

File

Menus

Description

This is type Menus.TMenuChangeEvent.

48.4.7 Menus.TMenuDrawItemEvent Type

TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.

Pascal

```
TMenuDrawItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; ARect: TRect; Selected:
Boolean) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; ARect: TRect; Selected: Boolean) ( TMenuDrawItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 4869) instance that represents the menu item to be drawn.

ACanvas provides a drawing surface on which to draw the menu item.

ARect indicates the boundaries (in pixels) of the menu item on ACanvas.

Selected indicates whether the menu item is selected (that is, if it will be clicked if the user presses return)

48.4.8 Menus.TMenuHandle Type

Pascal

```
TMenuHandle = HMENU;
```

C++

```
HMENU TMenuHandle;
```

File

Menus

Description

This is type Menus.TMenuHandle.

48.4.9 Menus.TMenuMeasureItemEvent Type

TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

Pascal

```
TMenuMeasureItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; var Width, Height:
Integer) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; var Width, Height: Integer) ( TMenuMeasureItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 4869) instance that represents the menu item to be measured.

ACanvas represents the canvas where the menu item will be drawn. Use its methods, if necessary, to determine the size of the menu item.

Width and Height return the dimensions, in pixels, that should be used to draw the menu item.

48.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
PopupList (see page 4975)	This is variable Menus.PopupList.
ShortCutItems (see page 4975)	This is variable Menus.ShortCutItems.
ValidMenuHotkeys (see page 4975)	This is variable Menus.ValidMenuHotkeys.

48.5.1 Menus.PopupList Variable

Pascal

```
PopupList: TPopupList;
```

C++

```
TPopupList PopupList;
```

File

Menus

Description

This is variable Menus.PopupList.

48.5.2 Menus.ShortCutItems Variable

Pascal

```
ShortCutItems: TMenuItemStack;
```

C++

```
TMenuItemStack ShortCutItems;
```

File

Menus

Description

This is variable Menus.ShortCutItems.

48.5.3 Menus.ValidMenuHotkeys Variable

Pascal

```
ValidMenuHotkeys: string = '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

C++

```
UnicodeString ValidMenuHotkeys = '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

File

Menus

Description

This is variable Menus.ValidMenuHotkeys.

48.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cDialogSuffix (see page 4977)	This is constant Menus.cDialogSuffix.
cHotkeyPrefix (see page 4977)	This is constant Menus.cHotkeyPrefix.
cLineCaption (see page 4977)	This is constant Menus.cLineCaption.

48.6.1 Menus.cDialogSuffix Constant

Pascal

```
cDialogSuffix = '...';
```

C++

```
cDialogSuffix = '...';
```

File

Menus

Description

This is constant Menus.cDialogSuffix.

48.6.2 Menus.cHotkeyPrefix Constant

Pascal

```
cHotkeyPrefix = '&';
```

C++

```
cHotkeyPrefix = '&';
```

File

Menus

Description

This is constant Menus.cHotkeyPrefix.

48.6.3 Menus.cLineCaption Constant

Pascal

```
cLineCaption = '-';
```

C++

```
cLineCaption = '-';
```

File

Menus

Description

This is constant Menus.cLineCaption.

49 Messages Namespace

This is namespace Messages.

Topics

Name	Description
Functions (see page 4991)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4995)	The following table lists structs, records, enums in this documentation.
Types (see page 5050)	The following table lists types in this documentation.
Constants (see page 5077)	The following table lists constants in this documentation.

Constants

Name	Description
BM_CLICK (see page 5084)	This is constant Messages.BM_CLICK.
BM_GETCHECK (see page 5085)	This is constant Messages.BM_GETCHECK.
BM_GETIMAGE (see page 5085)	This is constant Messages.BM_GETIMAGE.
BM_GETSTATE (see page 5085)	This is constant Messages.BM_GETSTATE.
BM_SETCHECK (see page 5086)	This is constant Messages.BM_SETCHECK.
BM_SETIMAGE (see page 5086)	This is constant Messages.BM_SETIMAGE.
BM_SETSTATE (see page 5086)	This is constant Messages.BM_SETSTATE.
BM_SETSTYLE (see page 5087)	This is constant Messages.BM_SETSTYLE.
BN_CLICKED (see page 5087)	This is constant Messages.BN_CLICKED.
BN_DBLCLK (see page 5087)	This is constant Messages.BN_DBLCLK.
BN_DISABLE (see page 5088)	This is constant Messages.BN_DISABLE.
BN_DOUBLECLICKED (see page 5088)	This is constant Messages.BN_DOUBLECLICKED.
BN_HILITE (see page 5088)	This is constant Messages.BN_HILITE.
BN_KILLFOCUS (see page 5089)	This is constant Messages.BN_KILLFOCUS.
BN_PAINT (see page 5089)	This is constant Messages.BN_PAINT.
BN_PUSHED (see page 5089)	This is constant Messages.BN_PUSHED.
BN_SETFOCUS (see page 5090)	This is constant Messages.BN_SETFOCUS.
BN_UNHILITE (see page 5090)	This is constant Messages.BN_UNHILITE.
BN_UNPUSHED (see page 5090)	This is constant Messages.BN_UNPUSHED.
CB_ADDSTRING (see page 5091)	This is constant Messages.CB_ADDSTRING.
CB_DELETESTRING (see page 5091)	This is constant Messages.CB_DELETESTRING.
CB_DIR (see page 5091)	This is constant Messages.CB_DIR.
CB_FINDSTRING (see page 5092)	This is constant Messages.CB_FINDSTRING.
CB_FINDSTRINGEXACT (see page 5092)	This is constant Messages.CB_FINDSTRINGEXACT.

CB_GETCOUNT (see page 5092)	This is constant Messages.CB_GETCOUNT.
CB_GETCURSEL (see page 5093)	This is constant Messages.CB_GETCURSEL.
CB_GETDROPPEDCONTROLRECT (see page 5093)	This is constant Messages.CB_GETDROPPEDCONTROLRECT.
CB_GETDROPPEDSTATE (see page 5093)	This is constant Messages.CB_GETDROPPEDSTATE.
CB_GETDROPPEDWIDTH (see page 5094)	This is constant Messages.CB_GETDROPPEDWIDTH.
CB_GETEDITSEL (see page 5094)	This is constant Messages.CB_GETEDITSEL.
CB_GETEXTENDEDUI (see page 5094)	This is constant Messages.CB_GETEXTENDEDUI.
CB_GETHORIZONTALEXTENT (see page 5095)	This is constant Messages.CB_GETHORIZONTALEXTENT.
CB_GETITEMDATA (see page 5095)	This is constant Messages.CB_GETITEMDATA.
CB_GETITEMHEIGHT (see page 5095)	This is constant Messages.CB_GETITEMHEIGHT.
CB_GETLBTEXT (see page 5096)	This is constant Messages.CB_GETLBTEXT.
CB_GETLBTEXTLEN (see page 5096)	This is constant Messages.CB_GETLBTEXTLEN.
CB_GETLOCALE (see page 5096)	This is constant Messages.CB_GETLOCALE.
CB_GETTOPINDEX (see page 5097)	This is constant Messages.CB_GETTOPINDEX.
CB_INITSTORAGE (see page 5097)	This is constant Messages.CB_INITSTORAGE.
CB_INSERTSTRING (see page 5097)	This is constant Messages.CB_INSERTSTRING.
CB_LIMITTEXT (see page 5098)	This is constant Messages.CB_LIMITTEXT.
CB_MSGMAX (see page 5098)	This is constant Messages.CB_MSGMAX.
CB_RESETCONTENT (see page 5098)	This is constant Messages.CB_RESETCONTENT.
CB_SELECTSTRING (see page 5099)	This is constant Messages.CB_SELECTSTRING.
CB_SETCURSEL (see page 5099)	This is constant Messages.CB_SETCURSEL.
CB_SETDROPPEDWIDTH (see page 5099)	This is constant Messages.CB_SETDROPPEDWIDTH.
CB_SETEDITSEL (see page 5100)	This is constant Messages.CB_SETEDITSEL.
CB_SETEXTENDEDUI (see page 5100)	This is constant Messages.CB_SETEXTENDEDUI.
CB_SETHORIZONTALEXTENT (see page 5100)	This is constant Messages.CB_SETHORIZONTALEXTENT.
CB_SETITEMDATA (see page 5101)	This is constant Messages.CB_SETITEMDATA.
CB_SETITEMHEIGHT (see page 5101)	This is constant Messages.CB_SETITEMHEIGHT.
CB_SETLOCALE (see page 5101)	This is constant Messages.CB_SETLOCALE.
CB_SETTOPINDEX (see page 5102)	This is constant Messages.CB_SETTOPINDEX.
CB_SHOWDROPDOWN (see page 5102)	This is constant Messages.CB_SHOWDROPDOWN.
CBN_CLOSEUP (see page 5102)	This is constant Messages.CBN_CLOSEUP.
CBN_DBLCLK (see page 5103)	This is constant Messages.CBN_DBLCLK.
CBN_DROPDOWN (see page 5103)	This is constant Messages.CBN_DROPDOWN.
CBN_EDITCHANGE (see page 5103)	This is constant Messages.CBN_EDITCHANGE.
CBN_EDITUPDATE (see page 5104)	This is constant Messages.CBN_EDITUPDATE.
CBN_ERRSPACE (see page 5104)	This is constant Messages.CBN_ERRSPACE.
CBN_KILLFOCUS (see page 5104)	This is constant Messages.CBN_KILLFOCUS.
CBN_SELCHANGE (see page 5105)	This is constant Messages.CBN_SELCHANGE.
CBN_SELENDCANCEL (see page 5105)	This is constant Messages.CBN_SELENDCANCEL.
CBN_SELENDOK (see page 5105)	This is constant Messages.CBN_SELENDOK.
CBN_SETFOCUS (see page 5106)	This is constant Messages.CBN_SETFOCUS.
DM_GETDEFID (see page 5106)	This is constant Messages.DM_GETDEFID.
DM_REPOSITION (see page 5106)	This is constant Messages.DM_REPOSITION.
DM_SETDEFID (see page 5107)	This is constant Messages.DM_SETDEFID.
EM_CANUNDO (see page 5107)	This is constant Messages.EM_CANUNDO.
EM_CHARFROMPOS (see page 5107)	This is constant Messages.EM_CHARFROMPOS.
EM_EMPTYUNDOBUFFER (see page 5108)	This is constant Messages.EM_EMPTYUNDOBUFFER.
EM_FMTLINES (see page 5108)	This is constant Messages.EM_FMTLINES.
EM_GETFIRSTVISIBLELINE (see page 5108)	This is constant Messages.EM_GETFIRSTVISIBLELINE.
EM_GETHANDLE (see page 5109)	This is constant Messages.EM_GETHANDLE.
EM_GETIMESTATUS (see page 5109)	This is constant Messages.EM_GETIMESTATUS.
EM_GETLIMITTEXT (see page 5109)	This is constant Messages.EM_GETLIMITTEXT.
EM_GETLINE (see page 5110)	This is constant Messages.EM_GETLINE.
EM_GETLINECOUNT (see page 5110)	This is constant Messages.EM_GETLINECOUNT.

EM_GETMARGINS (see page 5110)	This is constant Messages.EM_GETMARGINS.
EM_GETMODIFY (see page 5111)	This is constant Messages.EM_GETMODIFY.
EM_GETPASSWORDCHAR (see page 5111)	This is constant Messages.EM_GETPASSWORDCHAR.
EM_GETRECT (see page 5111)	This is constant Messages.EM_GETRECT.
EM_GETSEL (see page 5112)	This is constant Messages.EM_GETSEL.
EM_GETTHUMB (see page 5112)	This is constant Messages.EM_GETTHUMB.
EM_GETWORDBREAKPROC (see page 5112)	This is constant Messages.EM_GETWORDBREAKPROC.
EM_LIMITTEXT (see page 5113)	This is constant Messages.EM_LIMITTEXT.
EM_LINEFROMCHAR (see page 5113)	This is constant Messages.EM_LINEFROMCHAR.
EM_LINEINDEX (see page 5113)	This is constant Messages.EM_LINEINDEX.
EM_LINELENGTH (see page 5114)	This is constant Messages.EM_LINELENGTH.
EM_LINESCROLL (see page 5114)	This is constant Messages.EM_LINESCROLL.
EM_POSFROMCHAR (see page 5114)	This is constant Messages.EM_POSFROMCHAR.
EM_REPLACESEL (see page 5115)	This is constant Messages.EM_REPLACESEL.
EM_SCROLL (see page 5115)	This is constant Messages.EM_SCROLL.
EM_SCROLLCARET (see page 5115)	This is constant Messages.EM_SCROLLCARET.
EM_SETHANDLE (see page 5116)	This is constant Messages.EM_SETHANDLE.
EM_SETIMESTATUS (see page 5116)	This is constant Messages.EM_SETIMESTATUS.
EM_SETLIMITTEXT (see page 5116)	This is constant Messages.EM_SETLIMITTEXT.
EM_SETMARGINS (see page 5117)	This is constant Messages.EM_SETMARGINS.
EM_SETMODIFY (see page 5117)	This is constant Messages.EM_SETMODIFY.
EM_SETPASSWORDCHAR (see page 5117)	This is constant Messages.EM_SETPASSWORDCHAR.
EM_SETREADONLY (see page 5118)	This is constant Messages.EM_SETREADONLY.
EM_SETRECT (see page 5118)	This is constant Messages.EM_SETRECT.
EM_SETRECTNP (see page 5118)	This is constant Messages.EM_SETRECTNP.
EM_SETSEL (see page 5119)	This is constant Messages.EM_SETSEL.
EM_SETTABSTOPS (see page 5119)	This is constant Messages.EM_SETTABSTOPS.
EM_SETWORDBREAKPROC (see page 5119)	This is constant Messages.EM_SETWORDBREAKPROC.
EM_UNDO (see page 5120)	This is constant Messages.EM_UNDO.
EN_CHANGE (see page 5120)	This is constant Messages.EN_CHANGE.
EN_ERRSPACE (see page 5120)	This is constant Messages.EN_ERRSPACE.
EN_HSCROLL (see page 5121)	This is constant Messages.EN_HSCROLL.
EN_KILLFOCUS (see page 5121)	This is constant Messages.EN_KILLFOCUS.
EN_MAXTEXT (see page 5121)	This is constant Messages.EN_MAXTEXT.
EN_SETFOCUS (see page 5122)	This is constant Messages.EN_SETFOCUS.
EN_UPDATE (see page 5122)	This is constant Messages.EN_UPDATE.
EN_VSCROLL (see page 5122)	This is constant Messages.EN_VSCROLL.
LB_ADDFILE (see page 5123)	This is constant Messages.LB_ADDFILE.
LB_ADDSTRING (see page 5123)	This is constant Messages.LB_ADDSTRING.
LB_DELETESTRING (see page 5123)	This is constant Messages.LB_DELETESTRING.
LB_DIR (see page 5124)	This is constant Messages.LB_DIR.
LB_FINDSTRING (see page 5124)	This is constant Messages.LB_FINDSTRING.
LB_FINDSTRINGEXACT (see page 5124)	This is constant Messages.LB_FINDSTRINGEXACT.
LB_GETANCHORINDEX (see page 5125)	This is constant Messages.LB_GETANCHORINDEX.
LB_GETCARETINDEX (see page 5125)	This is constant Messages.LB_GETCARETINDEX.
LB_GETCOUNT (see page 5125)	This is constant Messages.LB_GETCOUNT.
LB_GETCURSEL (see page 5126)	This is constant Messages.LB_GETCURSEL.
LB_GETHORIZONTALEXTENT (see page 5126)	This is constant Messages.LB_GETHORIZONTALEXTENT.
LB_GETITEMDATA (see page 5126)	This is constant Messages.LB_GETITEMDATA.
LB_GETITEMHEIGHT (see page 5127)	This is constant Messages.LB_GETITEMHEIGHT.
LB_GETITEMRECT (see page 5127)	This is constant Messages.LB_GETITEMRECT.
LB_GETLOCALE (see page 5127)	This is constant Messages.LB_GETLOCALE.
LB_GETSEL (see page 5128)	This is constant Messages.LB_GETSEL.
LB_GETSELCOUNT (see page 5128)	This is constant Messages.LB_GETSELCOUNT.

LB_GETSELITEMS (see page 5128)	This is constant Messages.LB_GETSELITEMS.
LB_GETTEXT (see page 5129)	This is constant Messages.LB_GETTEXT.
LB_GETTEXTLEN (see page 5129)	This is constant Messages.LB_GETTEXTLEN.
LB_GETTOPINDEX (see page 5129)	This is constant Messages.LB_GETTOPINDEX.
LB_INITSTORAGE (see page 5130)	This is constant Messages.LB_INITSTORAGE.
LB_INSERTSTRING (see page 5130)	This is constant Messages.LB_INSERTSTRING.
LB_ITEMFROMPOINT (see page 5130)	This is constant Messages.LB_ITEMFROMPOINT.
LB_MSGMAX (see page 5131)	This is constant Messages.LB_MSGMAX.
LB_RESETCONTENT (see page 5131)	This is constant Messages.LB_RESETCONTENT.
LB_SELECTSTRING (see page 5131)	This is constant Messages.LB_SELECTSTRING.
LB_SELITEMRANGE (see page 5132)	This is constant Messages.LB_SELITEMRANGE.
LB_SELITEMRANGEEX (see page 5132)	This is constant Messages.LB_SELITEMRANGEEX.
LB_SETANCHORINDEX (see page 5132)	This is constant Messages.LB_SETANCHORINDEX.
LB_SETCARETINDEX (see page 5133)	This is constant Messages.LB_SETCARETINDEX.
LB_SETCOLUMNWIDTH (see page 5133)	This is constant Messages.LB_SETCOLUMNWIDTH.
LB_SETCOUNT (see page 5133)	This is constant Messages.LB_SETCOUNT.
LB_SETCURSEL (see page 5134)	This is constant Messages.LB_SETCURSEL.
LB_SETHORIZONTALEXTENT (see page 5134)	This is constant Messages.LB_SETHORIZONTALEXTENT.
LB_SETITEMDATA (see page 5134)	This is constant Messages.LB_SETITEMDATA.
LB_SETITEMHEIGHT (see page 5135)	This is constant Messages.LB_SETITEMHEIGHT.
LB_SETLOCALE (see page 5135)	This is constant Messages.LB_SETLOCALE.
LB_SETSEL (see page 5135)	This is constant Messages.LB_SETSEL.
LB_SETTABSTOPS (see page 5136)	This is constant Messages.LB_SETTABSTOPS.
LB_SETTOPINDEX (see page 5136)	This is constant Messages.LB_SETTOPINDEX.
LBN_DBLCLK (see page 5136)	This is constant Messages.LBN_DBLCLK.
LBN_ERRSPACE (see page 5137)	This is constant Messages.LBN_ERRSPACE.
LBN_KILLFOCUS (see page 5137)	This is constant Messages.LBN_KILLFOCUS.
LBN_SELCANCEL (see page 5137)	This is constant Messages.LBN_SELCANCEL.
LBN_SELCHANGE (see page 5138)	This is constant Messages.LBN_SELCHANGE.
LBN_SETFOCUS (see page 5138)	This is constant Messages.LBN_SETFOCUS.
PSM_PAGEINFO (see page 5138)	This is constant Messages.PSM_PAGEINFO.
PSM_SHEETINFO (see page 5139)	This is constant Messages.PSM_SHEETINFO.
SBM_ENABLE_ARROWS (see page 5139)	This is constant Messages.SBM_ENABLE_ARROWS.
SBM_GETPOS (see page 5139)	This is constant Messages.SBM_GETPOS.
SBM_GETRANGE (see page 5140)	This is constant Messages.SBM_GETRANGE.
SBM_GETSCROLLBARINFO (see page 5140)	This is constant Messages.SBM_GETSCROLLBARINFO.
SBM_GETSCROLLINFO (see page 5140)	This is constant Messages.SBM_GETSCROLLINFO.
SBM_SETPOS (see page 5141)	This is constant Messages.SBM_SETPOS.
SBM_SETRANGE (see page 5141)	This is constant Messages.SBM_SETRANGE.
SBM_SETRANGEREDRAW (see page 5141)	This is constant Messages.SBM_SETRANGEREDRAW.
SBM_SETSCROLLINFO (see page 5142)	This is constant Messages.SBM_SETSCROLLINFO.
WM_ACTIVATE (see page 5142)	This is constant Messages.WM_ACTIVATE.
WM_ACTIVATEAPP (see page 5142)	This is constant Messages.WM_ACTIVATEAPP.
WM_APP (see page 5143)	This is constant Messages.WM_APP.
WM_APPCOMMAND (see page 5143)	This is constant Messages.WM_APPCOMMAND.
WM_ASKCBFORMATNAME (see page 5143)	This is constant Messages.WM_ASKCBFORMATNAME.
WM_CANCELJOURNAL (see page 5144)	This is constant Messages.WM_CANCELJOURNAL.
WM_CANCELMODE (see page 5144)	This is constant Messages.WM_CANCELMODE.
WM_CAPTURECHANGED (see page 5144)	This is constant Messages.WM_CAPTURECHANGED.
WM_CHANGECBCHAIN (see page 5145)	This is constant Messages.WM_CHANGECBCHAIN.
WM_CHANGEUISTATE (see page 5145)	This is constant Messages.WM_CHANGEUISTATE.
WM_CHAR (see page 5145)	This is constant Messages.WM_CHAR.
WM_CHARTOITEM (see page 5146)	This is constant Messages.WM_CHARTOITEM.
WM_CHILDACTIVATE (see page 5146)	This is constant Messages.WM_CHILDACTIVATE.

WM_CLEAR (see page 5146)	This is constant Messages.WM_CLEAR.
WM_CLOSE (see page 5147)	This is constant Messages.WM_CLOSE.
WM_COALESCE_FIRST (see page 5147)	This is constant Messages.WM_COALESCE_FIRST.
WM_COALESCE_LAST (see page 5147)	This is constant Messages.WM_COALESCE_LAST.
WM_COMMAND (see page 5148)	This is constant Messages.WM_COMMAND.
WM_COMMNOTIFY (see page 5148)	This is constant Messages.WM_COMMNOTIFY.
WM_COMPACTING (see page 5148)	This is constant Messages.WM_COMPACTING.
WM_COMPAREITEM (see page 5149)	This is constant Messages.WM_COMPAREITEM.
WM_CONTEXTMENU (see page 5149)	This is constant Messages.WM_CONTEXTMENU.
WM_COPY (see page 5149)	This is constant Messages.WM_COPY.
WM_COPYDATA (see page 5150)	This is constant Messages.WM_COPYDATA.
WM_CREATE (see page 5150)	This is constant Messages.WM_CREATE.
WM_CTLCOLOR (see page 5150)	This is constant Messages.WM_CTLCOLOR.
WM_CTLCOLORBTN (see page 5151)	This is constant Messages.WM_CTLCOLORBTN.
WM_CTLCOLORDLG (see page 5151)	This is constant Messages.WM_CTLCOLORDLG.
WM_CTLCOLOREDIT (see page 5151)	This is constant Messages.WM_CTLCOLOREDIT.
WM_CTLCOLORLISTBOX (see page 5152)	This is constant Messages.WM_CTLCOLORLISTBOX.
WM_CTLCOLORMSGBOX (see page 5152)	This is constant Messages.WM_CTLCOLORMSGBOX.
WM_CTLCOLORSCROLLBAR (see page 5152)	This is constant Messages.WM_CTLCOLORSCROLLBAR.
WM_CTLCOLORSTATIC (see page 5153)	This is constant Messages.WM_CTLCOLORSTATIC.
WM_CUT (see page 5153)	This is constant Messages.WM_CUT.
WM_DDE_ACK (see page 5153)	This is constant Messages.WM_DDE_ACK.
WM_DDE_ADVISE (see page 5154)	This is constant Messages.WM_DDE_ADVISE.
WM_DDE_DATA (see page 5154)	This is constant Messages.WM_DDE_DATA.
WM_DDE_EXECUTE (see page 5154)	This is constant Messages.WM_DDE_EXECUTE.
WM_DDE_FIRST (see page 5155)	This is constant Messages.WM_DDE_FIRST.
WM_DDE_INITIATE (see page 5155)	This is constant Messages.WM_DDE_INITIATE.
WM_DDE_LAST (see page 5155)	This is constant Messages.WM_DDE_LAST.
WM_DDE_POKE (see page 5156)	This is constant Messages.WM_DDE_POKE.
WM_DDE_REQUEST (see page 5156)	This is constant Messages.WM_DDE_REQUEST.
WM_DDE_TERMINATE (see page 5156)	This is constant Messages.WM_DDE_TERMINATE.
WM_DDE_UNADVISE (see page 5157)	This is constant Messages.WM_DDE_UNADVISE.
WM_DEADCHAR (see page 5157)	This is constant Messages.WM_DEADCHAR.
WM_DELETEITEM (see page 5157)	This is constant Messages.WM_DELETEITEM.
WM_DESTROY (see page 5158)	This is constant Messages.WM_DESTROY.
WM_DESTROYCLIPBOARD (see page 5158)	This is constant Messages.WM_DESTROYCLIPBOARD.
WM_DEVICECHANGE (see page 5158)	This is constant Messages.WM_DEVICECHANGE.
WM_DEVMODECHANGE (see page 5159)	This is constant Messages.WM_DEVMODECHANGE.
WM_DISPLAYCHANGE (see page 5159)	This is constant Messages.WM_DISPLAYCHANGE.
WM_DRAWCLIPBOARD (see page 5159)	This is constant Messages.WM_DRAWCLIPBOARD.
WM_DRAWITEM (see page 5160)	This is constant Messages.WM_DRAWITEM.
WM_DROPFILES (see page 5160)	This is constant Messages.WM_DROPFILES.
WM_DWMCOLORIZATIONCOLORCHANGED (see page 5160)	This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.
WM_DWMCOMPOSITIONCHANGED (see page 5161)	This is constant Messages.WM_DWMCOMPOSITIONCHANGED.
WM_DWMNCRENDERINGCHANGED (see page 5161)	This is constant Messages.WM_DWMNCRENDERINGCHANGED.
WM_DWMWINDOWMAXIMIZEDCHANGE (see page 5161)	This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.
WM_ENABLE (see page 5162)	This is constant Messages.WM_ENABLE.
WM_ENDSESSION (see page 5162)	This is constant Messages.WM_ENDSESSION.
WM_ENTERIDLE (see page 5162)	This is constant Messages.WM_ENTERIDLE.
WM_ENTERMENULOOP (see page 5163)	This is constant Messages.WM_ENTERMENULOOP.
WM_ENTERSIZEMOVE (see page 5163)	This is constant Messages.WM_ENTERSIZEMOVE.
WM_ERASEBKGND (see page 5163)	This is constant Messages.WM_ERASEBKGND.
WM_EXITMENULOOP (see page 5164)	This is constant Messages.WM_EXITMENULOOP.
WM_EXITSIZEMOVE (see page 5164)	This is constant Messages.WM_EXITSIZEMOVE.

WM_FONTCHANGE (see page 5164)	This is constant Messages.WM_FONTCHANGE.
WM_GETDLGCODE (see page 5165)	This is constant Messages.WM_GETDLGCODE.
WM_GETFONT (see page 5165)	This is constant Messages.WM_GETFONT.
WM_GETHOTKEY (see page 5165)	This is constant Messages.WM_GETHOTKEY.
WM_GETICON (see page 5166)	This is constant Messages.WM_GETICON.
WM_GETMINMAXINFO (see page 5166)	This is constant Messages.WM_GETMINMAXINFO.
WM_GETOBJECT (see page 5166)	This is constant Messages.WM_GETOBJECT.
WM_GETTEXT (see page 5167)	This is constant Messages.WM_GETTEXT.
WM_GETTEXTLENGTH (see page 5167)	This is constant Messages.WM_GETTEXTLENGTH.
WM_HANDHELDFIRST (see page 5167)	This is constant Messages.WM_HANDHELDFIRST.
WM_HANDHELDLAST (see page 5168)	This is constant Messages.WM_HANDHELDLAST.
WM_HELP (see page 5168)	This is constant Messages.WM_HELP.
WM_HOTKEY (see page 5168)	This is constant Messages.WM_HOTKEY.
WM_HSCROLL (see page 5169)	This is constant Messages.WM_HSCROLL.
WM_HSCROLLCLIPBOARD (see page 5169)	This is constant Messages.WM_HSCROLLCLIPBOARD.
WM_ICONERASEBKGND (see page 5169)	This is constant Messages.WM_ICONERASEBKGND.
WM_IME_CHAR (see page 5170)	This is constant Messages.WM_IME_CHAR.
WM_IME_COMPOSITION (see page 5170)	This is constant Messages.WM_IME_COMPOSITION.
WM_IME_COMPOSITIONFULL (see page 5170)	This is constant Messages.WM_IME_COMPOSITIONFULL.
WM_IME_CONTROL (see page 5171)	This is constant Messages.WM_IME_CONTROL.
WM_IME_ENDCOMPOSITION (see page 5171)	This is constant Messages.WM_IME_ENDCOMPOSITION.
WM_IME_KEYDOWN (see page 5171)	This is constant Messages.WM_IME_KEYDOWN.
WM_IME_KEYLAST (see page 5172)	This is constant Messages.WM_IME_KEYLAST.
WM_IME_KEYUP (see page 5172)	This is constant Messages.WM_IME_KEYUP.
WM_IME_NOTIFY (see page 5172)	This is constant Messages.WM_IME_NOTIFY.
WM_IME_REQUEST (see page 5173)	This is constant Messages.WM_IME_REQUEST.
WM_IME_SELECT (see page 5173)	This is constant Messages.WM_IME_SELECT.
WM_IME_SETCONTEXT (see page 5173)	This is constant Messages.WM_IME_SETCONTEXT.
WM_IME_STARTCOMPOSITION (see page 5174)	This is constant Messages.WM_IME_STARTCOMPOSITION.
WM_INITDIALOG (see page 5174)	This is constant Messages.WM_INITDIALOG.
WM_INITMENU (see page 5174)	This is constant Messages.WM_INITMENU.
WM_INITMENUPOPUP (see page 5175)	This is constant Messages.WM_INITMENUPOPUP.
WM_INPUT (see page 5175)	This is constant Messages.WM_INPUT.
WM_INPUTLANGCHANGE (see page 5175)	This is constant Messages.WM_INPUTLANGCHANGE.
WM_INPUTLANGCHANGEREQUEST (see page 5176)	This is constant Messages.WM_INPUTLANGCHANGEREQUEST.
WM_KEYDOWN (see page 5176)	This is constant Messages.WM_KEYDOWN.
WM_KEYFIRST (see page 5176)	This is constant Messages.WM_KEYFIRST.
WM_KEYLAST (see page 5177)	This is constant Messages.WM_KEYLAST.
WM_KEYUP (see page 5177)	This is constant Messages.WM_KEYUP.
WM_KILLFOCUS (see page 5177)	This is constant Messages.WM_KILLFOCUS.
WM_LBUTTONDBLCLK (see page 5178)	This is constant Messages.WM_LBUTTONDBLCLK.
WM_LBUTTONDOWN (see page 5178)	This is constant Messages.WM_LBUTTONDOWN.
WM_LBUTTONUP (see page 5178)	This is constant Messages.WM_LBUTTONUP.
WM_MBUTTONDBLCLK (see page 5179)	This is constant Messages.WM_MBUTTONDBLCLK.
WM_MBUTTONDOWN (see page 5179)	This is constant Messages.WM_MBUTTONDOWN.
WM_MBUTTONUP (see page 5179)	This is constant Messages.WM_MBUTTONUP.
WM_MDIACTIVATE (see page 5180)	This is constant Messages.WM_MDIACTIVATE.
WM_MDICASCADE (see page 5180)	This is constant Messages.WM_MDICASCADE.
WM_MDICREATE (see page 5180)	This is constant Messages.WM_MDICREATE.
WM_MDIDESTROY (see page 5181)	This is constant Messages.WM_MDIDESTROY.
WM_MDIGETACTIVE (see page 5181)	This is constant Messages.WM_MDIGETACTIVE.
WM_MDIICONARRANGE (see page 5181)	This is constant Messages.WM_MDIICONARRANGE.
WM_MDIMAXIMIZE (see page 5182)	This is constant Messages.WM_MDIMAXIMIZE.
WM_MDINEXT (see page 5182)	This is constant Messages.WM_MDINEXT.

WM_MDIREFRESHMENU (see page 5182)	This is constant Messages.WM_MDIREFRESHMENU.
WM_MDIRESTORE (see page 5183)	This is constant Messages.WM_MDIRESTORE.
WM_MDISETMENU (see page 5183)	This is constant Messages.WM_MDISETMENU.
WM_MDITILE (see page 5183)	This is constant Messages.WM_MDITILE.
WM_MEASUREITEM (see page 5184)	This is constant Messages.WM_MEASUREITEM.
WM_MENUCHAR (see page 5184)	This is constant Messages.WM_MENUCHAR.
WM_MENUCOMMAND (see page 5184)	This is constant Messages.WM_MENUCOMMAND.
WM_MENUDRAG (see page 5185)	This is constant Messages.WM_MENUDRAG.
WM_MENUGETOBJECT (see page 5185)	This is constant Messages.WM_MENUGETOBJECT.
WM_MENURBUTTONUP (see page 5185)	This is constant Messages.WM_MENURBUTTONUP.
WM_MENUSELECT (see page 5186)	This is constant Messages.WM_MENUSELECT.
WM_MOUSEACTIVATE (see page 5186)	This is constant Messages.WM_MOUSEACTIVATE.
WM_MOUSEFIRST (see page 5186)	This is constant Messages.WM_MOUSEFIRST.
WM_MOUSEHOVER (see page 5187)	This is constant Messages.WM_MOUSEHOVER.
WM_MOUSELAST (see page 5187)	This is constant Messages.WM_MOUSELAST.
WM_MOUSELEAVE (see page 5187)	This is constant Messages.WM_MOUSELEAVE.
WM_MOUSEMOVE (see page 5188)	This is constant Messages.WM_MOUSEMOVE.
WM_MOUSEWHEEL (see page 5188)	This is constant Messages.WM_MOUSEWHEEL.
WM_MOVE (see page 5188)	This is constant Messages.WM_MOVE.
WM_MOVING (see page 5189)	This is constant Messages.WM_MOVING.
WM_NCACTIVATE (see page 5189)	This is constant Messages.WM_NCACTIVATE.
WM_NCCALCSIZE (see page 5189)	This is constant Messages.WM_NCCALCSIZE.
WM_NCCREATE (see page 5190)	This is constant Messages.WM_NCCREATE.
WM_NCDESTROY (see page 5190)	This is constant Messages.WM_NCDESTROY.
WM_NCHITTEST (see page 5190)	This is constant Messages.WM_NCHITTEST.
WM_NCLBUTTONDBLCLK (see page 5191)	This is constant Messages.WM_NCLBUTTONDBLCLK.
WM_NCLBUTTONDOWN (see page 5191)	This is constant Messages.WM_NCLBUTTONDOWN.
WM_NCLBUTTONUP (see page 5191)	This is constant Messages.WM_NCLBUTTONUP.
WM_NCMBUTTONDBLCLK (see page 5192)	This is constant Messages.WM_NCMBUTTONDBLCLK.
WM_NCMBUTTONDOWN (see page 5192)	This is constant Messages.WM_NCMBUTTONDOWN.
WM_NCMBUTTONUP (see page 5192)	This is constant Messages.WM_NCMBUTTONUP.
WM_NCMOUSEHOVER (see page 5193)	This is constant Messages.WM_NCMOUSEHOVER.
WM_NCMOUSELEAVE (see page 5193)	This is constant Messages.WM_NCMOUSELEAVE.
WM_NCMOUSEMOVE (see page 5193)	This is constant Messages.WM_NCMOUSEMOVE.
WM_NCPAINT (see page 5194)	This is constant Messages.WM_NCPAINT.
WM_NCRBUTTONDBLCLK (see page 5194)	This is constant Messages.WM_NCRBUTTONDBLCLK.
WM_NCRBUTTONDOWN (see page 5194)	This is constant Messages.WM_NCRBUTTONDOWN.
WM_NCRBUTTONUP (see page 5195)	This is constant Messages.WM_NCRBUTTONUP.
WM_NCXBUTTONDBLCLK (see page 5195)	This is constant Messages.WM_NCXBUTTONDBLCLK.
WM_NCXBUTTONDOWN (see page 5195)	This is constant Messages.WM_NCXBUTTONDOWN.
WM_NCXBUTTONUP (see page 5196)	This is constant Messages.WM_NCXBUTTONUP.
WM_NEXTDLGCTL (see page 5196)	This is constant Messages.WM_NEXTDLGCTL.
WM_NEXTMENU (see page 5196)	This is constant Messages.WM_NEXTMENU.
WM_NOTIFY (see page 5197)	This is constant Messages.WM_NOTIFY.
WM_NOTIFYFORMAT (see page 5197)	This is constant Messages.WM_NOTIFYFORMAT.
WM_NULL (see page 5197)	This is constant Messages.WM_NULL.
WM_PAINT (see page 5198)	This is constant Messages.WM_PAINT.
WM_PAINTCLIPBOARD (see page 5198)	This is constant Messages.WM_PAINTCLIPBOARD.
WM_PAINTICON (see page 5198)	This is constant Messages.WM_PAINTICON.
WM_PALETTECHANGED (see page 5199)	This is constant Messages.WM_PALETTECHANGED.
WM_PALETTEISCHANGING (see page 5199)	This is constant Messages.WM_PALETTEISCHANGING.
WM_PARENTNOTIFY (see page 5199)	This is constant Messages.WM_PARENTNOTIFY.
WM_PASTE (see page 5200)	This is constant Messages.WM_PASTE.
WM_PENWINFIRST (see page 5200)	This is constant Messages.WM_PENWINFIRST.

WM_PENWINLAST (see page 5200)	This is constant Messages.WM_PENWINLAST.
WM_POWER (see page 5201)	This is constant Messages.WM_POWER.
WM_POWERBROADCAST (see page 5201)	This is constant Messages.WM_POWERBROADCAST.
WM_PRINT (see page 5201)	This is constant Messages.WM_PRINT.
WM_PRINTCLIENT (see page 5202)	This is constant Messages.WM_PRINTCLIENT.
WM_QUERYDRAGICON (see page 5202)	This is constant Messages.WM_QUERYDRAGICON.
WM_QUERYENDSESSION (see page 5202)	This is constant Messages.WM_QUERYENDSESSION.
WM_QUERYNEWPALETTE (see page 5203)	This is constant Messages.WM_QUERYNEWPALETTE.
WM_QUERYOPEN (see page 5203)	This is constant Messages.WM_QUERYOPEN.
WM_QUERYUISTATE (see page 5203)	This is constant Messages.WM_QUERYUISTATE.
WM_QUEUESYNC (see page 5204)	This is constant Messages.WM_QUEUESYNC.
WM_QUIT (see page 5204)	This is constant Messages.WM_QUIT.
WM_RBUTTONDBLCLK (see page 5204)	This is constant Messages.WM_RBUTTONDBLCLK.
WM_RBUTTONDOWN (see page 5205)	This is constant Messages.WM_RBUTTONDOWN.
WM_RBUTTONUP (see page 5205)	This is constant Messages.WM_RBUTTONUP.
WM_RENDERALLFORMATS (see page 5205)	This is constant Messages.WM_RENDERALLFORMATS.
WM_RENDERFORMAT (see page 5206)	This is constant Messages.WM_RENDERFORMAT.
WM_SETCURSOR (see page 5206)	This is constant Messages.WM_SETCURSOR.
WM_SETFOCUS (see page 5206)	This is constant Messages.WM_SETFOCUS.
WM_SETFONT (see page 5207)	This is constant Messages.WM_SETFONT.
WM_SETHOTKEY (see page 5207)	This is constant Messages.WM_SETHOTKEY.
WM_SETICON (see page 5207)	This is constant Messages.WM_SETICON.
WM_SETREDRAW (see page 5208)	This is constant Messages.WM_SETREDRAW.
WM_SETTEXT (see page 5208)	This is constant Messages.WM_SETTEXT.
WM_SETTINGCHANGE (see page 5208)	This is constant Messages.WM_SETTINGCHANGE.
WM_SHOWWINDOW (see page 5209)	This is constant Messages.WM_SHOWWINDOW.
WM_SIZE (see page 5209)	This is constant Messages.WM_SIZE.
WM_SIZECLIPBOARD (see page 5209)	This is constant Messages.WM_SIZECLIPBOARD.
WM_SIZING (see page 5210)	This is constant Messages.WM_SIZING.
WM_SPOOLERSTATUS (see page 5210)	This is constant Messages.WM_SPOOLERSTATUS.
WM_STYLECHANGED (see page 5210)	This is constant Messages.WM_STYLECHANGED.
WM_STYLECHANGING (see page 5211)	This is constant Messages.WM_STYLECHANGING.
WM_SYSCHAR (see page 5211)	This is constant Messages.WM_SYSCHAR.
WM_SYSCOLORCHANGE (see page 5211)	This is constant Messages.WM_SYSCOLORCHANGE.
WM_SYSCOMMAND (see page 5212)	This is constant Messages.WM_SYSCOMMAND.
WM_SYSDEADCHAR (see page 5212)	This is constant Messages.WM_SYSDEADCHAR.
WM_SYSKEYDOWN (see page 5212)	This is constant Messages.WM_SYSKEYDOWN.
WM_SYSKEYUP (see page 5213)	This is constant Messages.WM_SYSKEYUP.
WM_SYSTEMERROR (see page 5213)	This is constant Messages.WM_SYSTEMERROR.
WM_TABLET_FIRST (see page 5213)	This is constant Messages.WM_TABLET_FIRST.
WM_TABLET_LAST (see page 5214)	This is constant Messages.WM_TABLET_LAST.
WM_TCARD (see page 5214)	This is constant Messages.WM_TCARD.
WM_THEMECHANGED (see page 5214)	This is constant Messages.WM_THEMECHANGED.
WM_TIMECHANGE (see page 5215)	This is constant Messages.WM_TIMECHANGE.
WM_TIMER (see page 5215)	This is constant Messages.WM_TIMER.
WM_UNDO (see page 5215)	This is constant Messages.WM_UNDO.
WM_UNICHAR (see page 5216)	This is constant Messages.WM_UNICHAR.
WM_UNINITMENUPOPUP (see page 5216)	This is constant Messages.WM_UNINITMENUPOPUP.
WM_UPDATEUISTATE (see page 5216)	This is constant Messages.WM_UPDATEUISTATE.
WM_USER (see page 5217)	This is constant Messages.WM_USER.
WM_USERCHANGED (see page 5217)	This is constant Messages.WM_USERCHANGED.
WM_VKEYTOITEM (see page 5217)	This is constant Messages.WM_VKEYTOITEM.
WM_VSCROLL (see page 5218)	This is constant Messages.WM_VSCROLL.
WM_VSCROLLCLIPBOARD (see page 5218)	This is constant Messages.WM_VSCROLLCLIPBOARD.

WM_WINDOWPOSCHANGED (see page 5218)	This is constant Messages.WM_WINDOWPOSCHANGED.
WM_WINDOWPOSCHANGING (see page 5219)	This is constant Messages.WM_WINDOWPOSCHANGING.
WM_WININICHANGE (see page 5219)	This is constant Messages.WM_WININICHANGE.
WM_WTSSESSION_CHANGE (see page 5219)	This is constant Messages.WM_WTSSESSION_CHANGE.

Functions

	Name	Description
	SendGetIntMessage (see page 4991)	Send Windows message, sending and receiving integer values.
	SendGetStructMessage (see page 4992)	Send Windows message, sending and receiving data structure.
	SendStructMessage (see page 4992)	Send Windows message, sending data structure.
	SendTextMessage (see page 4993)	Send Windows message, sending text.

Structs, Records, Enums

	Name	Description
	TMessage (see page 4997)	TMessage represents a Windows message.
	TMSHMouseWheel (see page 4998)	This is record Messages.TMSHMouseWheel.
	TWMActivate (see page 4998)	This is record Messages.TWMActivate.
	TWMActivateApp (see page 4999)	This is record Messages.TWMActivateApp.
	TWMAskCBFormatName (see page 4999)	This is record Messages.TWMAskCBFormatName.
	TWMChangeCBChain (see page 5000)	This is record Messages.TWMChangeCBChain.
	TWMCharToItem (see page 5000)	This is record Messages.TWMCharToItem.
	TWMChooseFont_GetLogFont (see page 5001)	This is record Messages.TWMChooseFont_GetLogFont.
	TWMCommand (see page 5001)	This is record Messages.TWMCommand.
	TWMCompacting (see page 5002)	This is record Messages.TWMCompacting.
	TWMCompareItem (see page 5002)	This is record Messages.TWMCompareItem.
	TWMContextMenu (see page 5003)	This is record Messages.TWMContextMenu.
	TWMCopyData (see page 5003)	This is record Messages.TWMCopyData.
	TWMCreate (see page 5004)	This is record Messages.TWMCreate.
	TWMCtlColor (see page 5004)	This is record Messages.TWMCtlColor.
	TWMDDE_Ack (see page 5005)	This is record Messages.TWMDDE_Ack.
	TWMDDE_Advise (see page 5005)	This is record Messages.TWMDDE_Advise.
	TWMDDE_Data (see page 5006)	This is record Messages.TWMDDE_Data.
	TWMDDE_Execute (see page 5006)	This is record Messages.TWMDDE_Execute.
	TWMDDE_Initiate (see page 5007)	This is record Messages.TWMDDE_Initiate.
	TWMDDE_Poke (see page 5007)	This is record Messages.TWMDDE_Poke.
	TWMDDE_Request (see page 5008)	This is record Messages.TWMDDE_Request.
	TWMDDE_Terminate (see page 5008)	This is record Messages.TWMDDE_Terminate.
	TWMDDE_Unadvise (see page 5009)	This is record Messages.TWMDDE_Unadvise.
	TWMDeleteItem (see page 5009)	This is record Messages.TWMDeleteItem.
	TWMDevModeChange (see page 5010)	This is record Messages.TWMDevModeChange.
	TWMDisplayChange (see page 5010)	This is record Messages.TWMDisplayChange.
	TWMDrawItem (see page 5011)	This is record Messages.TWMDrawItem.
	TWMDropFiles (see page 5011)	This is record Messages.TWMDropFiles.
	TWMEnable (see page 5012)	This is record Messages.TWMEnable.
	TWMEndSession (see page 5012)	This is record Messages.TWMEndSession.
	TWMEnterIdle (see page 5013)	This is record Messages.TWMEnterIdle.
	TWMEnterMenuLoop (see page 5013)	This is record Messages.TWMEnterMenuLoop.
	TWMEraseBkgnd (see page 5014)	This is record Messages.TWMEraseBkgnd.
	TWMGetIcon (see page 5014)	This is record Messages.TWMGetIcon.
	TWMGetMinMaxInfo (see page 5015)	This is record Messages.TWMGetMinMaxInfo.
	TWMGetText (see page 5015)	This is record Messages.TWMGetText.
	TWMHelp (see page 5016)	This is record Messages.TWMHelp.
	TWMHotKey (see page 5016)	This is record Messages.TWMHotKey.
	TWMHScrollClipboard (see page 5017)	This is record Messages.TWMHScrollClipboard.
	TWMInitDialog (see page 5017)	This is record Messages.TWMInitDialog.

	TWMInitMenu (see page 5018)	This is record Messages.TWMInitMenu.
	TWMInitMenuPopup (see page 5018)	This is record Messages.TWMInitMenuPopup.
	TWMKey (see page 5019)	This is record Messages.TWMKey.
	TWMKillFocus (see page 5019)	This is record Messages.TWMKillFocus.
	TWMMDIActivate (see page 5020)	This is record Messages.TWMMDIActivate.
	TWMMDICascade (see page 5020)	This is record Messages.TWMMDICascade.
	TWMMDICreate (see page 5021)	This is record Messages.TWMMDICreate.
	TWMMDIDestroy (see page 5021)	This is record Messages.TWMMDIDestroy.
	TWMMDIMaximize (see page 5022)	This is record Messages.TWMMDIMaximize.
	TWMMDINext (see page 5022)	This is record Messages.TWMMDINext.
	TWMMDIRestore (see page 5023)	This is record Messages.TWMMDIRestore.
	TWMMDISetMenu (see page 5023)	This is record Messages.TWMMDISetMenu.
	TWMMDITile (see page 5024)	This is record Messages.TWMMDITile.
	TWMMeasureItem (see page 5024)	This is record Messages.TWMMeasureItem.
	TWMMenuChar (see page 5025)	This is record Messages.TWMMenuChar.
	TWMMenuSelect (see page 5025)	This is record Messages.TWMMenuSelect.
	TWMMouse (see page 5026)	This is record Messages.TWMMouse.
	TWMMouseActivate (see page 5026)	This is record Messages.TWMMouseActivate.
	TWMMouseWheel (see page 5027)	This is record Messages.TWMMouseWheel.
	TWMMove (see page 5027)	This is record Messages.TWMMove.
	TWMMoving (see page 5028)	This is record Messages.TWMMoving.
	TWMNCActivate (see page 5028)	This is record Messages.TWMNCActivate.
	TWMNCCalcSize (see page 5029)	This is record Messages.TWMNCCalcSize.
	TWMNCCreate (see page 5029)	This is record Messages.TWMNCCreate.
	TWMNCHitMessage (see page 5030)	This is record Messages.TWMNCHitMessage.
	TWMNCHitTest (see page 5030)	This is record Messages.TWMNCHitTest.
	TWMNCPaint (see page 5031)	This is record Messages.TWMNCPaint.
	TWMNextDlgCtl (see page 5031)	This is record Messages.TWMNextDlgCtl.
	TWMNoParams (see page 5032)	This is record Messages.TWMNoParams.
	TWMNotify (see page 5032)	This is record Messages.TWMNotify.
	TWMNotifyFormat (see page 5033)	This is record Messages.TWMNotifyFormat.
	TWMPaint (see page 5033)	This is record Messages.TWMPaint.
	TWMPaintClipboard (see page 5034)	This is record Messages.TWMPaintClipboard.
	TWMPaletteChanged (see page 5034)	This is record Messages.TWMPaletteChanged.
	TWMPaletteIsChanging (see page 5035)	This is record Messages.TWMPaletteIsChanging.
	TWMParentNotify (see page 5035)	This is record Messages.TWMParentNotify.
	TWMPower (see page 5036)	This is record Messages.TWMPower.
	TWMPrint (see page 5036)	This is record Messages.TWMPrint.
	TWMQueryEndSession (see page 5037)	This is record Messages.TWMQueryEndSession.
	TWMQuit (see page 5037)	This is record Messages.TWMQuit.
	TWMRenderFormat (see page 5038)	This is record Messages.TWMRenderFormat.
	TWMScroll (see page 5038)	This is record Messages.TWMScroll.
	TWMSetCursor (see page 5039)	This is record Messages.TWMSetCursor.
	TWMSetFocus (see page 5039)	This is record Messages.TWMSetFocus.
	TWMSetFont (see page 5040)	This is record Messages.TWMSetFont.
	TWMSetHotKey (see page 5040)	This is record Messages.TWMSetHotKey.
	TWMSetIcon (see page 5041)	This is record Messages.TWMSetIcon.
	TWMSetRedraw (see page 5041)	This is record Messages.TWMSetRedraw.
	TWMSetText (see page 5042)	This is record Messages.TWMSetText.
	TWMSettingChange (see page 5042)	This is record Messages.TWMSettingChange.
	TWMShowWindow (see page 5043)	This is record Messages.TWMShowWindow.
	TWMSize (see page 5043)	This is record Messages.TWMSize.
	TWMSizeClipboard (see page 5044)	This is record Messages.TWMSizeClipboard.
	TWMSpoolerStatus (see page 5044)	This is record Messages.TWMSpoolerStatus.

	TWMStyleChange (see page 5045)	This is record Messages.TWMStyleChange.
	TWMSysCommand (see page 5045)	This is record Messages.TWMSysCommand.
	TWMSysDeadChar (see page 5046)	This is record Messages.TWMSysDeadChar.
	TWMSystemError (see page 5046)	This is record Messages.TWMSystemError.
	TWMTimer (see page 5047)	This is record Messages.TWMTimer.
	TWMUIState (see page 5047)	This is record Messages.TWMUIState.
	TWMVScrollClipboard (see page 5048)	This is record Messages.TWMVScrollClipboard.
	TWMWindowPosMsg (see page 5048)	This is record Messages.TWMWindowPosMsg.
	TWMWinIniChange (see page 5049)	This is record Messages.TWMWinIniChange.

Types

Name	Description
PMessage (see page 5051)	This is type Messages.PMessage.
TWMCancelMode (see page 5052)	This is type Messages.TWMCancelMode.
TWMChangeUIState (see page 5052)	This is type Messages.TWMChangeUIState.
TWMChar (see page 5052)	This is type Messages.TWMChar.
TWMChildActivate (see page 5053)	This is type Messages.TWMChildActivate.
TWMClear (see page 5053)	This is type Messages.TWMClear.
TWMClose (see page 5053)	This is type Messages.TWMClose.
TWMCopy (see page 5054)	This is type Messages.TWMCopy.
TWMCtlColorBtn (see page 5054)	This is type Messages.TWMCtlColorBtn.
TWMCtlColorDlg (see page 5054)	This is type Messages.TWMCtlColorDlg.
TWMCtlColorEdit (see page 5055)	This is type Messages.TWMCtlColorEdit.
TWMCtlColorListbox (see page 5055)	This is type Messages.TWMCtlColorListbox.
TWMCtlColorMsgbox (see page 5055)	This is type Messages.TWMCtlColorMsgbox.
TWMCtlColorScrollbar (see page 5056)	This is type Messages.TWMCtlColorScrollbar.
TWMCtlColorStatic (see page 5056)	This is type Messages.TWMCtlColorStatic.
TWMCut (see page 5056)	This is type Messages.TWMCut.
TWMDeadChar (see page 5057)	This is type Messages.TWMDeadChar.
TWMDestroy (see page 5057)	This is type Messages.TWMDestroy.
TWMDestroyClipboard (see page 5057)	This is type Messages.TWMDestroyClipboard.
TWMDrawClipboard (see page 5058)	This is type Messages.TWMDrawClipboard.
TWMExitMenuLoop (see page 5058)	This is type Messages.TWMExitMenuLoop.
TWMFontChange (see page 5058)	This is type Messages.TWMFontChange.
TWMGetDlgCode (see page 5059)	This is type Messages.TWMGetDlgCode.
TWMGetFont (see page 5059)	This is type Messages.TWMGetFont.
TWMGetHotKey (see page 5059)	This is type Messages.TWMGetHotKey.
TWMGetTextLength (see page 5060)	This is type Messages.TWMGetTextLength.
TWMHScroll (see page 5060)	This is type Messages.TWMHScroll.
TWMIconEraseBkgnd (see page 5060)	This is type Messages.TWMIconEraseBkgnd.
TWMKeyDown (see page 5061)	This is type Messages.TWMKeyDown.
TWMKeyUp (see page 5061)	This is type Messages.TWMKeyUp.
TWMLButtonDblClk (see page 5061)	This is type Messages.TWMLButtonDblClk.
TWMLButtonDown (see page 5062)	TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.
TWMLButtonUp (see page 5062)	This is type Messages.TWMLButtonUp.
TWMMButtonDblClk (see page 5062)	This is type Messages.TWMMButtonDblClk.
TWMMButtonDown (see page 5063)	This is type Messages.TWMMButtonDown.
TWMMButtonUp (see page 5063)	This is type Messages.TWMMButtonUp.
TWMMDIGetActive (see page 5063)	This is type Messages.TWMMDIGetActive.
TWMMDIIconArrange (see page 5064)	This is type Messages.TWMMDIIconArrange.
TWMMDIRefreshMenu (see page 5064)	This is type Messages.TWMMDIRefreshMenu.
TWMMouseMove (see page 5064)	This is type Messages.TWMMouseMove.
TWMNCDestroy (see page 5065)	This is type Messages.TWMNCDestroy.
TWMNCLButtonDblClk (see page 5065)	This is type Messages.TWMNCLButtonDblClk.

TWMNCLButtonDown (see page 5065)	This is type Messages.TWMNCLButtonDown.
TWMNCLButtonUp (see page 5066)	This is type Messages.TWMNCLButtonUp.
TWMNCMButtonDblClk (see page 5066)	This is type Messages.TWMNCMButtonDblClk.
TWMNCMButtonDown (see page 5066)	This is type Messages.TWMNCMButtonDown.
TWMNCMButtonUp (see page 5067)	This is type Messages.TWMNCMButtonUp.
TWMNCMouseMove (see page 5067)	This is type Messages.TWMNCMouseMove.
TWMNCRButtonDblClk (see page 5067)	This is type Messages.TWMNCRButtonDblClk.
TWMNCRButtonDown (see page 5068)	This is type Messages.TWMNCRButtonDown.
TWMNCRButtonUp (see page 5068)	This is type Messages.TWMNCRButtonUp.
TWMPaintIcon (see page 5068)	This is type Messages.TWMPaintIcon.
TWMPaste (see page 5069)	This is type Messages.TWMPaste.
TWMPrintClient (see page 5069)	This is type Messages.TWMPrintClient.
TWMQueryDragIcon (see page 5069)	This is type Messages.TWMQueryDragIcon.
TWMQueryNewPalette (see page 5070)	This is type Messages.TWMQueryNewPalette.
TWMQueryOpen (see page 5070)	This is type Messages.TWMQueryOpen.
TWMQueryUIState (see page 5070)	This is type Messages.TWMQueryUIState.
TWMQueueSync (see page 5071)	This is type Messages.TWMQueueSync.
TWMRButtonDblClk (see page 5071)	This is type Messages.TWMRButtonDblClk.
TWMRButtonDown (see page 5071)	This is type Messages.TWMRButtonDown.
TWMRButtonUp (see page 5072)	This is type Messages.TWMRButtonUp.
TWMRenderAllFormats (see page 5072)	This is type Messages.TWMRenderAllFormats.
TWMStyleChanged (see page 5072)	This is type Messages.TWMStyleChanged.
TWMStyleChanging (see page 5073)	This is type Messages.TWMStyleChanging.
TWMSysChar (see page 5073)	This is type Messages.TWMSysChar.
TWMSysColorChange (see page 5073)	This is type Messages.TWMSysColorChange.
TWMSysKeyDown (see page 5074)	This is type Messages.TWMSysKeyDown.
TWMSysKeyUp (see page 5074)	This is type Messages.TWMSysKeyUp.
TWMTimeChange (see page 5074)	This is type Messages.TWMTimeChange.
TWMUndo (see page 5075)	This is type Messages.TWMUndo.
TWMUpdateUIState (see page 5075)	This is type Messages.TWMUpdateUIState.
TWMVKeyToItem (see page 5075)	This is type Messages.TWMVKeyToItem.
TWMVScroll (see page 5076)	This is type Messages.TWMVScroll.
TWMWindowPosChanged (see page 5076)	This is type Messages.TWMWindowPosChanged.
TWMWindowPosChanging (see page 5076)	This is type Messages.TWMWindowPosChanging.

49.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SendGetIntMessage (see page 4991)	Send Windows message, sending and receiving integer values.
	SendGetStructMessage (see page 4992)	Send Windows message, sending and receiving data structure.
	SendStructMessage (see page 4992)	Send Windows message, sending data structure.
	SendTextMessage (see page 4993)	Send Windows message, sending text.

49.1.1 Messages.SendGetIntMessage Function

Send Windows message, sending and receiving integer values.

Pascal

```
function SendGetIntMessage(Handle: HWND; Msg: UINT; var WParam: Integer; var LParam: Integer):
LRESULT;
```

C++

```
LRESULT SendGetIntMessage(HWND Handle, UINT Msg, int WParam, int LParam);
```

File

Messages

Description

SendGetIntMessage calls the Windows **SendMessage** function and returns an **LRESULT**.

Handle is an **HWND** window handle.

Msg is the Windows message ID. This constant is exactly the same as the Windows message ID described in the Microsoft Windows SDK.

WParam is a **WPARAM** for the message, depending on **Msg**. A value may be sent and returned.

LParam is an integer **LPARAM** for the message, depending on **Msg**. A value may be sent and returned.

Various records have been defined in the **Messages (see page 4979)** unit to represent Windows messages. For instance, **TWMCopy (see page 5054)** represents the message **WM_COPY (see page 5149)**, **TWMKeyDown (see page 5061)** represents **WM_KEYDOWN (see page 5176)**, and so on.

See the Microsoft Windows SDK documentation for information on the Windows messages and their associated **WPARAM** and **LPARAM**.

See Also

SendMessage (MSDN)

LPARAM (MSDN)

WPARAM (MSDN)

Handling messages

Messages unit

49.1.2 Messages.SendGetStructMessage Function

Send Windows message, sending and receiving data structure.

Pascal

```
function SendGetStructMessage(Handle: HWND; Msg: UINT; WParam: WPARAM; var LParam; Unused:
Boolean = False): LRESULT;
```

C++

```
LRESULT SendGetStructMessage(HWND Handle, UINT Msg, WPARAM WParam,  LParam, Boolean Unused =
False);
```

File

Messages

Description

SendGetStructMessage calls the Windows **SendMessage** function and returns an **LRESULT**.

Handle is an **HWND** window handle.

Msg is the Windows message ID. This constant is exactly the same as the Windows message ID described in the Microsoft Windows SDK.

WParam is a **WPARAM** for the message, depending on **Msg**. The value is sent.

LParam is a data structure **LPARAM** for the message, depending on **Msg**. A value may be sent and returned.

Unused is not used and is optional.

Various records have been defined in the **Messages (see page 4979)** unit to represent Windows messages. For instance, **TWMCopy (see page 5054)** represents the message **WM_COPY (see page 5149)**, **TWMKeyDown (see page 5061)** represents **WM_KEYDOWN (see page 5176)**, and so on.

See the Microsoft Windows SDK documentation for information on the Windows messages and their associated **WPARAM** and **LPARAM**.

See Also

SendMessage (MSDN)

LPARAM (MSDN)

WPARAM (MSDN)

Handling messages

Messages unit

49.1.3 Messages.SendStructMessage Function

Send Windows message, sending data structure.

Pascal

```
function SendStructMessage(Handle: HWND; Msg: UINT; WParam: WPARAM; const LParam): LRESULT;
```

C++

```
LRESULT SendStructMessage(HWND Handle, UINT Msg, WPARAM WParam, const  LParam);
```

File

Messages

Description

SendStructMessage calls the Windows **SendMessage** function and returns an **LRESULT**.

Handle is an **HWND** window handle.

Msg is the Windows message ID. This constant is exactly the same as the Windows message ID described in the Microsoft Windows SDK.

WParam is a **WPARAM** for the message, depending on **Msg**. The value is sent.

LParam is a data structure **LPARAM** for the message, depending on **Msg**. The value is sent.

Various records have been defined in the **Messages (see page 4979)** unit to represent Windows messages. For instance, **TWMCopy (see page 5054)** represents the message **WM_COPY (see page 5149)**, **TWMKeyDown (see page 5061)** represents **WM_KEYDOWN (see page 5176)**, and so on.

See the Microsoft Windows SDK documentation for information on the Windows messages and their associated **WPARAM** and **LPARAM**.

See Also

SendMessage (MSDN)

LPARAM (MSDN)

WPARAM (MSDN)

Handling messages

Messages unit

49.1.4 Messages.SendTextMessage Function

Send Windows message, sending text.

Pascal

```
function SendTextMessage(Handle: HWND; Msg: UINT; WParam: WPARAM; LParam: UnicodeString):
LRESULT; overload;
```

C++

```
LRESULT SendTextMessage(HWND Handle, UINT Msg, WPARAM WParam, UnicodeString LParam);
```

File

Messages

Description

SendTextMessage calls the Windows **SendMessage** function and returns an **LRESULT**.

Handle is an **HWND** window handle.

Msg is the Windows message ID. This constant is exactly the same as the Windows message ID described in the Microsoft Windows SDK.

WParam is a **WPARAM** for the message, depending on **Msg**. The value is sent.

LParam is a **string LPARAM** for the message, depending on **Msg**. The value is sent.

Various records have been defined in the **Messages (see page 4979)** unit to represent Windows messages. For instance, **TWMCopy (see page 5054)** represents the message **WM_COPY (see page 5149)**, **TWMKeyDown (see page 5061)** represents **WM_KEYDOWN (see page 5176)**, and so on.

See the Microsoft Windows SDK documentation for information on the Windows messages and their associated **WPARAM** and **LPARAM**.

See Also

SendMessage (MSDN)

LPARAM (MSDN)

WPARAM (MSDN)

Handling messages

Messages unit

49.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TMessage (see page 4997)	TMessage represents a Windows message.
	TMSHMouseWheel (see page 4998)	This is record Messages.TMSHMouseWheel.
	TWMActivate (see page 4998)	This is record Messages.TWMActivate.
	TWMActivateApp (see page 4999)	This is record Messages.TWMActivateApp.
	TWMAskCBFormatName (see page 4999)	This is record Messages.TWMAskCBFormatName.
	TWMChangeCBChain (see page 5000)	This is record Messages.TWMChangeCBChain.
	TWMCharToItem (see page 5000)	This is record Messages.TWMCharToItem.
	TWMChooseFont_GetLogFont (see page 5001)	This is record Messages.TWMChooseFont_GetLogFont.
	TWMCommand (see page 5001)	This is record Messages.TWMCommand.
	TWMCompacting (see page 5002)	This is record Messages.TWMCompacting.
	TWMCompareItem (see page 5002)	This is record Messages.TWMCompareItem.
	TWMContextMenu (see page 5003)	This is record Messages.TWMContextMenu.
	TWMCopyData (see page 5003)	This is record Messages.TWMCopyData.
	TWMCreate (see page 5004)	This is record Messages.TWMCreate.
	TWMCtlColor (see page 5004)	This is record Messages.TWMCtlColor.
	TWMDDE_Ack (see page 5005)	This is record Messages.TWMDDE_Ack.
	TWMDDE_Advise (see page 5005)	This is record Messages.TWMDDE_Advise.
	TWMDDE_Data (see page 5006)	This is record Messages.TWMDDE_Data.
	TWMDDE_Execute (see page 5006)	This is record Messages.TWMDDE_Execute.
	TWMDDE_Initiate (see page 5007)	This is record Messages.TWMDDE_Initiate.
	TWMDDE_Poke (see page 5007)	This is record Messages.TWMDDE_Poke.
	TWMDDE_Request (see page 5008)	This is record Messages.TWMDDE_Request.
	TWMDDE_Terminate (see page 5008)	This is record Messages.TWMDDE_Terminate.
	TWMDDE_Unadvise (see page 5009)	This is record Messages.TWMDDE_Unadvise.
	TWMDeleteItem (see page 5009)	This is record Messages.TWMDeleteItem.
	TWMDevModeChange (see page 5010)	This is record Messages.TWMDevModeChange.
	TWMDisplayChange (see page 5010)	This is record Messages.TWMDisplayChange.
	TWMDrawItem (see page 5011)	This is record Messages.TWMDrawItem.
	TWMDropFiles (see page 5011)	This is record Messages.TWMDropFiles.
	TWMEnable (see page 5012)	This is record Messages.TWMEnable.
	TWMEndSession (see page 5012)	This is record Messages.TWMEndSession.
	TWMEnterIdle (see page 5013)	This is record Messages.TWMEnterIdle.
	TWMEnterMenuLoop (see page 5013)	This is record Messages.TWMEnterMenuLoop.
	TWMEraseBkgnd (see page 5014)	This is record Messages.TWMEraseBkgnd.
	TWMGetIcon (see page 5014)	This is record Messages.TWMGetIcon.
	TWMGetMinMaxInfo (see page 5015)	This is record Messages.TWMGetMinMaxInfo.
	TWMGetText (see page 5015)	This is record Messages.TWMGetText.
	TWMHelp (see page 5016)	This is record Messages.TWMHelp.
	TWMHotKey (see page 5016)	This is record Messages.TWMHotKey.
	TWMHScrollClipboard (see page 5017)	This is record Messages.TWMHScrollClipboard.
	TWMInitDialog (see page 5017)	This is record Messages.TWMInitDialog.
	TWMInitMenu (see page 5018)	This is record Messages.TWMInitMenu.
	TWMInitMenuPopup (see page 5018)	This is record Messages.TWMInitMenuPopup.
	TWMKey (see page 5019)	This is record Messages.TWMKey.

	TWMKillFocus (see page 5019)	This is record Messages.TWMKillFocus.
	TWMMDIActivate (see page 5020)	This is record Messages.TWMMDIActivate.
	TWMMDICascade (see page 5020)	This is record Messages.TWMMDICascade.
	TWMMDICreate (see page 5021)	This is record Messages.TWMMDICreate.
	TWMMDIDestroy (see page 5021)	This is record Messages.TWMMDIDestroy.
	TWMMDIMaximize (see page 5022)	This is record Messages.TWMMDIMaximize.
	TWMMDINext (see page 5022)	This is record Messages.TWMMDINext.
	TWMMDIRestore (see page 5023)	This is record Messages.TWMMDIRestore.
	TWMMDISetMenu (see page 5023)	This is record Messages.TWMMDISetMenu.
	TWMMDITile (see page 5024)	This is record Messages.TWMMDITile.
	TWMMeasureItem (see page 5024)	This is record Messages.TWMMeasureItem.
	TWMMenuChar (see page 5025)	This is record Messages.TWMMenuChar.
	TWMMenuSelect (see page 5025)	This is record Messages.TWMMenuSelect.
	TWMMouse (see page 5026)	This is record Messages.TWMMouse.
	TWMMouseActivate (see page 5026)	This is record Messages.TWMMouseActivate.
	TWMMouseWheel (see page 5027)	This is record Messages.TWMMouseWheel.
	TWMMove (see page 5027)	This is record Messages.TWMMove.
	TWMMoving (see page 5028)	This is record Messages.TWMMoving.
	TWMNCActivate (see page 5028)	This is record Messages.TWMNCActivate.
	TWMNCCalcSize (see page 5029)	This is record Messages.TWMNCCalcSize.
	TWMNCCreate (see page 5029)	This is record Messages.TWMNCCreate.
	TWMNCHitMessage (see page 5030)	This is record Messages.TWMNCHitMessage.
	TWMNCHitTest (see page 5030)	This is record Messages.TWMNCHitTest.
	TWMNCPaint (see page 5031)	This is record Messages.TWMNCPaint.
	TWMNextDlgCtl (see page 5031)	This is record Messages.TWMNextDlgCtl.
	TWMNoParams (see page 5032)	This is record Messages.TWMNoParams.
	TWMNotify (see page 5032)	This is record Messages.TWMNotify.
	TWMNotifyFormat (see page 5033)	This is record Messages.TWMNotifyFormat.
	TWMPaint (see page 5033)	This is record Messages.TWMPaint.
	TWMPaintClipboard (see page 5034)	This is record Messages.TWMPaintClipboard.
	TWMPaletteChanged (see page 5034)	This is record Messages.TWMPaletteChanged.
	TWMPaletteIsChanging (see page 5035)	This is record Messages.TWMPaletteIsChanging.
	TWMParentNotify (see page 5035)	This is record Messages.TWMParentNotify.
	TWMPower (see page 5036)	This is record Messages.TWMPower.
	TWMPrint (see page 5036)	This is record Messages.TWMPrint.
	TWMQueryEndSession (see page 5037)	This is record Messages.TWMQueryEndSession.
	TWMQuit (see page 5037)	This is record Messages.TWMQuit.
	TWMRenderFormat (see page 5038)	This is record Messages.TWMRenderFormat.
	TWMScroll (see page 5038)	This is record Messages.TWMScroll.
	TWMSetCursor (see page 5039)	This is record Messages.TWMSetCursor.
	TWMSetFocus (see page 5039)	This is record Messages.TWMSetFocus.
	TWMSetFont (see page 5040)	This is record Messages.TWMSetFont.
	TWMSetHotKey (see page 5040)	This is record Messages.TWMSetHotKey.
	TWMSetIcon (see page 5041)	This is record Messages.TWMSetIcon.
	TWMSetRedraw (see page 5041)	This is record Messages.TWMSetRedraw.
	TWMSetText (see page 5042)	This is record Messages.TWMSetText.
	TWMSettingChange (see page 5042)	This is record Messages.TWMSettingChange.
	TWMShowWindow (see page 5043)	This is record Messages.TWMShowWindow.
	TWMSize (see page 5043)	This is record Messages.TWMSize.
	TWMSizeClipboard (see page 5044)	This is record Messages.TWMSizeClipboard.
	TWMSpoolerStatus (see page 5044)	This is record Messages.TWMSpoolerStatus.
	TWMStyleChange (see page 5045)	This is record Messages.TWMStyleChange.
	TWMSysCommand (see page 5045)	This is record Messages.TWMSysCommand.
	TWMSysDeadChar (see page 5046)	This is record Messages.TWMSysDeadChar.

	TWMSystemError (see page 5046)	This is record Messages.TWMSystemError.
	TWMTimer (see page 5047)	This is record Messages.TWMTimer.
	TWMUIState (see page 5047)	This is record Messages.TWMUIState.
	TWMVScrollClipboard (see page 5048)	This is record Messages.TWMVScrollClipboard.
	TWMWindowPosMsg (see page 5048)	This is record Messages.TWMWindowPosMsg.
	TWMWinIniChange (see page 5049)	This is record Messages.TWMWinIniChange.

49.2.1 Messages.TMessage Record

TMessage represents a Windows message.

Pascal

```
TMessage = packed record
  Msg: Cardinal;
  case Integer of
    0: (WParam: WPARAM;
        LParam: LPARAM;
        Result: LRESULT;);
    1: (WParamLo: Word;
        WParamHi: Word;
        LParamLo: Word;
        LParamHi: Word;
        ResultLo: Word;
        ResultHi: Word;);
end;
```

C++

```
struct TMessage {
  unsigned Msg;
  WPARAM WParam;
  LPARAM LParam;
  LRESULT Result;
  Word WParamLo;
  Word WParamHi;
  Word LParamLo;
  Word LParamHi;
  Word ResultLo;
  Word ResultHi;
};
```

File

Messages

Description

The TMessage type represents a Windows message in WndProc and other methods.

The Msg field represents the Windows Message code.

The WParam field represents the WParam of the message. To access the low and high words of this field, use the WParamLo and WParamHi fields instead.

The LParam field represents the LParam of the message. To access the low and high words of this field, use the LParamLo and LParamHi fields instead.

The Result field holds the return value. To access the low and high words of this field, use the ResultLo and ResultHi fields instead.

See Also

TWMKey (see page 5019)

TWMMouse (see page 5026)

TWMPaint (see page 5033)

TWMMenuChar (see page 5025)

TWMNotify (see page 5032)

TWMLButtonDown (see page 5062)

TMsg

49.2.2 Messages.TMSHMouseWheel Record

Pascal

```
TMSHMouseWheel = packed record
  Msg: Cardinal;
  WheelDelta: Integer;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TMSHMouseWheel {
  unsigned Msg;
  int WheelDelta;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TMSHMouseWheel.

49.2.3 Messages.TWMActivate Record

Pascal

```
TWMActivate = packed record
  Msg: Cardinal;
  Active: Word;
  Minimized: WordBool;
  ActiveWindow: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMActivate {
  unsigned Msg;
  Word Active;
  WordBool Minimized;
  HWND ActiveWindow;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMActivate.

49.2.4 Messages.TWMActivateApp Record

Pascal

```
TWMActivateApp = packed record
  Msg: Cardinal;
  Active: BOOL;
  ThreadId: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMActivateApp {
  unsigned Msg;
  BOOL Active;
  Longint ThreadId;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMActivateApp.

49.2.5 Messages.TWMAskCBFormatName Record

Pascal

```
TWMAskCBFormatName = packed record
  Msg: Cardinal;
  NameLen: Word;
  Unused: Word;
  FormatName: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMAskCBFormatName {
```

```
  unsigned Msg;
  Word NameLen;
  Word Unused;
  const char * FormatName;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMAskCBFormatName.

49.2.6 Messages.TWMChangeCBChain Record

Pascal

```
TWMChangeCBChain = packed record
  Msg: Cardinal;
  Remove: HWND;
  Next: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMChangeCBChain {
  unsigned Msg;
  HWND Remove;
  HWND Next;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMChangeCBChain.

49.2.7 Messages.TWMCharToItem Record

Pascal

```
TWMCharToItem = packed record
  Msg: Cardinal;
  Key: Word;
  CaretPos: Word;
  ListBox: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMCharToItem {
  unsigned Msg;
  Word Key;
  Word CaretPos;
  HWND ListBox;
```

```
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCharToItem.

49.2.8 Messages.TWMChooseFont_GetLogFont Record

Pascal

```
TWMChooseFont_GetLogFont = packed record
  Msg: Cardinal;
  Unused: Longint;
  LogFont: PLogFont;
  Result: Longint;
end;
```

C++

```
struct TWMChooseFont_GetLogFont {
  unsigned Msg;
  Longint Unused;
  PLogFont LogFont;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMChooseFont_GetLogFont.

49.2.9 Messages.TWMCommand Record

Pascal

```
TWMCommand = packed record
  Msg: Cardinal;
  ItemID: Word;
  NotifyCode: Word;
  Ctl: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMCommand {
  unsigned Msg;
  Word ItemID;
  Word NotifyCode;
  HWND Ctl;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCommand.

49.2.10 Messages.TWMCompacting Record

Pascal

```
TWMCompacting = packed record
  Msg: Cardinal;
  CompactRatio: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMCompacting {
  unsigned Msg;
  Longint CompactRatio;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCompacting.

49.2.11 Messages.TWMCompareItem Record

Pascal

```
TWMCompareItem = packed record
  Msg: Cardinal;
  Ctl: HWnd;
  CompareItemStruct: PCompareItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCompareItem {
  unsigned Msg;
  HWnd Ctl;
  PCompareItemStruct CompareItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCompareItem.

49.2.12 Messages.TWMContextMenu Record

Pascal

```
TWMContextMenu = packed record
  Msg: Cardinal;
  hWnd: HWND;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMContextMenu {
  unsigned Msg;
  HWND hWnd;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMContextMenu.

49.2.13 Messages.TWMCopyData Record

Pascal

```
TWMCopyData = packed record
  Msg: Cardinal;
  From: HWND;
  CopyDataStruct: PCopyDataStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCopyData {
  unsigned Msg;
  HWND From;
  PCopyDataStruct CopyDataStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCopyData.

49.2.14 Messages.TWMCreate Record

Pascal

```
TWMCreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  CreateStruct: PCreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCreate {
  unsigned Msg;
  int Unused;
  PCreateStruct CreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCreate.

49.2.15 Messages.TWMCtlColor Record

Pascal

```
TWMCtlColor = packed record
  Msg: Cardinal;
  ChildDC: HDC;
  ChildWnd: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMCtlColor {
  unsigned Msg;
  HDC ChildDC;
  HWND ChildWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCtlColor.

49.2.16 Messages.TWMDDE_Ack Record

Pascal

```
TWMDDE_Ack = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  case Word of
    WM_DDE_EXECUTE: (PackedVal: Longint;);
    WM_DDE_INITIATE: (App: Word;
                      Topic: Word;
                      Result: Longint;);
end;
```

C++

```
struct TWMDDE_Ack {
  unsigned Msg;
  HWND PostingApp;
  Word App;
  Word Topic;
  Longint Result;
  Longint PackedVal;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Ack.

49.2.17 Messages.TWMDDE_Advise Record

Pascal

```
TWMDDE_Advise = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Advise {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Advise.

49.2.18 Messages.TWMDDE_Data Record

Pascal

```
TWMDDE_Data = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Data {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Data.

49.2.19 Messages.TWMDDE_Execute Record

Pascal

```
TWMDDE_Execute = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Commands: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Execute {
  unsigned Msg;
  HWND PostingApp;
  THandle Commands;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Execute.

49.2.20 Messages.TWMDDE_Initiate Record

Pascal

```
TWMDDE_Initiate = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  App: Word;
  Topic: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Initiate {
  unsigned Msg;
  HWND PostingApp;
  Word App;
  Word Topic;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Initiate.

49.2.21 Messages.TWMDDE_Poke Record

Pascal

```
TWMDDE_Poke = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Poke {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Poke.

49.2.22 Messages.TWMDDE_Request Record

Pascal

```
TWMDDE_Request = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Format: Word;
  Item: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Request {
  unsigned Msg;
  HWND PostingApp;
  Word Format;
  Word Item;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Request.

49.2.23 Messages.TWMDDE_Terminate Record

Pascal

```
TWMDDE_Terminate = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Terminate {
  unsigned Msg;
  HWND PostingApp;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Terminate.

49.2.24 Messages.TWMDDE_Unadvise Record

Pascal

```
TWMDDE_Unadvise = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Format: Word;
  Item: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Unadvise {
  unsigned Msg;
  HWND PostingApp;
  Word Format;
  Word Item;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Unadvise.

49.2.25 Messages.TWMDeleteItem Record

Pascal

```
TWMDeleteItem = packed record
  Msg: Cardinal;
  Ctl: HWND;
  DeleteItemStruct: PDeleteItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMDeleteItem {
  unsigned Msg;
  HWND Ctl;
  PDeleteItemStruct DeleteItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDeleteItem.

49.2.26 Messages.TWMDevModeChange Record

Pascal

```
TWMDevModeChange = packed record
  Msg: Cardinal;
  Unused: Integer;
  Device: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMDevModeChange {
  unsigned Msg;
  int Unused;
  const char * Device;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDevModeChange.

49.2.27 Messages.TWMDisplayChange Record

Pascal

```
TWMDisplayChange = packed record
  Msg: Cardinal;
  BitsPerPixel: Integer;
  Width: Word;
  Height: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDisplayChange {
  unsigned Msg;
  int BitsPerPixel;
  Word Width;
  Word Height;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDisplayChange.

49.2.28 Messages.TWMDrawItem Record

Pascal

```
TWMDrawItem = packed record
  Msg: Cardinal;
  Ctl: HWND;
  DrawItemStruct: PDrawItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMDrawItem {
  unsigned Msg;
  HWND Ctl;
  PDrawItemStruct DrawItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDrawItem.

49.2.29 Messages.TWMDropFiles Record

Pascal

```
TWMDropFiles = packed record
  Msg: Cardinal;
  Drop: THANDLE;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDropFiles {
  unsigned Msg;
  THANDLE Drop;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDropFiles.

49.2.30 Messages.TWMEnable Record

Pascal

```
TWMEnable = packed record
  Msg: Cardinal;
  Enabled: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEnable {
  unsigned Msg;
  LongBool Enabled;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnable.

49.2.31 Messages.TWMEndSession Record

Pascal

```
TWMEndSession = packed record
  Msg: Cardinal;
  EndSession: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEndSession {
  unsigned Msg;
  LongBool EndSession;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEndSession.

49.2.32 Messages.TWMEnterIdle Record

Pascal

```
TWMEnterIdle = packed record
  Msg: Cardinal;
  Source: Longint;
  IdleWnd: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMEnterIdle {
  unsigned Msg;
  Longint Source;
  HWND IdleWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnterIdle.

49.2.33 Messages.TWMEnterMenuLoop Record

Pascal

```
TWMEnterMenuLoop = packed record
  Msg: Cardinal;
  IsTrackPopupMenu: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEnterMenuLoop {
  unsigned Msg;
  LongBool IsTrackPopupMenu;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnterMenuLoop.

49.2.34 Messages.TWMEraseBkgnd Record

Pascal

```
TWMEraseBkgnd = packed record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEraseBkgnd {
  unsigned Msg;
  HDC DC;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEraseBkgnd.

49.2.35 Messages.TWMGetIcon Record

Pascal

```
TWMGetIcon = packed record
  Msg: Cardinal;
  BigIcon: Longbool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMGetIcon {
  unsigned Msg;
  Longbool BigIcon;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetIcon.

49.2.36 Messages.TWMGetMinMaxInfo Record

Pascal

```
TWMGetMinMaxInfo = packed record
  Msg: Cardinal;
  Unused: Integer;
  MinMaxInfo: PMinMaxInfo;
  Result: Longint;
end;
```

C++

```
struct TWMGetMinMaxInfo {
  unsigned Msg;
  int Unused;
  PMinMaxInfo MinMaxInfo;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetMinMaxInfo.

49.2.37 Messages.TWMGetText Record

Pascal

```
TWMGetText = packed record
  Msg: Cardinal;
  TextMax: Integer;
  Text: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMGetText {
  unsigned Msg;
  int TextMax;
  const char * Text;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetText.

49.2.38 Messages.TWMHelp Record

Pascal

```
TWMHelp = packed record
  Msg: Cardinal;
  Unused: Integer;
  HelpInfo: PHelpInfo;
  Result: Longint;
end;
```

C++

```
struct TWMHelp {
  unsigned Msg;
  int Unused;
  PHelpInfo HelpInfo;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHelp.

49.2.39 Messages.TWMHotKey Record

Pascal

```
TWMHotKey = packed record
  Msg: Cardinal;
  HotKey: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMHotKey {
  unsigned Msg;
  Longint HotKey;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHotKey.

49.2.40 Messages.TWMHScrollClipboard Record

Pascal

```
TWMHScrollClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  ScrollCode: Word;
  Pos: Word;
  Result: Longint;
end;
```

C++

```
struct TWMHScrollClipboard {
  unsigned Msg;
  HWND Viewer;
  Word ScrollCode;
  Word Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHScrollClipboard.

49.2.41 Messages.TWMInitDialog Record

Pascal

```
TWMInitDialog = packed record
  Msg: Cardinal;
  Focus: HWND;
  InitParam: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMInitDialog {
  unsigned Msg;
  HWND Focus;
  Longint InitParam;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitDialog.

49.2.42 Messages.TWMInitMenu Record

Pascal

```
TWMInitMenu = packed record
  Msg: Cardinal;
  Menu: HMENU;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMInitMenu {
  unsigned Msg;
  HMENU Menu;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitMenu.

49.2.43 Messages.TWMInitMenuPopup Record

Pascal

```
TWMInitMenuPopup = packed record
  Msg: Cardinal;
  MenuPopup: HMENU;
  Pos: Smallint;
  SystemMenu: WordBool;
  Result: Longint;
end;
```

C++

```
struct TWMInitMenuPopup {
  unsigned Msg;
  HMENU MenuPopup;
  Smallint Pos;
  WordBool SystemMenu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitMenuPopup.

49.2.44 Messages.TWMKey Record

Pascal

```
TWMKey = packed record
  Msg: Cardinal;
  CharCode: Word;
  Unused: Word;
  KeyData: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMKey {
  unsigned Msg;
  Word CharCode;
  Word Unused;
  Longint KeyData;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMKey.

49.2.45 Messages.TWMKillFocus Record

Pascal

```
TWMKillFocus = packed record
  Msg: Cardinal;
  FocusedWnd: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMKillFocus {
  unsigned Msg;
  HWND FocusedWnd;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMKillFocus.

49.2.46 Messages.TWMMDIActivate Record

Pascal

```
TWMMDIActivate = packed record
  Msg: Cardinal;
  case Integer of
    0: (ChildWnd: HWND;);
    1: (DeactiveWnd: HWND;
        ActiveWnd: HWND;
        Result: Longint;);
end;
```

C++

```
struct TWMMDIActivate {
  unsigned Msg;
  HWND ChildWnd;
  HWND DeactiveWnd;
  HWND ActiveWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIActivate.

49.2.47 Messages.TWMMDICascade Record

Pascal

```
TWMMDICascade = packed record
  Msg: Cardinal;
  Cascade: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDICascade {
  unsigned Msg;
  Longint Cascade;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDICascade.

49.2.48 Messages.TWMMDICreate Record

Pascal

```
TWMMDICreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  MDICreateStruct: PMDICreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMMDICreate {
  unsigned Msg;
  int Unused;
  PMDICreateStruct MDICreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDICreate.

49.2.49 Messages.TWMMDIDestroy Record

Pascal

```
TWMMDIDestroy = packed record
  Msg: Cardinal;
  Child: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIDestroy {
  unsigned Msg;
  HWND Child;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIDestroy.

49.2.50 Messages.TWMMDIMaximize Record

Pascal

```
TWMMDIMaximize = packed record
  Msg: Cardinal;
  Maximize: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIMaximize {
  unsigned Msg;
  HWND Maximize;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIMaximize.

49.2.51 Messages.TWMMDINext Record

Pascal

```
TWMMDINext = packed record
  Msg: Cardinal;
  Child: HWND;
  Next: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDINext {
  unsigned Msg;
  HWND Child;
  Longint Next;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDINext.

49.2.52 Messages.TWMMDIRestore Record

Pascal

```
TWMMDIRestore = packed record
  Msg: Cardinal;
  IDChild: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIRestore {
  unsigned Msg;
  HWND IDChild;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIRestore.

49.2.53 Messages.TWMMDISetMenu Record

Pascal

```
TWMMDISetMenu = packed record
  Msg: Cardinal;
  MenuFrame: HMENU;
  MenuWindow: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMDISetMenu {
  unsigned Msg;
  HMENU MenuFrame;
  HMENU MenuWindow;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDISetMenu.

49.2.54 Messages.TWMMDITile Record

Pascal

```
TWMMDITile = packed record
  Msg: Cardinal;
  Tile: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDITile {
  unsigned Msg;
  Longint Tile;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDITile.

49.2.55 Messages.TWMMeasureItem Record

Pascal

```
TWMMeasureItem = packed record
  Msg: Cardinal;
  IDCtl: HWnd;
  MeasureItemStruct: PMeasureItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMMeasureItem {
  unsigned Msg;
  HWnd IDCtl;
  PMeasureItemStruct MeasureItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMeasureItem.

49.2.56 Messages.TWMMenuChar Record

Pascal

```
TWMMenuChar = packed record
  Msg: Cardinal;
  User: Char;
  MenuFlag: Word;
  Menu: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMenuChar {
  unsigned Msg;
  Char User;
  Word MenuFlag;
  HMENU Menu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMenuChar.

49.2.57 Messages.TWMMenuSelect Record

Pascal

```
TWMMenuSelect = packed record
  Msg: Cardinal;
  IDItem: Word;
  MenuFlag: Word;
  Menu: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMenuSelect {
  unsigned Msg;
  Word IDItem;
  Word MenuFlag;
  HMENU Menu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMenuSelect.

49.2.58 Messages.TWMMouse Record

Pascal

```
TWMMouse = packed record
  Msg: Cardinal;
  Keys: Longint;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMouse {
  unsigned Msg;
  Longint Keys;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouse.

49.2.59 Messages.TWMMouseActivate Record

Pascal

```
TWMMouseActivate = packed record
  Msg: Cardinal;
  TopLevel: HWND;
  HitTestCode: Word;
  MouseMsg: Word;
  Result: Longint;
end;
```

C++

```
struct TWMMouseActivate {
  unsigned Msg;
  HWND TopLevel;
  Word HitTestCode;
  Word MouseMsg;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouseActivate.

49.2.60 Messages.TWMMouseWheel Record

Pascal

```
TWMMouseWheel = packed record
  Msg: Cardinal;
  Keys: SmallInt;
  WheelDelta: SmallInt;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMouseWheel {
  unsigned Msg;
  SmallInt Keys;
  SmallInt WheelDelta;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouseWheel.

49.2.61 Messages.TWMMove Record

Pascal

```
TWMMove = packed record
  Msg: Cardinal;
  Unused: Integer;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMove {
  unsigned Msg;
  int Unused;
  Smallint XPos;
  Smallint YPos;
```

```
    TSmallPoint Pos;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMMove.

49.2.62 Messages.TWMMoving Record

Pascal

```
  TWMMoving = packed record
    Msg: Cardinal;
    Edge: Integer;
    DragRect: PRect;
  end;
```

C++

```
  struct TWMMoving {
    unsigned Msg;
    int Edge;
    PRect DragRect;
  };
```

File

Messages

Description

This is record Messages.TWMMoving.

49.2.63 Messages.TWMNCActivate Record

Pascal

```
  TWMNCActivate = packed record
    Msg: Cardinal;
    Active: BOOL;
    Unused: Longint;
    Result: Longint;
  end;
```

C++

```
  struct TWMNCActivate {
    unsigned Msg;
    BOOL Active;
    Longint Unused;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMNCActivate.

49.2.64 Messages.TWMNCCalcSize Record

Pascal

```
TWMNCCalcSize = packed record
  Msg: Cardinal;
  CalcValidRects: BOOL;
  CalcSize_Params: PNCCalcSizeParams;
  Result: Longint;
end;
```

C++

```
struct TWMNCCalcSize {
  unsigned Msg;
  BOOL CalcValidRects;
  PNCCalcSizeParams CalcSize_Params;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCCalcSize.

49.2.65 Messages.TWMNCCreate Record

Pascal

```
TWMNCCreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  CreateStruct: PCreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMNCCreate {
  unsigned Msg;
  int Unused;
  PCreateStruct CreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCCreate.

49.2.66 Messages.TWMNCHitMessage Record

Pascal

```
TWMNCHitMessage = packed record
  Msg: Cardinal;
  HitTest: Longint;
  XCursor: Smallint;
  YCursor: Smallint;
  Result: Longint;
end;
```

C++

```
struct TWMNCHitMessage {
  unsigned Msg;
  Longint HitTest;
  Smallint XCursor;
  Smallint YCursor;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCHitMessage.

49.2.67 Messages.TWMNCHitTest Record

Pascal

```
TWMNCHitTest = packed record
  Msg: Cardinal;
  Unused: Longint;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMNCHitTest {
  unsigned Msg;
  Longint Unused;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCHitTest.

49.2.68 Messages.TWMNCPaint Record

Pascal

```
TWMNCPaint = packed record
  Msg: Cardinal;
  RGN: HRGN;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMNCPaint {
  unsigned Msg;
  HRGN RGN;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCPaint.

49.2.69 Messages.TWMNextDlgCtl Record

Pascal

```
TWMNextDlgCtl = packed record
  Msg: Cardinal;
  CtlFocus: Longint;
  Handle: WordBool;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMNextDlgCtl {
  unsigned Msg;
  Longint CtlFocus;
  WordBool Handle;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNextDlgCtl.

49.2.70 Messages.TWMNoParams Record

Pascal

```
TWMNoParams = packed record
  Msg: Cardinal;
  Unused: array[0..3] of Word;
  Result: Longint;
end;
```

C++

```
struct TWMNoParams {
  unsigned Msg;
  array[0..3] of Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNoParams.

49.2.71 Messages.TWMNotify Record

Pascal

```
TWMNotify = packed record
  Msg: Cardinal;
  IDCtrl: Longint;
  NMHdr: PNMHdr;
  Result: Longint;
end;
```

C++

```
struct TWMNotify {
  unsigned Msg;
  Longint IDCtrl;
  PNMHdr NMHdr;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNotify.

49.2.72 Messages.TWMNotifyFormat Record

Pascal

```
TWMNotifyFormat = packed record
  Msg: Cardinal;
  From: HWND;
  Command: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMNotifyFormat {
  unsigned Msg;
  HWND From;
  Longint Command;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNotifyFormat.

49.2.73 Messages.TWMPaint Record

Pascal

```
TWMPaint = packed record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaint {
  unsigned Msg;
  HDC DC;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaint.

49.2.74 Messages.TWMPaintClipboard Record

Pascal

```
TWMPaintClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  PaintStruct: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMPaintClipboard {
  unsigned Msg;
  HWND Viewer;
  THandle PaintStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaintClipboard.

49.2.75 Messages.TWMPaletteChanged Record

Pascal

```
TWMPaletteChanged = packed record
  Msg: Cardinal;
  PalChg: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaletteChanged {
  unsigned Msg;
  HWND PalChg;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaletteChanged.

49.2.76 Messages.TWMPaletteIsChanging Record

Pascal

```
TWMPaletteIsChanging = packed record
  Msg: Cardinal;
  Realize: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaletteIsChanging {
  unsigned Msg;
  HWND Realize;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaletteIsChanging.

49.2.77 Messages.TWMParentNotify Record

Pascal

```
TWMParentNotify = packed record
  Msg: Cardinal;
  Event: Word;
  case Word of
    0: (Value1: Word;
        Value2: Longint;
        Result: Longint;);
    WM_DESTROY: (ChildID: Word;
                 ChildWnd: HWnd;);
    WM_RBUTTONDOWN: (Value: Word;
                     XPos: Smallint;
                     YPos: Smallint;);
end;
```

C++

```
struct TWMParentNotify {
  unsigned Msg;
  Word Event;
  Word ChildID;
  HWnd ChildWnd;
  Word Value;
  Smallint XPos;
  Smallint YPos;
  Word Value1;
  Longint Value2;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMParentNotify.

49.2.78 Messages.TWMPower Record

Pascal

```
TWMPower = packed record
  Msg: Cardinal;
  PowerEvt: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPower {
  unsigned Msg;
  Longint PowerEvt;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPower.

49.2.79 Messages.TWMPrint Record

Pascal

```
TWMPrint = packed record
  Msg: Cardinal;
  DC: HDC;
  Flags: Cardinal;
  Result: Integer;
end;
```

C++

```
struct TWMPrint {
  unsigned Msg;
  HDC DC;
  unsigned Flags;
  int Result;
};
```

File

Messages

Description

This is record Messages.TWMPrint.

49.2.80 Messages.TWMQueryEndSession Record

Pascal

```
TWMQueryEndSession = packed record
  Msg: Cardinal;
  Source: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMQueryEndSession {
  unsigned Msg;
  Longint Source;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMQueryEndSession.

49.2.81 Messages.TWMQuit Record

Pascal

```
TWMQuit = packed record
  Msg: Cardinal;
  ExitCode: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMQuit {
  unsigned Msg;
  Longint ExitCode;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMQuit.

49.2.82 Messages.TWMRenderFormat Record

Pascal

```
TWMRenderFormat = packed record
  Msg: Cardinal;
  Format: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMRenderFormat {
  unsigned Msg;
  Longint Format;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMRenderFormat.

49.2.83 Messages.TWMScroll Record

Pascal

```
TWMScroll = packed record
  Msg: Cardinal;
  ScrollCode: Smallint;
  Pos: Smallint;
  ScrollBar: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMScroll {
  unsigned Msg;
  Smallint ScrollCode;
  Smallint Pos;
  HWND ScrollBar;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMScroll.

49.2.84 Messages.TWMSetCursor Record

Pascal

```
TWMSetCursor = packed record
  Msg: Cardinal;
  CursorWnd: HWND;
  HitTest: SmallInt;
  MouseMsg: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSetCursor {
  unsigned Msg;
  HWND CursorWnd;
  SmallInt HitTest;
  Word MouseMsg;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetCursor.

49.2.85 Messages.TWMSetFocus Record

Pascal

```
TWMSetFocus = packed record
  Msg: Cardinal;
  FocusedWnd: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetFocus {
  unsigned Msg;
  HWND FocusedWnd;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetFocus.

49.2.86 Messages.TWMSetFont Record

Pascal

```
TWMSetFont = packed record
  Msg: Cardinal;
  Font: HFONT;
  Redraw: WordBool;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSetFont {
  unsigned Msg;
  HFONT Font;
  WordBool Redraw;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetFont.

49.2.87 Messages.TWMSetHotKey Record

Pascal

```
TWMSetHotKey = packed record
  Msg: Cardinal;
  Key: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetHotKey {
  unsigned Msg;
  Longint Key;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetHotKey.

49.2.88 Messages.TWMSetIcon Record

Pascal

```
TWMSetIcon = packed record
  Msg: Cardinal;
  BigIcon: Longbool;
  Icon: HICON;
  Result: Longint;
end;
```

C++

```
struct TWMSetIcon {
  unsigned Msg;
  Longbool BigIcon;
  HICON Icon;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetIcon.

49.2.89 Messages.TWMSetRedraw Record

Pascal

```
TWMSetRedraw = packed record
  Msg: Cardinal;
  Redraw: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetRedraw {
  unsigned Msg;
  Longint Redraw;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetRedraw.

49.2.90 Messages.TWMSetText Record

Pascal

```
TWMSetText = packed record
  Msg: Cardinal;
  Unused: Longint;
  Text: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMSetText {
  unsigned Msg;
  Longint Unused;
  const char * Text;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetText.

49.2.91 Messages.TWMSettingChange Record

Pascal

```
TWMSettingChange = packed record
  Msg: Cardinal;
  Flag: Integer;
  Section: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMSettingChange {
  unsigned Msg;
  int Flag;
  const char * Section;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSettingChange.

49.2.92 Messages.TWMShowWindow Record

Pascal

```
TWMShowWindow = packed record
  Msg: Cardinal;
  Show: BOOL;
  Status: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMShowWindow {
  unsigned Msg;
  BOOL Show;
  Longint Status;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMShowWindow.

49.2.93 Messages.TWMSize Record

Pascal

```
TWMSize = packed record
  Msg: Cardinal;
  SizeType: Longint;
  Width: Word;
  Height: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSize {
  unsigned Msg;
  Longint SizeType;
  Word Width;
  Word Height;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSize.

49.2.94 Messages.TWMSizeClipboard Record

Pascal

```
TWMSizeClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  RC: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMSizeClipboard {
  unsigned Msg;
  HWND Viewer;
  THandle RC;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSizeClipboard.

49.2.95 Messages.TWMSpoolerStatus Record

Pascal

```
TWMSpoolerStatus = packed record
  Msg: Cardinal;
  JobStatus: Longint;
  JobsLeft: Word;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSpoolerStatus {
  unsigned Msg;
  Longint JobStatus;
  Word JobsLeft;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSpoolerStatus.

49.2.96 Messages.TWMStyleChange Record

Pascal

```
TWMStyleChange = packed record
  Msg: Cardinal;
  StyleType: Longint;
  StyleStruct: PStyleStruct;
  Result: Longint;
end;
```

C++

```
struct TWMStyleChange {
  unsigned Msg;
  Longint StyleType;
  PStyleStruct StyleStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMStyleChange.

49.2.97 Messages.TWMSysCommand Record

Pascal

```
TWMSysCommand = packed record
  Msg: Cardinal;
  CmdType: Longint;
  case Longint of
    SC_HOTKEY: (ActivateWnd: HWND;);
    SC_KEYMENU: (Key: Word;);
    SC_VSCROLL: (XPos: Smallint;
                 YPos: Smallint;
                 Result: Longint;);
end;
```

C++

```
struct TWMSysCommand {
  unsigned Msg;
  Longint CmdType;
  HWND ActivateWnd;
  Word Key;
  Smallint XPos;
  Smallint YPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSysCommand.

49.2.98 Messages.TWMSysDeadChar Record

Pascal

```
TWMSysDeadChar = packed record
  Msg: Cardinal;
  CharCode: Word;
  Unused: Word;
  KeyData: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSysDeadChar {
  unsigned Msg;
  Word CharCode;
  Word Unused;
  Longint KeyData;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSysDeadChar.

49.2.99 Messages.TWMSystemError Record

Pascal

```
TWMSystemError = packed record
  Msg: Cardinal;
  ErrSpec: Word;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSystemError {
  unsigned Msg;
  Word ErrSpec;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSystemError.

49.2.100 Messages.TWMTimer Record

Pascal

```
TWMTimer = packed record
  Msg: Cardinal;
  TimerID: Longint;
  TimerProc: TFarProc;
  Result: Longint;
end;
```

C++

```
struct TWMTimer {
  unsigned Msg;
  Longint TimerID;
  TFarProc TimerProc;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMTimer.

49.2.101 Messages.TWMUIState Record

Pascal

```
TWMUIState = packed record
  Msg: Cardinal;
  Action: Word;
  Flags: Word;
  Unused: Longint;
end;
```

C++

```
struct TWMUIState {
  unsigned Msg;
  Word Action;
  Word Flags;
  Longint Unused;
};
```

File

Messages

Description

This is record Messages.TWMUIState.

49.2.102 Messages.TWMVScrollClipboard Record

Pascal

```
TWMVScrollClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  ScollCode: Word;
  ThumbPos: Word;
  Result: Longint;
end;
```

C++

```
struct TWMVScrollClipboard {
  unsigned Msg;
  HWND Viewer;
  Word ScollCode;
  Word ThumbPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMVScrollClipboard.

49.2.103 Messages.TWMWindowPosMsg Record

Pascal

```
TWMWindowPosMsg = packed record
  Msg: Cardinal;
  Unused: Integer;
  WindowPos: PWindowPos;
  Result: Longint;
end;
```

C++

```
struct TWMWindowPosMsg {
  unsigned Msg;
  int Unused;
  PWindowPos WindowPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMWindowPosMsg.

49.2.104 Messages.TWMWinIniChange Record

Pascal

```
TWMWinIniChange = packed record
  Msg: Cardinal;
  Unused: Integer;
  Section: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMWinIniChange {
  unsigned Msg;
  int Unused;
  const char * Section;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMWinIniChange.

49.3 Types

The following table lists types in this documentation.

Types

Name	Description
PMessage (see page 5051)	This is type Messages.PMessage.
TWMCancelMode (see page 5052)	This is type Messages.TWMCancelMode.
TWMChangeUIState (see page 5052)	This is type Messages.TWMChangeUIState.
TWMChar (see page 5052)	This is type Messages.TWMChar.
TWMChildActivate (see page 5053)	This is type Messages.TWMChildActivate.
TWMClear (see page 5053)	This is type Messages.TWMClear.
TWMClose (see page 5053)	This is type Messages.TWMClose.
TWMCopy (see page 5054)	This is type Messages.TWMCopy.
TWMCtlColorBtn (see page 5054)	This is type Messages.TWMCtlColorBtn.
TWMCtlColorDlg (see page 5054)	This is type Messages.TWMCtlColorDlg.
TWMCtlColorEdit (see page 5055)	This is type Messages.TWMCtlColorEdit.
TWMCtlColorListbox (see page 5055)	This is type Messages.TWMCtlColorListbox.
TWMCtlColorMsgbox (see page 5055)	This is type Messages.TWMCtlColorMsgbox.
TWMCtlColorScrollbar (see page 5056)	This is type Messages.TWMCtlColorScrollbar.
TWMCtlColorStatic (see page 5056)	This is type Messages.TWMCtlColorStatic.
TWMCut (see page 5056)	This is type Messages.TWMCut.
TWMDeadChar (see page 5057)	This is type Messages.TWMDeadChar.
TWMDestroy (see page 5057)	This is type Messages.TWMDestroy.
TWMDestroyClipboard (see page 5057)	This is type Messages.TWMDestroyClipboard.
TWMDrawClipboard (see page 5058)	This is type Messages.TWMDrawClipboard.
TWMExitMenuLoop (see page 5058)	This is type Messages.TWMExitMenuLoop.
TWMFontChange (see page 5058)	This is type Messages.TWMFontChange.
TWMGetDlgCode (see page 5059)	This is type Messages.TWMGetDlgCode.
TWMGetFont (see page 5059)	This is type Messages.TWMGetFont.
TWMGetHotKey (see page 5059)	This is type Messages.TWMGetHotKey.
TWMGetTextLength (see page 5060)	This is type Messages.TWMGetTextLength.
TWMHScroll (see page 5060)	This is type Messages.TWMHScroll.
TWMIconEraseBkgnd (see page 5060)	This is type Messages.TWMIconEraseBkgnd.
TWMKeyDown (see page 5061)	This is type Messages.TWMKeyDown.
TWMKeyUp (see page 5061)	This is type Messages.TWMKeyUp.
TWMLButtonDblClk (see page 5061)	This is type Messages.TWMLButtonDblClk.
TWMLButtonDown (see page 5062)	TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.
TWMLButtonUp (see page 5062)	This is type Messages.TWMLButtonUp.
TWMMButtonDblClk (see page 5062)	This is type Messages.TWMMButtonDblClk.
TWMMButtonDown (see page 5063)	This is type Messages.TWMMButtonDown.
TWMMButtonUp (see page 5063)	This is type Messages.TWMMButtonUp.
TWMMDIGetActive (see page 5063)	This is type Messages.TWMMDIGetActive.
TWMMDIIconArrange (see page 5064)	This is type Messages.TWMMDIIconArrange.
TWMMDIRefreshMenu (see page 5064)	This is type Messages.TWMMDIRefreshMenu.
TWMMouseMove (see page 5064)	This is type Messages.TWMMouseMove.
TWMNCDestroy (see page 5065)	This is type Messages.TWMNCDestroy.
TWMNCLButtonDblClk (see page 5065)	This is type Messages.TWMNCLButtonDblClk.
TWMNCLButtonDown (see page 5065)	This is type Messages.TWMNCLButtonDown.
TWMNCLButtonUp (see page 5066)	This is type Messages.TWMNCLButtonUp.

TWMNCMButtonDblClk (see page 5066)	This is type Messages.TWMNCMButtonDblClk.
TWMNCMButtonDown (see page 5066)	This is type Messages.TWMNCMButtonDown.
TWMNCMButtonUp (see page 5067)	This is type Messages.TWMNCMButtonUp.
TWMNCMouseMove (see page 5067)	This is type Messages.TWMNCMouseMove.
TWMNCRButtonDblClk (see page 5067)	This is type Messages.TWMNCRButtonDblClk.
TWMNCRButtonDown (see page 5068)	This is type Messages.TWMNCRButtonDown.
TWMNCRButtonUp (see page 5068)	This is type Messages.TWMNCRButtonUp.
TWMPaintIcon (see page 5068)	This is type Messages.TWMPaintIcon.
TWMPaste (see page 5069)	This is type Messages.TWMPaste.
TWMPrintClient (see page 5069)	This is type Messages.TWMPrintClient.
TWMQueryDragIcon (see page 5069)	This is type Messages.TWMQueryDragIcon.
TWMQueryNewPalette (see page 5070)	This is type Messages.TWMQueryNewPalette.
TWMQueryOpen (see page 5070)	This is type Messages.TWMQueryOpen.
TWMQueryUIState (see page 5070)	This is type Messages.TWMQueryUIState.
TWMQueueSync (see page 5071)	This is type Messages.TWMQueueSync.
TWMRButtonDblClk (see page 5071)	This is type Messages.TWMRButtonDblClk.
TWMRButtonDown (see page 5071)	This is type Messages.TWMRButtonDown.
TWMRButtonUp (see page 5072)	This is type Messages.TWMRButtonUp.
TWMRenderAllFormats (see page 5072)	This is type Messages.TWMRenderAllFormats.
TWMStyleChanged (see page 5072)	This is type Messages.TWMStyleChanged.
TWMStyleChanging (see page 5073)	This is type Messages.TWMStyleChanging.
TWMSysChar (see page 5073)	This is type Messages.TWMSysChar.
TWMSysColorChange (see page 5073)	This is type Messages.TWMSysColorChange.
TWMSysKeyDown (see page 5074)	This is type Messages.TWMSysKeyDown.
TWMSysKeyUp (see page 5074)	This is type Messages.TWMSysKeyUp.
TWMTimeChange (see page 5074)	This is type Messages.TWMTimeChange.
TWMUndo (see page 5075)	This is type Messages.TWMUndo.
TWMUpdateUIState (see page 5075)	This is type Messages.TWMUpdateUIState.
TWMVKeyToItem (see page 5075)	This is type Messages.TWMVKeyToItem.
TWMVScroll (see page 5076)	This is type Messages.TWMVScroll.
TWMWindowPosChanged (see page 5076)	This is type Messages.TWMWindowPosChanged.
TWMWindowPosChanging (see page 5076)	This is type Messages.TWMWindowPosChanging.

49.3.1 Messages.PMessage Type

Pascal

```
PMessage = ^TMessage;
```

C++

```
^TMessage PMessage;
```

File

Messages

Description

This is type Messages.PMessage.

49.3.2 Messages.TWMCancelMode Type

Pascal

```
TWMCancelMode = TWMNoParams;
```

C++

```
TWMNoParams TWMCancelMode;
```

File

Messages

Description

This is type Messages.TWMCancelMode.

49.3.3 Messages.TWMChangeUIState Type

Pascal

```
TWMChangeUIState = TWMUIState;
```

C++

```
TWMUIState TWMChangeUIState;
```

File

Messages

Description

This is type Messages.TWMChangeUIState.

49.3.4 Messages.TWMChar Type

Pascal

```
TWMChar = TWMKey;
```

C++

```
TWMKey TWMChar;
```

File

Messages

Description

This is type Messages.TWMChar.

49.3.5 Messages.TWMChildActivate Type

Pascal

```
TWMChildActivate = TWMNoParams;
```

C++

```
TWMNoParams TWMChildActivate;
```

File

Messages

Description

This is type Messages.TWMChildActivate.

49.3.6 Messages.TWMClear Type

Pascal

```
TWMClear = TWMNoParams;
```

C++

```
TWMNoParams TWMClear;
```

File

Messages

Description

This is type Messages.TWMClear.

49.3.7 Messages.TWMClose Type

Pascal

```
TWMClose = TWMNoParams;
```

C++

```
TWMNoParams TWMClose;
```

File

Messages

Description

This is type Messages.TWMClose.

49.3.8 Messages.TWMCopy Type

Pascal

```
TWMCopy = TWMNoParams;
```

C++

```
TWMNoParams TWMCopy;
```

File

Messages

Description

This is type Messages.TWMCopy.

49.3.9 Messages.TWMCtlColorBtn Type

Pascal

```
TWMCtlColorBtn = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorBtn;
```

File

Messages

Description

This is type Messages.TWMCtlColorBtn.

49.3.10 Messages.TWMCtlColorDlg Type

Pascal

```
TWMCtlColorDlg = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorDlg;
```

File

Messages

Description

This is type Messages.TWMCtlColorDlg.

49.3.11 Messages.TWMCtlColorEdit Type

Pascal

```
TWMCtlColorEdit = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorEdit;
```

File

Messages

Description

This is type Messages.TWMCtlColorEdit.

49.3.12 Messages.TWMCtlColorListbox Type

Pascal

```
TWMCtlColorListbox = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorListbox;
```

File

Messages

Description

This is type Messages.TWMCtlColorListbox.

49.3.13 Messages.TWMCtlColorMsgbox Type

Pascal

```
TWMCtlColorMsgbox = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorMsgbox;
```

File

Messages

Description

This is type Messages.TWMCtlColorMsgbox.

49.3.14 Messages.TWMCtlColorScrollbar Type

Pascal

```
TWMCtlColorScrollbar = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorScrollbar;
```

File

Messages

Description

This is type Messages.TWMCtlColorScrollbar.

49.3.15 Messages.TWMCtlColorStatic Type

Pascal

```
TWMCtlColorStatic = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorStatic;
```

File

Messages

Description

This is type Messages.TWMCtlColorStatic.

49.3.16 Messages.TWMCut Type

Pascal

```
TWMCut = TWMNoParams;
```

C++

```
TWMNoParams TWMCut;
```

File

Messages

Description

This is type Messages.TWMCut.

49.3.17 Messages.TWMDeadChar Type

Pascal

```
TWMDeadChar = TWMChar;
```

C++

```
TWMChar TWMDeadChar;
```

File

Messages

Description

This is type Messages.TWMDeadChar.

49.3.18 Messages.TWMDestroy Type

Pascal

```
TWMDestroy = TWMNoParams;
```

C++

```
TWMNoParams TWMDestroy;
```

File

Messages

Description

This is type Messages.TWMDestroy.

49.3.19 Messages.TWMDestroyClipboard Type

Pascal

```
TWMDestroyClipboard = TWMNoParams;
```

C++

```
TWMNoParams TWMDestroyClipboard;
```

File

Messages

Description

This is type Messages.TWMDestroyClipboard.

49.3.20 Messages.TWMDrawClipboard Type

Pascal

```
TWMDrawClipboard = TWMNoParams;
```

C++

```
TWMNoParams TWMDrawClipboard;
```

File

Messages

Description

This is type Messages.TWMDrawClipboard.

49.3.21 Messages.TWMExitMenuLoop Type

Pascal

```
TWMExitMenuLoop = TWMEnterMenuLoop;
```

C++

```
TWMEnterMenuLoop TWMExitMenuLoop;
```

File

Messages

Description

This is type Messages.TWMExitMenuLoop.

49.3.22 Messages.TWMFontChange Type

Pascal

```
TWMFontChange = TWMNoParams;
```

C++

```
TWMNoParams TWMFontChange;
```

File

Messages

Description

This is type Messages.TWMFontChange.

49.3.23 Messages.TWMGetDlgCode Type

Pascal

```
TWMGetDlgCode = TWMNoParams;
```

C++

```
TWMNoParams TWMGetDlgCode;
```

File

Messages

Description

This is type Messages.TWMGetDlgCode.

49.3.24 Messages.TWMGetFont Type

Pascal

```
TWMGetFont = TWMNoParams;
```

C++

```
TWMNoParams TWMGetFont;
```

File

Messages

Description

This is type Messages.TWMGetFont.

49.3.25 Messages.TWMGetHotKey Type

Pascal

```
TWMGetHotKey = TWMNoParams;
```

C++

```
TWMNoParams TWMGetHotKey;
```

File

Messages

Description

This is type Messages.TWMGetHotKey.

49.3.26 Messages.TWMGetTextLength Type

Pascal

```
TWMGetTextLength = TWMNoParams;
```

C++

```
TWMNoParams TWMGetTextLength;
```

File

Messages

Description

This is type Messages.TWMGetTextLength.

49.3.27 Messages.TWMHScroll Type

Pascal

```
TWMHScroll = TWMScroll;
```

C++

```
TWMScroll TWMHScroll;
```

File

Messages

Description

This is type Messages.TWMHScroll.

49.3.28 Messages.TWMIconEraseBkgnd Type

Pascal

```
TWMIconEraseBkgnd = TWMEraseBkgnd;
```

C++

```
TWMEraseBkgnd TWMIconEraseBkgnd;
```

File

Messages

Description

This is type Messages.TWMIconEraseBkgnd.

49.3.29 Messages.TWMKeyDown Type

Pascal

```
TWMKeyDown = TWMKey;
```

C++

```
TWMKey TWMKeyDown;
```

File

Messages

Description

This is type Messages.TWMKeyDown.

49.3.30 Messages.TWMKeyUp Type

Pascal

```
TWMKeyUp = TWMKey;
```

C++

```
TWMKey TWMKeyUp;
```

File

Messages

Description

This is type Messages.TWMKeyUp.

49.3.31 Messages.TWMLButtonDblClk Type

Pascal

```
TWMLButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMLButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMLButtonDblClk.

49.3.32 Messages.TWMLButtonDown Type

TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.

Pascal

```
TWMLButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMLButtonDown;
```

File

Messages

Description

TWMLButtonDown is the data structure that CLX uses for Windows messages informing an application that the user just clicked the left mouse button. It is the same as all mouse messages.

See Also

TMessage

TWMMouse

TWMPaint

TWMMenuChar

TWMNotify

TWMKey

49.3.33 Messages.TWMLButtonUp Type

Pascal

```
TWMLButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMLButtonUp;
```

File

Messages

Description

This is type Messages.TWMLButtonUp.

49.3.34 Messages.TWMMButtonDblClk Type

Pascal

```
TWMMButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMMButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMMButtonDblClk.

49.3.35 Messages.TWMMButtonDown Type

Pascal

```
TWMMButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMMButtonDown;
```

File

Messages

Description

This is type Messages.TWMMButtonDown.

49.3.36 Messages.TWMMButtonUp Type

Pascal

```
TWMMButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMMButtonUp;
```

File

Messages

Description

This is type Messages.TWMMButtonUp.

49.3.37 Messages.TWMMDIGetActive Type

Pascal

```
TWMMDIGetActive = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIGetActive;
```

File

Messages

Description

This is type Messages.TWMMDIGetActive.

49.3.38 Messages.TWMMDIIconArrange Type

Pascal

```
TWMMDIIconArrange = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIIconArrange;
```

File

Messages

Description

This is type Messages.TWMMDIIconArrange.

49.3.39 Messages.TWMMDIRefreshMenu Type

Pascal

```
TWMMDIRefreshMenu = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIRefreshMenu;
```

File

Messages

Description

This is type Messages.TWMMDIRefreshMenu.

49.3.40 Messages.TWMMouseMove Type

Pascal

```
TWMMouseMove = TWMMouse;
```

C++

```
TWMMouse TWMMouseMove;
```

File

Messages

Description

This is type Messages.TWMMouseMove.

49.3.41 Messages.TWMNCDestroy Type

Pascal

```
TWMNCDestroy = TWMNoParams;
```

C++

```
TWMNoParams TWMNCDestroy;
```

File

Messages

Description

This is type Messages.TWMNCDestroy.

49.3.42 Messages.TWMNCLButtonDblClk Type

Pascal

```
TWMNCLButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCLButtonDblClk.

49.3.43 Messages.TWMNCLButtonDown Type

Pascal

```
TWMNCLButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCLButtonDown.

49.3.44 Messages.TWMNCLButtonUp Type

Pascal

```
TWMNCLButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCLButtonUp.

49.3.45 Messages.TWMNCMButtonDblClk Type

Pascal

```
TWMNCMButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCMButtonDblClk.

49.3.46 Messages.TWMNCMButtonDown Type

Pascal

```
TWMNCMButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCMButtonDown.

49.3.47 Messages.TWMNCMButtonUp Type

Pascal

```
TWMNCMButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCMButtonUp.

49.3.48 Messages.TWMNCMouseMove Type

Pascal

```
TWMNCMouseMove = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMouseMove;
```

File

Messages

Description

This is type Messages.TWMNCMouseMove.

49.3.49 Messages.TWMNCRButtonDblClk Type

Pascal

```
TWMNCRButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCRButtonDblClk.

49.3.50 Messages.TWMNCRButtonDown Type

Pascal

```
TWMNCRButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCRButtonDown.

49.3.51 Messages.TWMNCRButtonUp Type

Pascal

```
TWMNCRButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCRButtonUp.

49.3.52 Messages.TWMPaintIcon Type

Pascal

```
TWMPaintIcon = TWMNoParams;
```

C++

```
TWMNoParams TWMPaintIcon;
```

File

Messages

Description

This is type Messages.TWMPaintIcon.

49.3.53 Messages.TWMPaste Type

Pascal

```
TWMPaste = TWMNoParams;
```

C++

```
TWMNoParams TWMPaste;
```

File

Messages

Description

This is type Messages.TWMPaste.

49.3.54 Messages.TWMPrintClient Type

Pascal

```
TWMPrintClient = TWMPrint;
```

C++

```
TWMPrint TWMPrintClient;
```

File

Messages

Description

This is type Messages.TWMPrintClient.

49.3.55 Messages.TWMQueryDragIcon Type

Pascal

```
TWMQueryDragIcon = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryDragIcon;
```

File

Messages

Description

This is type Messages.TWMQueryDragIcon.

49.3.56 Messages.TWMQueryNewPalette Type

Pascal

```
TWMQueryNewPalette = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryNewPalette;
```

File

Messages

Description

This is type Messages.TWMQueryNewPalette.

49.3.57 Messages.TWMQueryOpen Type

Pascal

```
TWMQueryOpen = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryOpen;
```

File

Messages

Description

This is type Messages.TWMQueryOpen.

49.3.58 Messages.TWMQueryUIState Type

Pascal

```
TWMQueryUIState = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryUIState;
```

File

Messages

Description

This is type Messages.TWMQueryUIState.

49.3.59 Messages.TWMQueueSync Type

Pascal

```
TWMQueueSync = TWMNoParams;
```

C++

```
TWMNoParams TWMQueueSync;
```

File

Messages

Description

This is type Messages.TWMQueueSync.

49.3.60 Messages.TWMRButtonDblClk Type

Pascal

```
TWMRButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMRButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMRButtonDblClk.

49.3.61 Messages.TWMRButtonDown Type

Pascal

```
TWMRButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMRButtonDown;
```

File

Messages

Description

This is type Messages.TWMRButtonDown.

49.3.62 Messages.TWMRButtonUp Type

Pascal

```
TWMRButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMRButtonUp;
```

File

Messages

Description

This is type Messages.TWMRButtonUp.

49.3.63 Messages.TWMRenderAllFormats Type

Pascal

```
TWMRenderAllFormats = TWMNoParams;
```

C++

```
TWMNoParams TWMRenderAllFormats;
```

File

Messages

Description

This is type Messages.TWMRenderAllFormats.

49.3.64 Messages.TWMStyleChanged Type

Pascal

```
TWMStyleChanged = TWMStyleChange;
```

C++

```
TWMStyleChange TWMStyleChanged;
```

File

Messages

Description

This is type Messages.TWMStyleChanged.

49.3.65 Messages.TWMStyleChanging Type

Pascal

```
TWMStyleChanging = TWMStyleChange;
```

C++

```
TWMStyleChange TWMStyleChanging;
```

File

Messages

Description

This is type Messages.TWMStyleChanging.

49.3.66 Messages.TWMSysChar Type

Pascal

```
TWMSysChar = TWMKey;
```

C++

```
TWMKey TWMSysChar;
```

File

Messages

Description

This is type Messages.TWMSysChar.

49.3.67 Messages.TWMSysColorChange Type

Pascal

```
TWMSysColorChange = TWMNoParams;
```

C++

```
TWMNoParams TWMSysColorChange;
```

File

Messages

Description

This is type Messages.TWMSysColorChange.

49.3.68 Messages.TWMSysKeyDown Type

Pascal

```
TWMSysKeyDown = TWMKey;
```

C++

```
TWMKey TWMSysKeyDown;
```

File

Messages

Description

This is type Messages.TWMSysKeyDown.

49.3.69 Messages.TWMSysKeyUp Type

Pascal

```
TWMSysKeyUp = TWMKey;
```

C++

```
TWMKey TWMSysKeyUp;
```

File

Messages

Description

This is type Messages.TWMSysKeyUp.

49.3.70 Messages.TWMTimeChange Type

Pascal

```
TWMTimeChange = TWMNoParams;
```

C++

```
TWMNoParams TWMTimeChange;
```

File

Messages

Description

This is type Messages.TWMTimeChange.

49.3.71 Messages.TWMUndo Type

Pascal

```
TWMUndo = TWMNoParams;
```

C++

```
TWMNoParams TWMUndo;
```

File

Messages

Description

This is type Messages.TWMUndo.

49.3.72 Messages.TWMUpdateUIState Type

Pascal

```
TWMUpdateUIState = TWMUIState;
```

C++

```
TWMUIState TWMUpdateUIState;
```

File

Messages

Description

This is type Messages.TWMUpdateUIState.

49.3.73 Messages.TWMVKeyToItem Type

Pascal

```
TWMVKeyToItem = TWMCharToItem;
```

C++

```
TWMCharToItem TWMVKeyToItem;
```

File

Messages

Description

This is type Messages.TWMVKeyToItem.

49.3.74 Messages.TWMVScroll Type

Pascal

```
TWMVScroll = TWMScroll;
```

C++

```
TWMScroll TWMVScroll;
```

File

Messages

Description

This is type Messages.TWMVScroll.

49.3.75 Messages.TWMWindowPosChanged Type

Pascal

```
TWMWindowPosChanged = TWMWindowPosMsg;
```

C++

```
TWMWindowPosMsg TWMWindowPosChanged;
```

File

Messages

Description

This is type Messages.TWMWindowPosChanged.

49.3.76 Messages.TWMWindowPosChanging Type

Pascal

```
TWMWindowPosChanging = TWMWindowPosMsg;
```

C++

```
TWMWindowPosMsg TWMWindowPosChanging;
```

File

Messages

Description

This is type Messages.TWMWindowPosChanging.

49.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BM_CLICK (see page 5084)	This is constant Messages.BM_CLICK.
BM_GETCHECK (see page 5085)	This is constant Messages.BM_GETCHECK.
BM_GETIMAGE (see page 5085)	This is constant Messages.BM_GETIMAGE.
BM_GETSTATE (see page 5085)	This is constant Messages.BM_GETSTATE.
BM_SETCHECK (see page 5086)	This is constant Messages.BM_SETCHECK.
BM_SETIMAGE (see page 5086)	This is constant Messages.BM_SETIMAGE.
BM_SETSTATE (see page 5086)	This is constant Messages.BM_SETSTATE.
BM_SETSTYLE (see page 5087)	This is constant Messages.BM_SETSTYLE.
BN_CLICKED (see page 5087)	This is constant Messages.BN_CLICKED.
BN_DBLCLK (see page 5087)	This is constant Messages.BN_DBLCLK.
BN_DISABLE (see page 5088)	This is constant Messages.BN_DISABLE.
BN_DOUBLECLICKED (see page 5088)	This is constant Messages.BN_DOUBLECLICKED.
BN_HILITE (see page 5088)	This is constant Messages.BN_HILITE.
BN_KILLFOCUS (see page 5089)	This is constant Messages.BN_KILLFOCUS.
BN_PAINT (see page 5089)	This is constant Messages.BN_PAINT.
BN_PUSHED (see page 5089)	This is constant Messages.BN_PUSHED.
BN_SETFOCUS (see page 5090)	This is constant Messages.BN_SETFOCUS.
BN_UNHILITE (see page 5090)	This is constant Messages.BN_UNHILITE.
BN_UNPUSHED (see page 5090)	This is constant Messages.BN_UNPUSHED.
CB_ADDSTRING (see page 5091)	This is constant Messages.CB_ADDSTRING.
CB_DELETESTRING (see page 5091)	This is constant Messages.CB_DELETESTRING.
CB_DIR (see page 5091)	This is constant Messages.CB_DIR.
CB_FINDSTRING (see page 5092)	This is constant Messages.CB_FINDSTRING.
CB_FINDSTRINGEXACT (see page 5092)	This is constant Messages.CB_FINDSTRINGEXACT.
CB_GETCOUNT (see page 5092)	This is constant Messages.CB_GETCOUNT.
CB_GETCURSEL (see page 5093)	This is constant Messages.CB_GETCURSEL.
CB_GETDROPPEDCONTROLRECT (see page 5093)	This is constant Messages.CB_GETDROPPEDCONTROLRECT.
CB_GETDROPPEDSTATE (see page 5093)	This is constant Messages.CB_GETDROPPEDSTATE.
CB_GETDROPPEDWIDTH (see page 5094)	This is constant Messages.CB_GETDROPPEDWIDTH.
CB_GETEDITSEL (see page 5094)	This is constant Messages.CB_GETEDITSEL.
CB_GETEXTENDEDUI (see page 5094)	This is constant Messages.CB_GETEXTENDEDUI.
CB_GETHORIZONTALEXTENT (see page 5095)	This is constant Messages.CB_GETHORIZONTALEXTENT.
CB_GETITEMDATA (see page 5095)	This is constant Messages.CB_GETITEMDATA.
CB_GETITEMHEIGHT (see page 5095)	This is constant Messages.CB_GETITEMHEIGHT.
CB_GETLBTEXT (see page 5096)	This is constant Messages.CB_GETLBTEXT.
CB_GETLBTEXTLEN (see page 5096)	This is constant Messages.CB_GETLBTEXTLEN.
CB_GETLOCALE (see page 5096)	This is constant Messages.CB_GETLOCALE.
CB_GETTOPINDEX (see page 5097)	This is constant Messages.CB_GETTOPINDEX.
CB_INITSTORAGE (see page 5097)	This is constant Messages.CB_INITSTORAGE.
CB_INSERTSTRING (see page 5097)	This is constant Messages.CB_INSERTSTRING.
CB_LIMITTEXT (see page 5098)	This is constant Messages.CB_LIMITTEXT.
CB_MSGMAX (see page 5098)	This is constant Messages.CB_MSGMAX.
CB_RESETCONTENT (see page 5098)	This is constant Messages.CB_RESETCONTENT.
CB_SELECTSTRING (see page 5099)	This is constant Messages.CB_SELECTSTRING.

CB_SETCURSEL (see page 5099)	This is constant Messages.CB_SETCURSEL.
CB_SETDROPPEDWIDTH (see page 5099)	This is constant Messages.CB_SETDROPPEDWIDTH.
CB_SETEDITSEL (see page 5100)	This is constant Messages.CB_SETEDITSEL.
CB_SETEXTENDEDUI (see page 5100)	This is constant Messages.CB_SETEXTENDEDUI.
CB_SETHORIZONTALEXTENT (see page 5100)	This is constant Messages.CB_SETHORIZONTALEXTENT.
CB_SETITEMDATA (see page 5101)	This is constant Messages.CB_SETITEMDATA.
CB_SETITEMHEIGHT (see page 5101)	This is constant Messages.CB_SETITEMHEIGHT.
CB_SETLOCALE (see page 5101)	This is constant Messages.CB_SETLOCALE.
CB_SETTOPINDEX (see page 5102)	This is constant Messages.CB_SETTOPINDEX.
CB_SHOWDROPDOWN (see page 5102)	This is constant Messages.CB_SHOWDROPDOWN.
CBN_CLOSEUP (see page 5102)	This is constant Messages.CBN_CLOSEUP.
CBN_DBLCLK (see page 5103)	This is constant Messages.CBN_DBLCLK.
CBN_DROPDOWN (see page 5103)	This is constant Messages.CBN_DROPDOWN.
CBN_EDITCHANGE (see page 5103)	This is constant Messages.CBN_EDITCHANGE.
CBN_EDITUPDATE (see page 5104)	This is constant Messages.CBN_EDITUPDATE.
CBN_ERRSPACE (see page 5104)	This is constant Messages.CBN_ERRSPACE.
CBN_KILLFOCUS (see page 5104)	This is constant Messages.CBN_KILLFOCUS.
CBN_SELCHANGE (see page 5105)	This is constant Messages.CBN_SELCHANGE.
CBN_SELENDCANCEL (see page 5105)	This is constant Messages.CBN_SELENDCANCEL.
CBN_SELENDOK (see page 5105)	This is constant Messages.CBN_SELENDOK.
CBN_SETFOCUS (see page 5106)	This is constant Messages.CBN_SETFOCUS.
DM_GETDEFID (see page 5106)	This is constant Messages.DM_GETDEFID.
DM_REPOSITION (see page 5106)	This is constant Messages.DM_REPOSITION.
DM_SETDEFID (see page 5107)	This is constant Messages.DM_SETDEFID.
EM_CANUNDO (see page 5107)	This is constant Messages.EM_CANUNDO.
EM_CHARFROMPOS (see page 5107)	This is constant Messages.EM_CHARFROMPOS.
EM_EMPTYUNDOBUFFER (see page 5108)	This is constant Messages.EM_EMPTYUNDOBUFFER.
EM_FMTLINES (see page 5108)	This is constant Messages.EM_FMTLINES.
EM_GETFIRSTVISIBLELINE (see page 5108)	This is constant Messages.EM_GETFIRSTVISIBLELINE.
EM_GETHANDLE (see page 5109)	This is constant Messages.EM_GETHANDLE.
EM_GETIMESTATUS (see page 5109)	This is constant Messages.EM_GETIMESTATUS.
EM_GETLIMITTEXT (see page 5109)	This is constant Messages.EM_GETLIMITTEXT.
EM_GETLINE (see page 5110)	This is constant Messages.EM_GETLINE.
EM_GETLINECOUNT (see page 5110)	This is constant Messages.EM_GETLINECOUNT.
EM_GETMARGINS (see page 5110)	This is constant Messages.EM_GETMARGINS.
EM_GETMODIFY (see page 5111)	This is constant Messages.EM_GETMODIFY.
EM_GETPASSWORDCHAR (see page 5111)	This is constant Messages.EM_GETPASSWORDCHAR.
EM_GETRECT (see page 5111)	This is constant Messages.EM_GETRECT.
EM_GETSEL (see page 5112)	This is constant Messages.EM_GETSEL.
EM_GETTHUMB (see page 5112)	This is constant Messages.EM_GETTHUMB.
EM_GETWORDBREAKPROC (see page 5112)	This is constant Messages.EM_GETWORDBREAKPROC.
EM_LIMITTEXT (see page 5113)	This is constant Messages.EM_LIMITTEXT.
EM_LINEFROMCHAR (see page 5113)	This is constant Messages.EM_LINEFROMCHAR.
EM_LINEINDEX (see page 5113)	This is constant Messages.EM_LINEINDEX.
EM_LINELENGTH (see page 5114)	This is constant Messages.EM_LINELENGTH.
EM_LINESCROLL (see page 5114)	This is constant Messages.EM_LINESCROLL.
EM_POSFROMCHAR (see page 5114)	This is constant Messages.EM_POSFROMCHAR.
EM_REPLACESEL (see page 5115)	This is constant Messages.EM_REPLACESEL.
EM_SCROLL (see page 5115)	This is constant Messages.EM_SCROLL.
EM_SCROLLCARET (see page 5115)	This is constant Messages.EM_SCROLLCARET.
EM_SETHANDLE (see page 5116)	This is constant Messages.EM_SETHANDLE.
EM_SETIMESTATUS (see page 5116)	This is constant Messages.EM_SETIMESTATUS.
EM_SETLIMITTEXT (see page 5116)	This is constant Messages.EM_SETLIMITTEXT.
EM_SETMARGINS (see page 5117)	This is constant Messages.EM_SETMARGINS.

EM_SETMODIFY (see page 5117)	This is constant Messages.EM_SETMODIFY.
EM_SETPASSWORDCHAR (see page 5117)	This is constant Messages.EM_SETPASSWORDCHAR.
EM_SETREADONLY (see page 5118)	This is constant Messages.EM_SETREADONLY.
EM_SETRECT (see page 5118)	This is constant Messages.EM_SETRECT.
EM_SETRECTNP (see page 5118)	This is constant Messages.EM_SETRECTNP.
EM_SETSEL (see page 5119)	This is constant Messages.EM_SETSEL.
EM_SETTABSTOPS (see page 5119)	This is constant Messages.EM_SETTABSTOPS.
EM_SETWORDBREAKPROC (see page 5119)	This is constant Messages.EM_SETWORDBREAKPROC.
EM_UNDO (see page 5120)	This is constant Messages.EM_UNDO.
EN_CHANGE (see page 5120)	This is constant Messages.EN_CHANGE.
EN_ERRSPACE (see page 5120)	This is constant Messages.EN_ERRSPACE.
EN_HSCROLL (see page 5121)	This is constant Messages.EN_HSCROLL.
EN_KILLFOCUS (see page 5121)	This is constant Messages.EN_KILLFOCUS.
EN_MAXTEXT (see page 5121)	This is constant Messages.EN_MAXTEXT.
EN_SETFOCUS (see page 5122)	This is constant Messages.EN_SETFOCUS.
EN_UPDATE (see page 5122)	This is constant Messages.EN_UPDATE.
EN_VSCROLL (see page 5122)	This is constant Messages.EN_VSCROLL.
LB_ADDFILE (see page 5123)	This is constant Messages.LB_ADDFILE.
LB_ADDSTRING (see page 5123)	This is constant Messages.LB_ADDSTRING.
LB_DELETESTRING (see page 5123)	This is constant Messages.LB_DELETESTRING.
LB_DIR (see page 5124)	This is constant Messages.LB_DIR.
LB_FINDSTRING (see page 5124)	This is constant Messages.LB_FINDSTRING.
LB_FINDSTRINGEXACT (see page 5124)	This is constant Messages.LB_FINDSTRINGEXACT.
LB_GETANCHORINDEX (see page 5125)	This is constant Messages.LB_GETANCHORINDEX.
LB_GETCARETINDEX (see page 5125)	This is constant Messages.LB_GETCARETINDEX.
LB_GETCOUNT (see page 5125)	This is constant Messages.LB_GETCOUNT.
LB_GETCURSEL (see page 5126)	This is constant Messages.LB_GETCURSEL.
LB_GETHORIZONTALEXTENT (see page 5126)	This is constant Messages.LB_GETHORIZONTALEXTENT.
LB_GETITEMDATA (see page 5126)	This is constant Messages.LB_GETITEMDATA.
LB_GETITEMHEIGHT (see page 5127)	This is constant Messages.LB_GETITEMHEIGHT.
LB_GETITEMRECT (see page 5127)	This is constant Messages.LB_GETITEMRECT.
LB_GETLOCALE (see page 5127)	This is constant Messages.LB_GETLOCALE.
LB_GETSEL (see page 5128)	This is constant Messages.LB_GETSEL.
LB_GETSELCOUNT (see page 5128)	This is constant Messages.LB_GETSELCOUNT.
LB_GETSELITEMS (see page 5128)	This is constant Messages.LB_GETSELITEMS.
LB_GETTEXT (see page 5129)	This is constant Messages.LB_GETTEXT.
LB_GETTEXTLEN (see page 5129)	This is constant Messages.LB_GETTEXTLEN.
LB_GETTOPINDEX (see page 5129)	This is constant Messages.LB_GETTOPINDEX.
LB_INITSTORAGE (see page 5130)	This is constant Messages.LB_INITSTORAGE.
LB_INSERTSTRING (see page 5130)	This is constant Messages.LB_INSERTSTRING.
LB_ITEMFROMPOINT (see page 5130)	This is constant Messages.LB_ITEMFROMPOINT.
LB_MSGMAX (see page 5131)	This is constant Messages.LB_MSGMAX.
LB_RESETCONTENT (see page 5131)	This is constant Messages.LB_RESETCONTENT.
LB_SELECTSTRING (see page 5131)	This is constant Messages.LB_SELECTSTRING.
LB_SELITEMRANGE (see page 5132)	This is constant Messages.LB_SELITEMRANGE.
LB_SELITEMRANGEEX (see page 5132)	This is constant Messages.LB_SELITEMRANGEEX.
LB_SETANCHORINDEX (see page 5132)	This is constant Messages.LB_SETANCHORINDEX.
LB_SETCARETINDEX (see page 5133)	This is constant Messages.LB_SETCARETINDEX.
LB_SETCOLUMNWIDTH (see page 5133)	This is constant Messages.LB_SETCOLUMNWIDTH.
LB_SETCOUNT (see page 5133)	This is constant Messages.LB_SETCOUNT.
LB_SETCURSEL (see page 5134)	This is constant Messages.LB_SETCURSEL.
LB_SETHORIZONTALEXTENT (see page 5134)	This is constant Messages.LB_SETHORIZONTALEXTENT.
LB_SETITEMDATA (see page 5134)	This is constant Messages.LB_SETITEMDATA.
LB_SETITEMHEIGHT (see page 5135)	This is constant Messages.LB_SETITEMHEIGHT.

LB_SETLOCALE (see page 5135)	This is constant Messages.LB_SETLOCALE.
LB_SETSEL (see page 5135)	This is constant Messages.LB_SETSEL.
LB_SETTABSTOPS (see page 5136)	This is constant Messages.LB_SETTABSTOPS.
LB_SETTOPINDEX (see page 5136)	This is constant Messages.LB_SETTOPINDEX.
LBN_DBLCLK (see page 5136)	This is constant Messages.LBN_DBLCLK.
LBN_ERRSPACE (see page 5137)	This is constant Messages.LBN_ERRSPACE.
LBN_KILLFOCUS (see page 5137)	This is constant Messages.LBN_KILLFOCUS.
LBN_SELCANCEL (see page 5137)	This is constant Messages.LBN_SELCANCEL.
LBN_SELCHANGE (see page 5138)	This is constant Messages.LBN_SELCHANGE.
LBN_SETFOCUS (see page 5138)	This is constant Messages.LBN_SETFOCUS.
PSM_PAGEINFO (see page 5138)	This is constant Messages.PSM_PAGEINFO.
PSM_SHEETINFO (see page 5139)	This is constant Messages.PSM_SHEETINFO.
SBM_ENABLE_ARROWS (see page 5139)	This is constant Messages.SBM_ENABLE_ARROWS.
SBM_GETPOS (see page 5139)	This is constant Messages.SBM_GETPOS.
SBM_GETRANGE (see page 5140)	This is constant Messages.SBM_GETRANGE.
SBM_GETSCROLLBARINFO (see page 5140)	This is constant Messages.SBM_GETSCROLLBARINFO.
SBM_GETSCROLLINFO (see page 5140)	This is constant Messages.SBM_GETSCROLLINFO.
SBM_SETPOS (see page 5141)	This is constant Messages.SBM_SETPOS.
SBM_SETRANGE (see page 5141)	This is constant Messages.SBM_SETRANGE.
SBM_SETRANGEREDRAW (see page 5141)	This is constant Messages.SBM_SETRANGEREDRAW.
SBM_SETSCROLLINFO (see page 5142)	This is constant Messages.SBM_SETSCROLLINFO.
WM_ACTIVATE (see page 5142)	This is constant Messages.WM_ACTIVATE.
WM_ACTIVATEAPP (see page 5142)	This is constant Messages.WM_ACTIVATEAPP.
WM_APP (see page 5143)	This is constant Messages.WM_APP.
WM_APPCOMMAND (see page 5143)	This is constant Messages.WM_APPCOMMAND.
WM_ASKCBFORMATNAME (see page 5143)	This is constant Messages.WM_ASKCBFORMATNAME.
WM_CANCELJOURNAL (see page 5144)	This is constant Messages.WM_CANCELJOURNAL.
WM_CANCELMODE (see page 5144)	This is constant Messages.WM_CANCELMODE.
WM_CAPTURECHANGED (see page 5144)	This is constant Messages.WM_CAPTURECHANGED.
WM_CHANGECBCHAIN (see page 5145)	This is constant Messages.WM_CHANGECBCHAIN.
WM_CHANGEUISTATE (see page 5145)	This is constant Messages.WM_CHANGEUISTATE.
WM_CHAR (see page 5145)	This is constant Messages.WM_CHAR.
WM_CHARTOITEM (see page 5146)	This is constant Messages.WM_CHARTOITEM.
WM_CHILDACTIVATE (see page 5146)	This is constant Messages.WM_CHILDACTIVATE.
WM_CLEAR (see page 5146)	This is constant Messages.WM_CLEAR.
WM_CLOSE (see page 5147)	This is constant Messages.WM_CLOSE.
WM_COALESCE_FIRST (see page 5147)	This is constant Messages.WM_COALESCE_FIRST.
WM_COALESCE_LAST (see page 5147)	This is constant Messages.WM_COALESCE_LAST.
WM_COMMAND (see page 5148)	This is constant Messages.WM_COMMAND.
WM_COMMNOTIFY (see page 5148)	This is constant Messages.WM_COMMNOTIFY.
WM_COMPACTING (see page 5148)	This is constant Messages.WM_COMPACTING.
WM_COMPAREITEM (see page 5149)	This is constant Messages.WM_COMPAREITEM.
WM_CONTEXTMENU (see page 5149)	This is constant Messages.WM_CONTEXTMENU.
WM_COPY (see page 5149)	This is constant Messages.WM_COPY.
WM_COPYDATA (see page 5150)	This is constant Messages.WM_COPYDATA.
WM_CREATE (see page 5150)	This is constant Messages.WM_CREATE.
WM_CTLCOLOR (see page 5150)	This is constant Messages.WM_CTLCOLOR.
WM_CTLCOLORBTN (see page 5151)	This is constant Messages.WM_CTLCOLORBTN.
WM_CTLCOLORDLG (see page 5151)	This is constant Messages.WM_CTLCOLORDLG.
WM_CTLCOLOREDIT (see page 5151)	This is constant Messages.WM_CTLCOLOREDIT.
WM_CTLCOLORLISTBOX (see page 5152)	This is constant Messages.WM_CTLCOLORLISTBOX.
WM_CTLCOLORMSGBOX (see page 5152)	This is constant Messages.WM_CTLCOLORMSGBOX.
WM_CTLCOLORSCROLLBAR (see page 5152)	This is constant Messages.WM_CTLCOLORSCROLLBAR.
WM_CTLCOLORSTATIC (see page 5153)	This is constant Messages.WM_CTLCOLORSTATIC.

WM_CUT (see page 5153)	This is constant Messages.WM_CUT.
WM_DDE_ACK (see page 5153)	This is constant Messages.WM_DDE_ACK.
WM_DDE_ADVISE (see page 5154)	This is constant Messages.WM_DDE_ADVISE.
WM_DDE_DATA (see page 5154)	This is constant Messages.WM_DDE_DATA.
WM_DDE_EXECUTE (see page 5154)	This is constant Messages.WM_DDE_EXECUTE.
WM_DDE_FIRST (see page 5155)	This is constant Messages.WM_DDE_FIRST.
WM_DDE_INITIATE (see page 5155)	This is constant Messages.WM_DDE_INITIATE.
WM_DDE_LAST (see page 5155)	This is constant Messages.WM_DDE_LAST.
WM_DDE_POKE (see page 5156)	This is constant Messages.WM_DDE_POKE.
WM_DDE_REQUEST (see page 5156)	This is constant Messages.WM_DDE_REQUEST.
WM_DDE_TERMINATE (see page 5156)	This is constant Messages.WM_DDE_TERMINATE.
WM_DDE_UNADVISE (see page 5157)	This is constant Messages.WM_DDE_UNADVISE.
WM_DEADCHAR (see page 5157)	This is constant Messages.WM_DEADCHAR.
WM_DELETEITEM (see page 5157)	This is constant Messages.WM_DELETEITEM.
WM_DESTROY (see page 5158)	This is constant Messages.WM_DESTROY.
WM_DESTROYCLIPBOARD (see page 5158)	This is constant Messages.WM_DESTROYCLIPBOARD.
WM_DEVICECHANGE (see page 5158)	This is constant Messages.WM_DEVICECHANGE.
WM_DEVMODECHANGE (see page 5159)	This is constant Messages.WM_DEVMODECHANGE.
WM_DISPLAYCHANGE (see page 5159)	This is constant Messages.WM_DISPLAYCHANGE.
WM_DRAWCLIPBOARD (see page 5159)	This is constant Messages.WM_DRAWCLIPBOARD.
WM_DRAWITEM (see page 5160)	This is constant Messages.WM_DRAWITEM.
WM_DROPFILES (see page 5160)	This is constant Messages.WM_DROPFILES.
WM_DWMCOLORIZATIONCOLORCHANGED (see page 5160)	This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.
WM_DWMCOMPOSITIONCHANGED (see page 5161)	This is constant Messages.WM_DWMCOMPOSITIONCHANGED.
WM_DWMNCRENDERINGCHANGED (see page 5161)	This is constant Messages.WM_DWMNCRENDERINGCHANGED.
WM_DWMWINDOWMAXIMIZEDCHANGE (see page 5161)	This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.
WM_ENABLE (see page 5162)	This is constant Messages.WM_ENABLE.
WM_ENDSESSION (see page 5162)	This is constant Messages.WM_ENDSESSION.
WM_ENTERIDLE (see page 5162)	This is constant Messages.WM_ENTERIDLE.
WM_ENTERMENULOOP (see page 5163)	This is constant Messages.WM_ENTERMENULOOP.
WM_ENTERSIZEMOVE (see page 5163)	This is constant Messages.WM_ENTERSIZEMOVE.
WM_ERASEBKGND (see page 5163)	This is constant Messages.WM_ERASEBKGND.
WM_EXITMENULOOP (see page 5164)	This is constant Messages.WM_EXITMENULOOP.
WM_EXITSIZEMOVE (see page 5164)	This is constant Messages.WM_EXITSIZEMOVE.
WM_FONTCHANGE (see page 5164)	This is constant Messages.WM_FONTCHANGE.
WM_GETDLGCODE (see page 5165)	This is constant Messages.WM_GETDLGCODE.
WM_GETFONT (see page 5165)	This is constant Messages.WM_GETFONT.
WM_GETHOTKEY (see page 5165)	This is constant Messages.WM_GETHOTKEY.
WM_GETICON (see page 5166)	This is constant Messages.WM_GETICON.
WM_GETMINMAXINFO (see page 5166)	This is constant Messages.WM_GETMINMAXINFO.
WM_GETOBJECT (see page 5166)	This is constant Messages.WM_GETOBJECT.
WM_GETTEXT (see page 5167)	This is constant Messages.WM_GETTEXT.
WM_GETTEXTLENGTH (see page 5167)	This is constant Messages.WM_GETTEXTLENGTH.
WM_HANDHELDFIRST (see page 5167)	This is constant Messages.WM_HANDHELDFIRST.
WM_HANDHELDLAST (see page 5168)	This is constant Messages.WM_HANDHELDLAST.
WM_HELP (see page 5168)	This is constant Messages.WM_HELP.
WM_HOTKEY (see page 5168)	This is constant Messages.WM_HOTKEY.
WM_HSCROLL (see page 5169)	This is constant Messages.WM_HSCROLL.
WM_HSCROLLCLIPBOARD (see page 5169)	This is constant Messages.WM_HSCROLLCLIPBOARD.
WM_ICONERASEBKGND (see page 5169)	This is constant Messages.WM_ICONERASEBKGND.
WM_IME_CHAR (see page 5170)	This is constant Messages.WM_IME_CHAR.
WM_IME_COMPOSITION (see page 5170)	This is constant Messages.WM_IME_COMPOSITION.
WM_IME_COMPOSITIONFULL (see page 5170)	This is constant Messages.WM_IME_COMPOSITIONFULL.
WM_IME_CONTROL (see page 5171)	This is constant Messages.WM_IME_CONTROL.

WM_IME_ENDCOMPOSITION (see page 5171)	This is constant Messages.WM_IME_ENDCOMPOSITION.
WM_IME_KEYDOWN (see page 5171)	This is constant Messages.WM_IME_KEYDOWN.
WM_IME_KEYLAST (see page 5172)	This is constant Messages.WM_IME_KEYLAST.
WM_IME_KEYUP (see page 5172)	This is constant Messages.WM_IME_KEYUP.
WM_IME_NOTIFY (see page 5172)	This is constant Messages.WM_IME_NOTIFY.
WM_IME_REQUEST (see page 5173)	This is constant Messages.WM_IME_REQUEST.
WM_IME_SELECT (see page 5173)	This is constant Messages.WM_IME_SELECT.
WM_IME_SETCONTEXT (see page 5173)	This is constant Messages.WM_IME_SETCONTEXT.
WM_IME_STARTCOMPOSITION (see page 5174)	This is constant Messages.WM_IME_STARTCOMPOSITION.
WM_INITDIALOG (see page 5174)	This is constant Messages.WM_INITDIALOG.
WM_INITMENU (see page 5174)	This is constant Messages.WM_INITMENU.
WM_INITMENUPOPUP (see page 5175)	This is constant Messages.WM_INITMENUPOPUP.
WM_INPUT (see page 5175)	This is constant Messages.WM_INPUT.
WM_INPUTLANGCHANGE (see page 5175)	This is constant Messages.WM_INPUTLANGCHANGE.
WM_INPUTLANGCHANGEREQUEST (see page 5176)	This is constant Messages.WM_INPUTLANGCHANGEREQUEST.
WM_KEYDOWN (see page 5176)	This is constant Messages.WM_KEYDOWN.
WM_KEYFIRST (see page 5176)	This is constant Messages.WM_KEYFIRST.
WM_KEYLAST (see page 5177)	This is constant Messages.WM_KEYLAST.
WM_KEYUP (see page 5177)	This is constant Messages.WM_KEYUP.
WM_KILLFOCUS (see page 5177)	This is constant Messages.WM_KILLFOCUS.
WM_LBUTTONDBLCLK (see page 5178)	This is constant Messages.WM_LBUTTONDBLCLK.
WM_LBUTTONDOWN (see page 5178)	This is constant Messages.WM_LBUTTONDOWN.
WM_LBUTTONUP (see page 5178)	This is constant Messages.WM_LBUTTONUP.
WM_MBUTTONDBLCLK (see page 5179)	This is constant Messages.WM_MBUTTONDBLCLK.
WM_MBUTTONDOWN (see page 5179)	This is constant Messages.WM_MBUTTONDOWN.
WM_MBUTTONUP (see page 5179)	This is constant Messages.WM_MBUTTONUP.
WM_MDIACTIVATE (see page 5180)	This is constant Messages.WM_MDIACTIVATE.
WM_MDICASCADE (see page 5180)	This is constant Messages.WM_MDICASCADE.
WM_MDICREATE (see page 5180)	This is constant Messages.WM_MDICREATE.
WM_MDIDESTROY (see page 5181)	This is constant Messages.WM_MDIDESTROY.
WM_MDIGETACTIVE (see page 5181)	This is constant Messages.WM_MDIGETACTIVE.
WM_MDIICONARRANGE (see page 5181)	This is constant Messages.WM_MDIICONARRANGE.
WM_MDIMAXIMIZE (see page 5182)	This is constant Messages.WM_MDIMAXIMIZE.
WM_MDINEXT (see page 5182)	This is constant Messages.WM_MDINEXT.
WM_MDIREFRESHMENU (see page 5182)	This is constant Messages.WM_MDIREFRESHMENU.
WM_MDIRESTORE (see page 5183)	This is constant Messages.WM_MDIRESTORE.
WM_MDISETMENU (see page 5183)	This is constant Messages.WM_MDISETMENU.
WM_MDITILE (see page 5183)	This is constant Messages.WM_MDITILE.
WM_MEASUREITEM (see page 5184)	This is constant Messages.WM_MEASUREITEM.
WM_MENUCHAR (see page 5184)	This is constant Messages.WM_MENUCHAR.
WM_MENUCOMMAND (see page 5184)	This is constant Messages.WM_MENUCOMMAND.
WM_MENUDRAG (see page 5185)	This is constant Messages.WM_MENUDRAG.
WM_MENUGETOBJECT (see page 5185)	This is constant Messages.WM_MENUGETOBJECT.
WM_MENURBUTTONUP (see page 5185)	This is constant Messages.WM_MENURBUTTONUP.
WM_MENUSELECT (see page 5186)	This is constant Messages.WM_MENUSELECT.
WM_MOUSEACTIVATE (see page 5186)	This is constant Messages.WM_MOUSEACTIVATE.
WM_MOUSEFIRST (see page 5186)	This is constant Messages.WM_MOUSEFIRST.
WM_MOUSEHOVER (see page 5187)	This is constant Messages.WM_MOUSEHOVER.
WM_MOUSELAST (see page 5187)	This is constant Messages.WM_MOUSELAST.
WM_MOUSELEAVE (see page 5187)	This is constant Messages.WM_MOUSELEAVE.
WM_MOUSEMOVE (see page 5188)	This is constant Messages.WM_MOUSEMOVE.
WM_MOUSEWHEEL (see page 5188)	This is constant Messages.WM_MOUSEWHEEL.
WM_MOVE (see page 5188)	This is constant Messages.WM_MOVE.
WM_MOVING (see page 5189)	This is constant Messages.WM_MOVING.

WM_NCACTIVATE (see page 5189)	This is constant Messages.WM_NCACTIVATE.
WM_NCCALCSIZE (see page 5189)	This is constant Messages.WM_NCCALCSIZE.
WM_NCCREATE (see page 5190)	This is constant Messages.WM_NCCREATE.
WM_NCDESTROY (see page 5190)	This is constant Messages.WM_NCDESTROY.
WM_NCHITTEST (see page 5190)	This is constant Messages.WM_NCHITTEST.
WM_NCLBUTTONDBLCLK (see page 5191)	This is constant Messages.WM_NCLBUTTONDBLCLK.
WM_NCLBUTTONDOWN (see page 5191)	This is constant Messages.WM_NCLBUTTONDOWN.
WM_NCLBUTTONUP (see page 5191)	This is constant Messages.WM_NCLBUTTONUP.
WM_NCMBUTTONDBLCLK (see page 5192)	This is constant Messages.WM_NCMBUTTONDBLCLK.
WM_NCMBUTTONDOWN (see page 5192)	This is constant Messages.WM_NCMBUTTONDOWN.
WM_NCMBUTTONUP (see page 5192)	This is constant Messages.WM_NCMBUTTONUP.
WM_NCMOUSEHOVER (see page 5193)	This is constant Messages.WM_NCMOUSEHOVER.
WM_NCMOUSELEAVE (see page 5193)	This is constant Messages.WM_NCMOUSELEAVE.
WM_NCMOUSEMOVE (see page 5193)	This is constant Messages.WM_NCMOUSEMOVE.
WM_NCPAINT (see page 5194)	This is constant Messages.WM_NCPAINT.
WM_NCRBUTTONDBLCLK (see page 5194)	This is constant Messages.WM_NCRBUTTONDBLCLK.
WM_NCRBUTTONDOWN (see page 5194)	This is constant Messages.WM_NCRBUTTONDOWN.
WM_NCRBUTTONUP (see page 5195)	This is constant Messages.WM_NCRBUTTONUP.
WM_NCXBUTTONDBLCLK (see page 5195)	This is constant Messages.WM_NCXBUTTONDBLCLK.
WM_NCXBUTTONDOWN (see page 5195)	This is constant Messages.WM_NCXBUTTONDOWN.
WM_NCXBUTTONUP (see page 5196)	This is constant Messages.WM_NCXBUTTONUP.
WM_NEXTDLGCTL (see page 5196)	This is constant Messages.WM_NEXTDLGCTL.
WM_NEXTMENU (see page 5196)	This is constant Messages.WM_NEXTMENU.
WM_NOTIFY (see page 5197)	This is constant Messages.WM_NOTIFY.
WM_NOTIFYFORMAT (see page 5197)	This is constant Messages.WM_NOTIFYFORMAT.
WM_NULL (see page 5197)	This is constant Messages.WM_NULL.
WM_PAINT (see page 5198)	This is constant Messages.WM_PAINT.
WM_PAINTCLIPBOARD (see page 5198)	This is constant Messages.WM_PAINTCLIPBOARD.
WM_PAINTICON (see page 5198)	This is constant Messages.WM_PAINTICON.
WM_PALETTECHANGED (see page 5199)	This is constant Messages.WM_PALETTECHANGED.
WM_PALETTEISCHANGING (see page 5199)	This is constant Messages.WM_PALETTEISCHANGING.
WM_PARENTNOTIFY (see page 5199)	This is constant Messages.WM_PARENTNOTIFY.
WM_PASTE (see page 5200)	This is constant Messages.WM_PASTE.
WM_PENWINFIRST (see page 5200)	This is constant Messages.WM_PENWINFIRST.
WM_PENWINLAST (see page 5200)	This is constant Messages.WM_PENWINLAST.
WM_POWER (see page 5201)	This is constant Messages.WM_POWER.
WM_POWERBROADCAST (see page 5201)	This is constant Messages.WM_POWERBROADCAST.
WM_PRINT (see page 5201)	This is constant Messages.WM_PRINT.
WM_PRINTCLIENT (see page 5202)	This is constant Messages.WM_PRINTCLIENT.
WM_QUERYDRAGICON (see page 5202)	This is constant Messages.WM_QUERYDRAGICON.
WM_QUERYENDSESSION (see page 5202)	This is constant Messages.WM_QUERYENDSESSION.
WM_QUERYNEWPALETTE (see page 5203)	This is constant Messages.WM_QUERYNEWPALETTE.
WM_QUERYOPEN (see page 5203)	This is constant Messages.WM_QUERYOPEN.
WM_QUERYUISTATE (see page 5203)	This is constant Messages.WM_QUERYUISTATE.
WM_QUEUESYNC (see page 5204)	This is constant Messages.WM_QUEUESYNC.
WM_QUIT (see page 5204)	This is constant Messages.WM_QUIT.
WM_RBUTTONDBLCLK (see page 5204)	This is constant Messages.WM_RBUTTONDBLCLK.
WM_RBUTTONDOWN (see page 5205)	This is constant Messages.WM_RBUTTONDOWN.
WM_RBUTTONUP (see page 5205)	This is constant Messages.WM_RBUTTONUP.
WM_RENDERALLFORMATS (see page 5205)	This is constant Messages.WM_RENDERALLFORMATS.
WM_RENDERFORMAT (see page 5206)	This is constant Messages.WM_RENDERFORMAT.
WM_SETCURSOR (see page 5206)	This is constant Messages.WM_SETCURSOR.
WM_SETFOCUS (see page 5206)	This is constant Messages.WM_SETFOCUS.
WM_SETFONT (see page 5207)	This is constant Messages.WM_SETFONT.

WM_SETHOTKEY (see page 5207)	This is constant Messages.WM_SETHOTKEY.
WM_SETICON (see page 5207)	This is constant Messages.WM_SETICON.
WM_SETREDRAW (see page 5208)	This is constant Messages.WM_SETREDRAW.
WM_SETTEXT (see page 5208)	This is constant Messages.WM_SETTEXT.
WM_SETTINGCHANGE (see page 5208)	This is constant Messages.WM_SETTINGCHANGE.
WM_SHOWWINDOW (see page 5209)	This is constant Messages.WM_SHOWWINDOW.
WM_SIZE (see page 5209)	This is constant Messages.WM_SIZE.
WM_SIZECLIPBOARD (see page 5209)	This is constant Messages.WM_SIZECLIPBOARD.
WM_SIZING (see page 5210)	This is constant Messages.WM_SIZING.
WM_SPOOLERSTATUS (see page 5210)	This is constant Messages.WM_SPOOLERSTATUS.
WM_STYLECHANGED (see page 5210)	This is constant Messages.WM_STYLECHANGED.
WM_STYLECHANGING (see page 5211)	This is constant Messages.WM_STYLECHANGING.
WM_SYSCHAR (see page 5211)	This is constant Messages.WM_SYSCHAR.
WM_SYSCOLORCHANGE (see page 5211)	This is constant Messages.WM_SYSCOLORCHANGE.
WM_SYSCOMMAND (see page 5212)	This is constant Messages.WM_SYSCOMMAND.
WM_SYSDEADCHAR (see page 5212)	This is constant Messages.WM_SYSDEADCHAR.
WM_SYSKEYDOWN (see page 5212)	This is constant Messages.WM_SYSKEYDOWN.
WM_SYSKEYUP (see page 5213)	This is constant Messages.WM_SYSKEYUP.
WM_SYSTEMERROR (see page 5213)	This is constant Messages.WM_SYSTEMERROR.
WM_TABLET_FIRST (see page 5213)	This is constant Messages.WM_TABLET_FIRST.
WM_TABLET_LAST (see page 5214)	This is constant Messages.WM_TABLET_LAST.
WM_TCARD (see page 5214)	This is constant Messages.WM_TCARD.
WM_THEMECHANGED (see page 5214)	This is constant Messages.WM_THEMECHANGED.
WM_TIMECHANGE (see page 5215)	This is constant Messages.WM_TIMECHANGE.
WM_TIMER (see page 5215)	This is constant Messages.WM_TIMER.
WM_UNDO (see page 5215)	This is constant Messages.WM_UNDO.
WM_UNICHAR (see page 5216)	This is constant Messages.WM_UNICHAR.
WM_UNINITMENUPOPUP (see page 5216)	This is constant Messages.WM_UNINITMENUPOPUP.
WM_UPDATEUISTATE (see page 5216)	This is constant Messages.WM_UPDATEUISTATE.
WM_USER (see page 5217)	This is constant Messages.WM_USER.
WM_USERCHANGED (see page 5217)	This is constant Messages.WM_USERCHANGED.
WM_VKEYTOITEM (see page 5217)	This is constant Messages.WM_VKEYTOITEM.
WM_VSCROLL (see page 5218)	This is constant Messages.WM_VSCROLL.
WM_VSCROLLCLIPBOARD (see page 5218)	This is constant Messages.WM_VSCROLLCLIPBOARD.
WM_WINDOWPOSCHANGED (see page 5218)	This is constant Messages.WM_WINDOWPOSCHANGED.
WM_WINDOWPOSCHANGING (see page 5219)	This is constant Messages.WM_WINDOWPOSCHANGING.
WM_WININICHANGE (see page 5219)	This is constant Messages.WM_WININICHANGE.
WM_WTSSESSION_CHANGE (see page 5219)	This is constant Messages.WM_WTSSESSION_CHANGE.

49.4.1 Messages.BM_CLICK Constant

Pascal

```
BM_CLICK = $00F5;
```

C++

```
BM_CLICK = 0x00F5;
```

File

Messages

Description

This is constant Messages.BM_CLICK.

49.4.2 Messages.BM_GETCHECK Constant

Pascal

```
BM_GETCHECK = $00F0;
```

C++

```
BM_GETCHECK = 0x00F0;
```

File

Messages

Description

This is constant Messages.BM_GETCHECK.

49.4.3 Messages.BM_GETIMAGE Constant

Pascal

```
BM_GETIMAGE = $00F6;
```

C++

```
BM_GETIMAGE = 0x00F6;
```

File

Messages

Description

This is constant Messages.BM_GETIMAGE.

49.4.4 Messages.BM_GETSTATE Constant

Pascal

```
BM_GETSTATE = $00F2;
```

C++

```
BM_GETSTATE = 0x00F2;
```

File

Messages

Description

This is constant Messages.BM_GETSTATE.

49.4.5 Messages.BM_SETCHECK Constant

Pascal

```
BM_SETCHECK = $00F1;
```

C++

```
BM_SETCHECK = 0x00F1;
```

File

Messages

Description

This is constant Messages.BM_SETCHECK.

49.4.6 Messages.BM_SETIMAGE Constant

Pascal

```
BM_SETIMAGE = $00F7;
```

C++

```
BM_SETIMAGE = 0x00F7;
```

File

Messages

Description

This is constant Messages.BM_SETIMAGE.

49.4.7 Messages.BM_SETSTATE Constant

Pascal

```
BM_SETSTATE = $00F3;
```

C++

```
BM_SETSTATE = 0x00F3;
```

File

Messages

Description

This is constant Messages.BM_SETSTATE.

49.4.8 Messages.BM_SETSTYLE Constant

Pascal

```
BM_SETSTYLE = $00F4;
```

C++

```
BM_SETSTYLE = 0x00F4;
```

File

Messages

Description

This is constant Messages.BM_SETSTYLE.

49.4.9 Messages.BN_CLICKED Constant

Pascal

```
BN_CLICKED = 0;
```

C++

```
BN_CLICKED = 0;
```

File

Messages

Description

This is constant Messages.BN_CLICKED.

49.4.10 Messages.BN_DBLCLK Constant

Pascal

```
BN_DBLCLK = BN_DOUBLECLICKED;
```

C++

```
BN_DBLCLK = BN_DOUBLECLICKED;
```

File

Messages

Description

This is constant Messages.BN_DBLCLK.

49.4.11 Messages.BN_DISABLE Constant

Pascal

```
BN_DISABLE = 4;
```

C++

```
BN_DISABLE = 4;
```

File

Messages

Description

This is constant Messages.BN_DISABLE.

49.4.12 Messages.BN_DOUBLECLICKED Constant

Pascal

```
BN_DOUBLECLICKED = 5;
```

C++

```
BN_DOUBLECLICKED = 5;
```

File

Messages

Description

This is constant Messages.BN_DOUBLECLICKED.

49.4.13 Messages.BN_HILITE Constant

Pascal

```
BN_HILITE = 2;
```

C++

```
BN_HILITE = 2;
```

File

Messages

Description

This is constant Messages.BN_HILITE.

49.4.14 Messages.BN_KILLFOCUS Constant

Pascal

```
BN_KILLFOCUS = 7;
```

C++

```
BN_KILLFOCUS = 7;
```

File

Messages

Description

This is constant Messages.BN_KILLFOCUS.

49.4.15 Messages.BN_PAINT Constant

Pascal

```
BN_PAINT = 1;
```

C++

```
BN_PAINT = 1;
```

File

Messages

Description

This is constant Messages.BN_PAINT.

49.4.16 Messages.BN_PUSHED Constant

Pascal

```
BN_PUSHED = BN_HILITE;
```

C++

```
BN_PUSHED = BN_HILITE;
```

File

Messages

Description

This is constant Messages.BN_PUSHED.

49.4.17 Messages.BN_SETFOCUS Constant

Pascal

```
BN_SETFOCUS = 6;
```

C++

```
BN_SETFOCUS = 6;
```

File

Messages

Description

This is constant Messages.BN_SETFOCUS.

49.4.18 Messages.BN_UNHILITE Constant

Pascal

```
BN_UNHILITE = 3;
```

C++

```
BN_UNHILITE = 3;
```

File

Messages

Description

This is constant Messages.BN_UNHILITE.

49.4.19 Messages.BN_UNPUSHED Constant

Pascal

```
BN_UNPUSHED = BN_UNHILITE;
```

C++

```
BN_UNPUSHED = BN_UNHILITE;
```

File

Messages

Description

This is constant Messages.BN_UNPUSHED.

49.4.20 Messages.CB_ADDSTRING Constant

Pascal

```
CB_ADDSTRING = $0143;
```

C++

```
CB_ADDSTRING = 0x0143;
```

File

Messages

Description

This is constant Messages.CB_ADDSTRING.

49.4.21 Messages.CB_DELETESTRING Constant

Pascal

```
CB_DELETESTRING = $0144;
```

C++

```
CB_DELETESTRING = 0x0144;
```

File

Messages

Description

This is constant Messages.CB_DELETESTRING.

49.4.22 Messages.CB_DIR Constant

Pascal

```
CB_DIR = $0145;
```

C++

```
CB_DIR = 0x0145;
```

File

Messages

Description

This is constant Messages.CB_DIR.

49.4.23 Messages.CB_FINDSTRING Constant

Pascal

```
CB_FINDSTRING = $014C;
```

C++

```
CB_FINDSTRING = 0x014C;
```

File

Messages

Description

This is constant Messages.CB_FINDSTRING.

49.4.24 Messages.CB_FINDSTRINGEXACT Constant

Pascal

```
CB_FINDSTRINGEXACT = $0158;
```

C++

```
CB_FINDSTRINGEXACT = 0x0158;
```

File

Messages

Description

This is constant Messages.CB_FINDSTRINGEXACT.

49.4.25 Messages.CB_GETCOUNT Constant

Pascal

```
CB_GETCOUNT = $0146;
```

C++

```
CB_GETCOUNT = 0x0146;
```

File

Messages

Description

This is constant Messages.CB_GETCOUNT.

49.4.26 Messages.CB_GETCURSEL Constant

Pascal

```
CB_GETCURSEL = $0147;
```

C++

```
CB_GETCURSEL = 0x0147;
```

File

Messages

Description

This is constant Messages.CB_GETCURSEL.

49.4.27 Messages.CB_GETDROPPEDCONTROLRECT Constant

Pascal

```
CB_GETDROPPEDCONTROLRECT = $0152;
```

C++

```
CB_GETDROPPEDCONTROLRECT = 0x0152;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDCONTROLRECT.

49.4.28 Messages.CB_GETDROPPEDSTATE Constant

Pascal

```
CB_GETDROPPEDSTATE = $0157;
```

C++

```
CB_GETDROPPEDSTATE = 0x0157;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDSTATE.

49.4.29 Messages.CB_GETDROPPEDWIDTH Constant

Pascal

```
CB_GETDROPPEDWIDTH = 351;
```

C++

```
CB_GETDROPPEDWIDTH = 351;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDWIDTH.

49.4.30 Messages.CB_GETEDITSEL Constant

Pascal

```
CB_GETEDITSEL = $0140;
```

C++

```
CB_GETEDITSEL = 0x0140;
```

File

Messages

Description

This is constant Messages.CB_GETEDITSEL.

49.4.31 Messages.CB_GETEXTENDEDUI Constant

Pascal

```
CB_GETEXTENDEDUI = $0156;
```

C++

```
CB_GETEXTENDEDUI = 0x0156;
```

File

Messages

Description

This is constant Messages.CB_GETEXTENDEDUI.

49.4.32 Messages.CB_GETHORIZONTALEXTENT Constant

Pascal

```
CB_GETHORIZONTALEXTENT = 349;
```

C++

```
CB_GETHORIZONTALEXTENT = 349;
```

File

Messages

Description

This is constant Messages.CB_GETHORIZONTALEXTENT.

49.4.33 Messages.CB_GETITEMDATA Constant

Pascal

```
CB_GETITEMDATA = $0150;
```

C++

```
CB_GETITEMDATA = 0x0150;
```

File

Messages

Description

This is constant Messages.CB_GETITEMDATA.

49.4.34 Messages.CB_GETITEMHEIGHT Constant

Pascal

```
CB_GETITEMHEIGHT = $0154;
```

C++

```
CB_GETITEMHEIGHT = 0x0154;
```

File

Messages

Description

This is constant Messages.CB_GETITEMHEIGHT.

49.4.35 Messages.CB_GETLBTEXT Constant

Pascal

```
CB_GETLBTEXT = $0148;
```

C++

```
CB_GETLBTEXT = 0x0148;
```

File

Messages

Description

This is constant Messages.CB_GETLBTEXT.

49.4.36 Messages.CB_GETLBTEXTLEN Constant

Pascal

```
CB_GETLBTEXTLEN = $0149;
```

C++

```
CB_GETLBTEXTLEN = 0x0149;
```

File

Messages

Description

This is constant Messages.CB_GETLBTEXTLEN.

49.4.37 Messages.CB_GETLOCALE Constant

Pascal

```
CB_GETLOCALE = 346;
```

C++

```
CB_GETLOCALE = 346;
```

File

Messages

Description

This is constant Messages.CB_GETLOCALE.

49.4.38 Messages.CB_GETTOPINDEX Constant

Pascal

```
CB_GETTOPINDEX = 347;
```

C++

```
CB_GETTOPINDEX = 347;
```

File

Messages

Description

This is constant Messages.CB_GETTOPINDEX.

49.4.39 Messages.CB_INITSTORAGE Constant

Pascal

```
CB_INITSTORAGE = 353;
```

C++

```
CB_INITSTORAGE = 353;
```

File

Messages

Description

This is constant Messages.CB_INITSTORAGE.

49.4.40 Messages.CB_INSERTSTRING Constant

Pascal

```
CB_INSERTSTRING = $014A;
```

C++

```
CB_INSERTSTRING = 0x014A;
```

File

Messages

Description

This is constant Messages.CB_INSERTSTRING.

49.4.41 Messages.CB_LIMITTEXT Constant

Pascal

```
CB_LIMITTEXT = $0141;
```

C++

```
CB_LIMITTEXT = 0x0141;
```

File

Messages

Description

This is constant Messages.CB_LIMITTEXT.

49.4.42 Messages.CB_MSGMAX Constant

Pascal

```
CB_MSGMAX = 354;
```

C++

```
CB_MSGMAX = 354;
```

File

Messages

Description

This is constant Messages.CB_MSGMAX.

49.4.43 Messages.CB_RESETCONTENT Constant

Pascal

```
CB_RESETCONTENT = $014B;
```

C++

```
CB_RESETCONTENT = 0x014B;
```

File

Messages

Description

This is constant Messages.CB_RESETCONTENT.

49.4.44 Messages.CB_SELECTSTRING Constant

Pascal

```
CB_SELECTSTRING = $014D;
```

C++

```
CB_SELECTSTRING = 0x014D;
```

File

Messages

Description

This is constant Messages.CB_SELECTSTRING.

49.4.45 Messages.CB_SETCURSEL Constant

Pascal

```
CB_SETCURSEL = $014E;
```

C++

```
CB_SETCURSEL = 0x014E;
```

File

Messages

Description

This is constant Messages.CB_SETCURSEL.

49.4.46 Messages.CB_SETDROPPEDWIDTH Constant

Pascal

```
CB_SETDROPPEDWIDTH = 352;
```

C++

```
CB_SETDROPPEDWIDTH = 352;
```

File

Messages

Description

This is constant Messages.CB_SETDROPPEDWIDTH.

49.4.47 Messages.CB_SETEDITSEL Constant

Pascal

```
CB_SETEDITSEL = $0142;
```

C++

```
CB_SETEDITSEL = 0x0142;
```

File

Messages

Description

This is constant Messages.CB_SETEDITSEL.

49.4.48 Messages.CB_SETEXTENDEDUI Constant

Pascal

```
CB_SETEXTENDEDUI = $0155;
```

C++

```
CB_SETEXTENDEDUI = 0x0155;
```

File

Messages

Description

This is constant Messages.CB_SETEXTENDEDUI.

49.4.49 Messages.CB_SETHORIZONTALEXTENT Constant

Pascal

```
CB_SETHORIZONTALEXTENT = 350;
```

C++

```
CB_SETHORIZONTALEXTENT = 350;
```

File

Messages

Description

This is constant Messages.CB_SETHORIZONTALEXTENT.

49.4.50 Messages.CB_SETITEMDATA Constant

Pascal

```
CB_SETITEMDATA = $0151;
```

C++

```
CB_SETITEMDATA = 0x0151;
```

File

Messages

Description

This is constant Messages.CB_SETITEMDATA.

49.4.51 Messages.CB_SETITEMHEIGHT Constant

Pascal

```
CB_SETITEMHEIGHT = $0153;
```

C++

```
CB_SETITEMHEIGHT = 0x0153;
```

File

Messages

Description

This is constant Messages.CB_SETITEMHEIGHT.

49.4.52 Messages.CB_SETLOCALE Constant

Pascal

```
CB_SETLOCALE = 345;
```

C++

```
CB_SETLOCALE = 345;
```

File

Messages

Description

This is constant Messages.CB_SETLOCALE.

49.4.53 Messages.CB_SETTOPINDEX Constant

Pascal

```
CB_SETTOPINDEX = 348;
```

C++

```
CB_SETTOPINDEX = 348;
```

File

Messages

Description

This is constant Messages.CB_SETTOPINDEX.

49.4.54 Messages.CB_SHOWDROPDOWN Constant

Pascal

```
CB_SHOWDROPDOWN = $014F;
```

C++

```
CB_SHOWDROPDOWN = 0x014F;
```

File

Messages

Description

This is constant Messages.CB_SHOWDROPDOWN.

49.4.55 Messages.CBN_CLOSEUP Constant

Pascal

```
CBN_CLOSEUP = 8;
```

C++

```
CBN_CLOSEUP = 8;
```

File

Messages

Description

This is constant Messages.CBN_CLOSEUP.

49.4.56 Messages.CBN_DBLCLK Constant

Pascal

```
CBN_DBLCLK = 2;
```

C++

```
CBN_DBLCLK = 2;
```

File

Messages

Description

This is constant Messages.CBN_DBLCLK.

49.4.57 Messages.CBN_DROPDOWN Constant

Pascal

```
CBN_DROPDOWN = 7;
```

C++

```
CBN_DROPDOWN = 7;
```

File

Messages

Description

This is constant Messages.CBN_DROPDOWN.

49.4.58 Messages.CBN_EDITCHANGE Constant

Pascal

```
CBN_EDITCHANGE = 5;
```

C++

```
CBN_EDITCHANGE = 5;
```

File

Messages

Description

This is constant Messages.CBN_EDITCHANGE.

49.4.59 Messages.CBN_EDITUPDATE Constant

Pascal

```
CBN_EDITUPDATE = 6;
```

C++

```
CBN_EDITUPDATE = 6;
```

File

Messages

Description

This is constant Messages.CBN_EDITUPDATE.

49.4.60 Messages.CBN_ERRSPACE Constant

Pascal

```
CBN_ERRSPACE = (-1);
```

C++

```
CBN_ERRSPACE = (-1);
```

File

Messages

Description

This is constant Messages.CBN_ERRSPACE.

49.4.61 Messages.CBN_KILLFOCUS Constant

Pascal

```
CBN_KILLFOCUS = 4;
```

C++

```
CBN_KILLFOCUS = 4;
```

File

Messages

Description

This is constant Messages.CBN_KILLFOCUS.

49.4.62 Messages.CBN_SELCHANGE Constant

Pascal

```
CBN_SELCHANGE = 1;
```

C++

```
CBN_SELCHANGE = 1;
```

File

Messages

Description

This is constant Messages.CBN_SELCHANGE.

49.4.63 Messages.CBN_SELENDCANCEL Constant

Pascal

```
CBN_SELENDCANCEL = 10;
```

C++

```
CBN_SELENDCANCEL = 10;
```

File

Messages

Description

This is constant Messages.CBN_SELENDCANCEL.

49.4.64 Messages.CBN_SELENDOK Constant

Pascal

```
CBN_SELENDOK = 9;
```

C++

```
CBN_SELENDOK = 9;
```

File

Messages

Description

This is constant Messages.CBN_SELENDOK.

49.4.65 Messages.CBN_SETFOCUS Constant

Pascal

```
CBN_SETFOCUS = 3;
```

C++

```
CBN_SETFOCUS = 3;
```

File

Messages

Description

This is constant Messages.CBN_SETFOCUS.

49.4.66 Messages.DM_GETDEFID Constant

Pascal

```
DM_GETDEFID = (WM_USER+0);
```

C++

```
DM_GETDEFID = (WM_USER+0);
```

File

Messages

Description

This is constant Messages.DM_GETDEFID.

49.4.67 Messages.DM_REPOSITION Constant

Pascal

```
DM_REPOSITION = (WM_USER+2);
```

C++

```
DM_REPOSITION = (WM_USER+2);
```

File

Messages

Description

This is constant Messages.DM_REPOSITION.

49.4.68 Messages.DM_SETDEFID Constant

Pascal

```
DM_SETDEFID = (WM_USER+1);
```

C++

```
DM_SETDEFID = (WM_USER+1);
```

File

Messages

Description

This is constant Messages.DM_SETDEFID.

49.4.69 Messages.EM_CANUNDO Constant

Pascal

```
EM_CANUNDO = $00C6;
```

C++

```
EM_CANUNDO = 0x00C6;
```

File

Messages

Description

This is constant Messages.EM_CANUNDO.

49.4.70 Messages.EM_CHARFROMPOS Constant

Pascal

```
EM_CHARFROMPOS = 215;
```

C++

```
EM_CHARFROMPOS = 215;
```

File

Messages

Description

This is constant Messages.EM_CHARFROMPOS.

49.4.71 Messages.EM_EMPTYUNDOBUFFER Constant

Pascal

```
EM_EMPTYUNDOBUFFER = $00CD;
```

C++

```
EM_EMPTYUNDOBUFFER = 0x00CD;
```

File

Messages

Description

This is constant Messages.EM_EMPTYUNDOBUFFER.

49.4.72 Messages.EM_FMTLINES Constant

Pascal

```
EM_FMTLINES = $00C8;
```

C++

```
EM_FMTLINES = 0x00C8;
```

File

Messages

Description

This is constant Messages.EM_FMTLINES.

49.4.73 Messages.EM_GETFIRSTVISIBLELINE Constant

Pascal

```
EM_GETFIRSTVISIBLELINE = $00CE;
```

C++

```
EM_GETFIRSTVISIBLELINE = 0x00CE;
```

File

Messages

Description

This is constant Messages.EM_GETFIRSTVISIBLELINE.

49.4.74 Messages.EM_GETHANDLE Constant

Pascal

```
EM_GETHANDLE = $00BD;
```

C++

```
EM_GETHANDLE = 0x00BD;
```

File

Messages

Description

This is constant Messages.EM_GETHANDLE.

49.4.75 Messages.EM_GETIMESTATUS Constant

Pascal

```
EM_GETIMESTATUS = 217;
```

C++

```
EM_GETIMESTATUS = 217;
```

File

Messages

Description

This is constant Messages.EM_GETIMESTATUS.

49.4.76 Messages.EM_GETLIMITTEXT Constant

Pascal

```
EM_GETLIMITTEXT = 213;
```

C++

```
EM_GETLIMITTEXT = 213;
```

File

Messages

Description

This is constant Messages.EM_GETLIMITTEXT.

49.4.77 Messages.EM_GETLINE Constant

Pascal

```
EM_GETLINE = $00C4;
```

C++

```
EM_GETLINE = 0x00C4;
```

File

Messages

Description

This is constant Messages.EM_GETLINE.

49.4.78 Messages.EM_GETLINECOUNT Constant

Pascal

```
EM_GETLINECOUNT = $00BA;
```

C++

```
EM_GETLINECOUNT = 0x00BA;
```

File

Messages

Description

This is constant Messages.EM_GETLINECOUNT.

49.4.79 Messages.EM_GETMARGINS Constant

Pascal

```
EM_GETMARGINS = 212;
```

C++

```
EM_GETMARGINS = 212;
```

File

Messages

Description

This is constant Messages.EM_GETMARGINS.

49.4.80 Messages.EM_GETMODIFY Constant

Pascal

```
EM_GETMODIFY = $00B8;
```

C++

```
EM_GETMODIFY = 0x00B8;
```

File

Messages

Description

This is constant Messages.EM_GETMODIFY.

49.4.81 Messages.EM_GETPASSWORDCHAR Constant

Pascal

```
EM_GETPASSWORDCHAR = $00D2;
```

C++

```
EM_GETPASSWORDCHAR = 0x00D2;
```

File

Messages

Description

This is constant Messages.EM_GETPASSWORDCHAR.

49.4.82 Messages.EM_GETRECT Constant

Pascal

```
EM_GETRECT = $00B2;
```

C++

```
EM_GETRECT = 0x00B2;
```

File

Messages

Description

This is constant Messages.EM_GETRECT.

49.4.83 Messages.EM_GETSEL Constant

Pascal

```
EM_GETSEL = $00B0;
```

C++

```
EM_GETSEL = 0x00B0;
```

File

Messages

Description

This is constant Messages.EM_GETSEL.

49.4.84 Messages.EM_GETTHUMB Constant

Pascal

```
EM_GETTHUMB = $00BE;
```

C++

```
EM_GETTHUMB = 0x00BE;
```

File

Messages

Description

This is constant Messages.EM_GETTHUMB.

49.4.85 Messages.EM_GETWORDBREAKPROC Constant

Pascal

```
EM_GETWORDBREAKPROC = $00D1;
```

C++

```
EM_GETWORDBREAKPROC = 0x00D1;
```

File

Messages

Description

This is constant Messages.EM_GETWORDBREAKPROC.

49.4.86 Messages.EM_LIMITTEXT Constant

Pascal

```
EM_LIMITTEXT = $00C5;
```

C++

```
EM_LIMITTEXT = 0x00C5;
```

File

Messages

Description

This is constant Messages.EM_LIMITTEXT.

49.4.87 Messages.EM_LINEFROMCHAR Constant

Pascal

```
EM_LINEFROMCHAR = $00C9;
```

C++

```
EM_LINEFROMCHAR = 0x00C9;
```

File

Messages

Description

This is constant Messages.EM_LINEFROMCHAR.

49.4.88 Messages.EM_LINEINDEX Constant

Pascal

```
EM_LINEINDEX = $00BB;
```

C++

```
EM_LINEINDEX = 0x00BB;
```

File

Messages

Description

This is constant Messages.EM_LINEINDEX.

49.4.89 Messages.EM_LINELENGTH Constant

Pascal

```
EM_LINELENGTH = $00C1;
```

C++

```
EM_LINELENGTH = 0x00C1;
```

File

Messages

Description

This is constant Messages.EM_LINELENGTH.

49.4.90 Messages.EM_LINESCROLL Constant

Pascal

```
EM_LINESCROLL = $00B6;
```

C++

```
EM_LINESCROLL = 0x00B6;
```

File

Messages

Description

This is constant Messages.EM_LINESCROLL.

49.4.91 Messages.EM_POSFROMCHAR Constant

Pascal

```
EM_POSFROMCHAR = 214;
```

C++

```
EM_POSFROMCHAR = 214;
```

File

Messages

Description

This is constant Messages.EM_POSFROMCHAR.

49.4.92 Messages.EM_REPLACESEL Constant

Pascal

```
EM_REPLACESEL = $00C2;
```

C++

```
EM_REPLACESEL = 0x00C2;
```

File

Messages

Description

This is constant Messages.EM_REPLACESEL.

49.4.93 Messages.EM_SCROLL Constant

Pascal

```
EM_SCROLL = $00B5;
```

C++

```
EM_SCROLL = 0x00B5;
```

File

Messages

Description

This is constant Messages.EM_SCROLL.

49.4.94 Messages.EM_SCROLLCARET Constant

Pascal

```
EM_SCROLLCARET = $00B7;
```

C++

```
EM_SCROLLCARET = 0x00B7;
```

File

Messages

Description

This is constant Messages.EM_SCROLLCARET.

49.4.95 Messages.EM_SETHANDLE Constant

Pascal

```
EM_SETHANDLE = $00BC;
```

C++

```
EM_SETHANDLE = 0x00BC;
```

File

Messages

Description

This is constant Messages.EM_SETHANDLE.

49.4.96 Messages.EM_SETIMESTATUS Constant

Pascal

```
EM_SETIMESTATUS = 216;
```

C++

```
EM_SETIMESTATUS = 216;
```

File

Messages

Description

This is constant Messages.EM_SETIMESTATUS.

49.4.97 Messages.EM_SETLIMITTEXT Constant

Pascal

```
EM_SETLIMITTEXT = EM_LIMITTEXT;
```

C++

```
EM_SETLIMITTEXT = EM_LIMITTEXT;
```

File

Messages

Description

This is constant Messages.EM_SETLIMITTEXT.

49.4.98 Messages.EM_SETMARGINS Constant

Pascal

```
EM_SETMARGINS = 211;
```

C++

```
EM_SETMARGINS = 211;
```

File

Messages

Description

This is constant Messages.EM_SETMARGINS.

49.4.99 Messages.EM_SETMODIFY Constant

Pascal

```
EM_SETMODIFY = $00B9;
```

C++

```
EM_SETMODIFY = 0x00B9;
```

File

Messages

Description

This is constant Messages.EM_SETMODIFY.

49.4.100 Messages.EM_SETPASSWORDCHAR Constant

Pascal

```
EM_SETPASSWORDCHAR = $00CC;
```

C++

```
EM_SETPASSWORDCHAR = 0x00CC;
```

File

Messages

Description

This is constant Messages.EM_SETPASSWORDCHAR.

49.4.101 Messages.EM_SETREADONLY Constant

Pascal

```
EM_SETREADONLY = $00CF;
```

C++

```
EM_SETREADONLY = 0x00CF;
```

File

Messages

Description

This is constant Messages.EM_SETREADONLY.

49.4.102 Messages.EM_SETRECT Constant

Pascal

```
EM_SETRECT = $00B3;
```

C++

```
EM_SETRECT = 0x00B3;
```

File

Messages

Description

This is constant Messages.EM_SETRECT.

49.4.103 Messages.EM_SETRECTNP Constant

Pascal

```
EM_SETRECTNP = $00B4;
```

C++

```
EM_SETRECTNP = 0x00B4;
```

File

Messages

Description

This is constant Messages.EM_SETRECTNP.

49.4.104 Messages.EM_SETSEL Constant

Pascal

```
EM_SETSEL = $00B1;
```

C++

```
EM_SETSEL = 0x00B1;
```

File

Messages

Description

This is constant Messages.EM_SETSEL.

49.4.105 Messages.EM_SETTABSTOPS Constant

Pascal

```
EM_SETTABSTOPS = $00CB;
```

C++

```
EM_SETTABSTOPS = 0x00CB;
```

File

Messages

Description

This is constant Messages.EM_SETTABSTOPS.

49.4.106 Messages.EM_SETWORDBREAKPROC Constant

Pascal

```
EM_SETWORDBREAKPROC = $00D0;
```

C++

```
EM_SETWORDBREAKPROC = 0x00D0;
```

File

Messages

Description

This is constant Messages.EM_SETWORDBREAKPROC.

49.4.107 Messages.EM_UNDO Constant

Pascal

```
EM_UNDO = $00C7;
```

C++

```
EM_UNDO = 0x00C7;
```

File

Messages

Description

This is constant Messages.EM_UNDO.

49.4.108 Messages.EN_CHANGE Constant

Pascal

```
EN_CHANGE = $0300;
```

C++

```
EN_CHANGE = 0x0300;
```

File

Messages

Description

This is constant Messages.EN_CHANGE.

49.4.109 Messages.EN_ERRSPACE Constant

Pascal

```
EN_ERRSPACE = $0500;
```

C++

```
EN_ERRSPACE = 0x0500;
```

File

Messages

Description

This is constant Messages.EN_ERRSPACE.

49.4.110 Messages.EN_HSCROLL Constant

Pascal

```
EN_HSCROLL = $0601;
```

C++

```
EN_HSCROLL = 0x0601;
```

File

Messages

Description

This is constant Messages.EN_HSCROLL.

49.4.111 Messages.EN_KILLFOCUS Constant

Pascal

```
EN_KILLFOCUS = $0200;
```

C++

```
EN_KILLFOCUS = 0x0200;
```

File

Messages

Description

This is constant Messages.EN_KILLFOCUS.

49.4.112 Messages.EN_MAXTEXT Constant

Pascal

```
EN_MAXTEXT = $0501;
```

C++

```
EN_MAXTEXT = 0x0501;
```

File

Messages

Description

This is constant Messages.EN_MAXTEXT.

49.4.113 Messages.EN_SETFOCUS Constant

Pascal

```
EN_SETFOCUS = $0100;
```

C++

```
EN_SETFOCUS = 0x0100;
```

File

Messages

Description

This is constant Messages.EN_SETFOCUS.

49.4.114 Messages.EN_UPDATE Constant

Pascal

```
EN_UPDATE = $0400;
```

C++

```
EN_UPDATE = 0x0400;
```

File

Messages

Description

This is constant Messages.EN_UPDATE.

49.4.115 Messages.EN_VSCROLL Constant

Pascal

```
EN_VSCROLL = $0602;
```

C++

```
EN_VSCROLL = 0x0602;
```

File

Messages

Description

This is constant Messages.EN_VSCROLL.

49.4.116 Messages.LB_ADDFILE Constant

Pascal

```
LB_ADDFILE = $0196;
```

C++

```
LB_ADDFILE = 0x0196;
```

File

Messages

Description

This is constant Messages.LB_ADDFILE.

49.4.117 Messages.LB_ADDSTRING Constant

Pascal

```
LB_ADDSTRING = $0180;
```

C++

```
LB_ADDSTRING = 0x0180;
```

File

Messages

Description

This is constant Messages.LB_ADDSTRING.

49.4.118 Messages.LB_DELETESTRING Constant

Pascal

```
LB_DELETESTRING = $0182;
```

C++

```
LB_DELETESTRING = 0x0182;
```

File

Messages

Description

This is constant Messages.LB_DELETESTRING.

49.4.119 Messages.LB_DIR Constant

Pascal

```
LB_DIR = $018D;
```

C++

```
LB_DIR = 0x018D;
```

File

Messages

Description

This is constant Messages.LB_DIR.

49.4.120 Messages.LB_FINDSTRING Constant

Pascal

```
LB_FINDSTRING = $018F;
```

C++

```
LB_FINDSTRING = 0x018F;
```

File

Messages

Description

This is constant Messages.LB_FINDSTRING.

49.4.121 Messages.LB_FINDSTRINGEXACT Constant

Pascal

```
LB_FINDSTRINGEXACT = $01A2;
```

C++

```
LB_FINDSTRINGEXACT = 0x01A2;
```

File

Messages

Description

This is constant Messages.LB_FINDSTRINGEXACT.

49.4.122 Messages.LB_GETANCHORINDEX Constant

Pascal

```
LB_GETANCHORINDEX = $019D;
```

C++

```
LB_GETANCHORINDEX = 0x019D;
```

File

Messages

Description

This is constant Messages.LB_GETANCHORINDEX.

49.4.123 Messages.LB_GETCARETINDEX Constant

Pascal

```
LB_GETCARETINDEX = $019F;
```

C++

```
LB_GETCARETINDEX = 0x019F;
```

File

Messages

Description

This is constant Messages.LB_GETCARETINDEX.

49.4.124 Messages.LB_GETCOUNT Constant

Pascal

```
LB_GETCOUNT = $018B;
```

C++

```
LB_GETCOUNT = 0x018B;
```

File

Messages

Description

This is constant Messages.LB_GETCOUNT.

49.4.125 Messages.LB_GETCURSEL Constant

Pascal

```
LB_GETCURSEL = $0188;
```

C++

```
LB_GETCURSEL = 0x0188;
```

File

Messages

Description

This is constant Messages.LB_GETCURSEL.

49.4.126 Messages.LB_GETHORIZONTALEXTENT Constant

Pascal

```
LB_GETHORIZONTALEXTENT = $0193;
```

C++

```
LB_GETHORIZONTALEXTENT = 0x0193;
```

File

Messages

Description

This is constant Messages.LB_GETHORIZONTALEXTENT.

49.4.127 Messages.LB_GETITEMDATA Constant

Pascal

```
LB_GETITEMDATA = $0199;
```

C++

```
LB_GETITEMDATA = 0x0199;
```

File

Messages

Description

This is constant Messages.LB_GETITEMDATA.

49.4.128 Messages.LB_GETITEMHEIGHT Constant

Pascal

```
LB_GETITEMHEIGHT = $01A1;
```

C++

```
LB_GETITEMHEIGHT = 0x01A1;
```

File

Messages

Description

This is constant Messages.LB_GETITEMHEIGHT.

49.4.129 Messages.LB_GETITEMRECT Constant

Pascal

```
LB_GETITEMRECT = $0198;
```

C++

```
LB_GETITEMRECT = 0x0198;
```

File

Messages

Description

This is constant Messages.LB_GETITEMRECT.

49.4.130 Messages.LB_GETLOCALE Constant

Pascal

```
LB_GETLOCALE = $01A6;
```

C++

```
LB_GETLOCALE = 0x01A6;
```

File

Messages

Description

This is constant Messages.LB_GETLOCALE.

49.4.131 Messages.LB_GETSEL Constant

Pascal

```
LB_GETSEL = $0187;
```

C++

```
LB_GETSEL = 0x0187;
```

File

Messages

Description

This is constant Messages.LB_GETSEL.

49.4.132 Messages.LB_GETSELCOUNT Constant

Pascal

```
LB_GETSELCOUNT = $0190;
```

C++

```
LB_GETSELCOUNT = 0x0190;
```

File

Messages

Description

This is constant Messages.LB_GETSELCOUNT.

49.4.133 Messages.LB_GETSELITEMS Constant

Pascal

```
LB_GETSELITEMS = $0191;
```

C++

```
LB_GETSELITEMS = 0x0191;
```

File

Messages

Description

This is constant Messages.LB_GETSELITEMS.

49.4.134 Messages.LB_GETTEXT Constant

Pascal

```
LB_GETTEXT = $0189;
```

C++

```
LB_GETTEXT = 0x0189;
```

File

Messages

Description

This is constant Messages.LB_GETTEXT.

49.4.135 Messages.LB_GETTEXTLEN Constant

Pascal

```
LB_GETTEXTLEN = $018A;
```

C++

```
LB_GETTEXTLEN = 0x018A;
```

File

Messages

Description

This is constant Messages.LB_GETTEXTLEN.

49.4.136 Messages.LB_GETTOPINDEX Constant

Pascal

```
LB_GETTOPINDEX = $018E;
```

C++

```
LB_GETTOPINDEX = 0x018E;
```

File

Messages

Description

This is constant Messages.LB_GETTOPINDEX.

49.4.137 Messages.LB_INITSTORAGE Constant

Pascal

```
LB_INITSTORAGE = $01A8;
```

C++

```
LB_INITSTORAGE = 0x01A8;
```

File

Messages

Description

This is constant Messages.LB_INITSTORAGE.

49.4.138 Messages.LB_INSERTSTRING Constant

Pascal

```
LB_INSERTSTRING = $0181;
```

C++

```
LB_INSERTSTRING = 0x0181;
```

File

Messages

Description

This is constant Messages.LB_INSERTSTRING.

49.4.139 Messages.LB_ITEMFROMPOINT Constant

Pascal

```
LB_ITEMFROMPOINT = $01A9;
```

C++

```
LB_ITEMFROMPOINT = 0x01A9;
```

File

Messages

Description

This is constant Messages.LB_ITEMFROMPOINT.

49.4.140 Messages.LB_MSGMAX Constant

Pascal

```
LB_MSGMAX = 432;
```

C++

```
LB_MSGMAX = 432;
```

File

Messages

Description

This is constant Messages.LB_MSGMAX.

49.4.141 Messages.LB_RESETCONTENT Constant

Pascal

```
LB_RESETCONTENT = $0184;
```

C++

```
LB_RESETCONTENT = 0x0184;
```

File

Messages

Description

This is constant Messages.LB_RESETCONTENT.

49.4.142 Messages.LB_SELECTSTRING Constant

Pascal

```
LB_SELECTSTRING = $018C;
```

C++

```
LB_SELECTSTRING = 0x018C;
```

File

Messages

Description

This is constant Messages.LB_SELECTSTRING.

49.4.143 Messages.LB_SELITEMRANGE Constant

Pascal

```
LB_SELITEMRANGE = $019B;
```

C++

```
LB_SELITEMRANGE = 0x019B;
```

File

Messages

Description

This is constant Messages.LB_SELITEMRANGE.

49.4.144 Messages.LB_SELITEMRANGEEX Constant

Pascal

```
LB_SELITEMRANGEEX = $0183;
```

C++

```
LB_SELITEMRANGEEX = 0x0183;
```

File

Messages

Description

This is constant Messages.LB_SELITEMRANGEEX.

49.4.145 Messages.LB_SETANCHORINDEX Constant

Pascal

```
LB_SETANCHORINDEX = $019C;
```

C++

```
LB_SETANCHORINDEX = 0x019C;
```

File

Messages

Description

This is constant Messages.LB_SETANCHORINDEX.

49.4.146 Messages.LB_SETCARETINDEX Constant

Pascal

```
LB_SETCARETINDEX = $019E;
```

C++

```
LB_SETCARETINDEX = 0x019E;
```

File

Messages

Description

This is constant Messages.LB_SETCARETINDEX.

49.4.147 Messages.LB_SETCOLUMNWIDTH Constant

Pascal

```
LB_SETCOLUMNWIDTH = $0195;
```

C++

```
LB_SETCOLUMNWIDTH = 0x0195;
```

File

Messages

Description

This is constant Messages.LB_SETCOLUMNWIDTH.

49.4.148 Messages.LB_SETCOUNT Constant

Pascal

```
LB_SETCOUNT = $01A7;
```

C++

```
LB_SETCOUNT = 0x01A7;
```

File

Messages

Description

This is constant Messages.LB_SETCOUNT.

49.4.149 Messages.LB_SETCURSEL Constant

Pascal

```
LB_SETCURSEL = $0186;
```

C++

```
LB_SETCURSEL = 0x0186;
```

File

Messages

Description

This is constant Messages.LB_SETCURSEL.

49.4.150 Messages.LB_SETHORIZONTALEXTENT Constant

Pascal

```
LB_SETHORIZONTALEXTENT = $0194;
```

C++

```
LB_SETHORIZONTALEXTENT = 0x0194;
```

File

Messages

Description

This is constant Messages.LB_SETHORIZONTALEXTENT.

49.4.151 Messages.LB_SETITEMDATA Constant

Pascal

```
LB_SETITEMDATA = $019A;
```

C++

```
LB_SETITEMDATA = 0x019A;
```

File

Messages

Description

This is constant Messages.LB_SETITEMDATA.

49.4.152 Messages.LB_SETITEMHEIGHT Constant

Pascal

```
LB_SETITEMHEIGHT = $01A0;
```

C++

```
LB_SETITEMHEIGHT = 0x01A0;
```

File

Messages

Description

This is constant Messages.LB_SETITEMHEIGHT.

49.4.153 Messages.LB_SETLOCALE Constant

Pascal

```
LB_SETLOCALE = $01A5;
```

C++

```
LB_SETLOCALE = 0x01A5;
```

File

Messages

Description

This is constant Messages.LB_SETLOCALE.

49.4.154 Messages.LB_SETSEL Constant

Pascal

```
LB_SETSEL = $0185;
```

C++

```
LB_SETSEL = 0x0185;
```

File

Messages

Description

This is constant Messages.LB_SETSEL.

49.4.155 Messages.LB_SETTABSTOPS Constant

Pascal

```
LB_SETTABSTOPS = $0192;
```

C++

```
LB_SETTABSTOPS = 0x0192;
```

File

Messages

Description

This is constant Messages.LB_SETTABSTOPS.

49.4.156 Messages.LB_SETTOPINDEX Constant

Pascal

```
LB_SETTOPINDEX = $0197;
```

C++

```
LB_SETTOPINDEX = 0x0197;
```

File

Messages

Description

This is constant Messages.LB_SETTOPINDEX.

49.4.157 Messages.LBN_DBLCLK Constant

Pascal

```
LBN_DBLCLK = 2;
```

C++

```
LBN_DBLCLK = 2;
```

File

Messages

Description

This is constant Messages.LBN_DBLCLK.

49.4.158 Messages.LBN_ERRSPACE Constant

Pascal

```
LBN_ERRSPACE = (-2);
```

C++

```
LBN_ERRSPACE = (-2);
```

File

Messages

Description

This is constant Messages.LBN_ERRSPACE.

49.4.159 Messages.LBN_KILLFOCUS Constant

Pascal

```
LBN_KILLFOCUS = 5;
```

C++

```
LBN_KILLFOCUS = 5;
```

File

Messages

Description

This is constant Messages.LBN_KILLFOCUS.

49.4.160 Messages.LBN_SELCANCEL Constant

Pascal

```
LBN_SELCANCEL = 3;
```

C++

```
LBN_SELCANCEL = 3;
```

File

Messages

Description

This is constant Messages.LBN_SELCANCEL.

49.4.161 Messages.LBN_SELCHANGE Constant

Pascal

```
LBN_SELCHANGE = 1;
```

C++

```
LBN_SELCHANGE = 1;
```

File

Messages

Description

This is constant Messages.LBN_SELCHANGE.

49.4.162 Messages.LBN_SETFOCUS Constant

Pascal

```
LBN_SETFOCUS = 4;
```

C++

```
LBN_SETFOCUS = 4;
```

File

Messages

Description

This is constant Messages.LBN_SETFOCUS.

49.4.163 Messages.PSM_PAGEINFO Constant

Pascal

```
PSM_PAGEINFO = (WM_USER+100);
```

C++

```
PSM_PAGEINFO = (WM_USER+100);
```

File

Messages

Description

This is constant Messages.PSM_PAGEINFO.

49.4.164 Messages.PSM_SHEETINFO Constant

Pascal

```
PSM_SHEETINFO = (WM_USER+101);
```

C++

```
PSM_SHEETINFO = (WM_USER+101);
```

File

Messages

Description

This is constant Messages.PSM_SHEETINFO.

49.4.165 Messages.SBM_ENABLE_ARROWS Constant

Pascal

```
SBM_ENABLE_ARROWS = 228;
```

C++

```
SBM_ENABLE_ARROWS = 228;
```

File

Messages

Description

This is constant Messages.SBM_ENABLE_ARROWS.

49.4.166 Messages.SBM_GETPOS Constant

Pascal

```
SBM_GETPOS = 225;
```

C++

```
SBM_GETPOS = 225;
```

File

Messages

Description

This is constant Messages.SBM_GETPOS.

49.4.167 Messages.SBM_GETRANGE Constant

Pascal

```
SBM_GETRANGE = 227;
```

C++

```
SBM_GETRANGE = 227;
```

File

Messages

Description

This is constant Messages.SBM_GETRANGE.

49.4.168 Messages.SBM_GETSCROLLBARINFO Constant

Pascal

```
SBM_GETSCROLLBARINFO = 235;
```

C++

```
SBM_GETSCROLLBARINFO = 235;
```

File

Messages

Description

This is constant Messages.SBM_GETSCROLLBARINFO.

49.4.169 Messages.SBM_GETSCROLLINFO Constant

Pascal

```
SBM_GETSCROLLINFO = 234;
```

C++

```
SBM_GETSCROLLINFO = 234;
```

File

Messages

Description

This is constant Messages.SBM_GETSCROLLINFO.

49.4.170 Messages.SBM_SETPOS Constant

Pascal

```
SBM_SETPOS = 224;
```

C++

```
SBM_SETPOS = 224;
```

File

Messages

Description

This is constant Messages.SBM_SETPOS.

49.4.171 Messages.SBM_SETRANGE Constant

Pascal

```
SBM_SETRANGE = 226;
```

C++

```
SBM_SETRANGE = 226;
```

File

Messages

Description

This is constant Messages.SBM_SETRANGE.

49.4.172 Messages.SBM_SETRANGEREDRAW Constant

Pascal

```
SBM_SETRANGEREDRAW = 230;
```

C++

```
SBM_SETRANGEREDRAW = 230;
```

File

Messages

Description

This is constant Messages.SBM_SETRANGEREDRAW.

49.4.173 Messages.SBM_SETSCROLLINFO Constant

Pascal

```
SBM_SETSCROLLINFO = 233;
```

C++

```
SBM_SETSCROLLINFO = 233;
```

File

Messages

Description

This is constant Messages.SBM_SETSCROLLINFO.

49.4.174 Messages.WM_ACTIVATE Constant

Pascal

```
WM_ACTIVATE = $0006;
```

C++

```
WM_ACTIVATE = 0x0006;
```

File

Messages

Description

This is constant Messages.WM_ACTIVATE.

49.4.175 Messages.WM_ACTIVATEAPP Constant

Pascal

```
WM_ACTIVATEAPP = $001C;
```

C++

```
WM_ACTIVATEAPP = 0x001C;
```

File

Messages

Description

This is constant Messages.WM_ACTIVATEAPP.

49.4.176 Messages.WM_APP Constant

Pascal

```
WM_APP = $8000;
```

C++

```
WM_APP = 0x8000;
```

File

Messages

Description

This is constant Messages.WM_APP.

49.4.177 Messages.WM_APPCOMMAND Constant

Pascal

```
WM_APPCOMMAND = $0319;
```

C++

```
WM_APPCOMMAND = 0x0319;
```

File

Messages

Description

This is constant Messages.WM_APPCOMMAND.

49.4.178 Messages.WM_ASKCBFORMATNAME Constant

Pascal

```
WM_ASKCBFORMATNAME = $030C;
```

C++

```
WM_ASKCBFORMATNAME = 0x030C;
```

File

Messages

Description

This is constant Messages.WM_ASKCBFORMATNAME.

49.4.179 Messages.WM_CANCELJOURNAL Constant

Pascal

```
WM_CANCELJOURNAL = $004B;
```

C++

```
WM_CANCELJOURNAL = 0x004B;
```

File

Messages

Description

This is constant Messages.WM_CANCELJOURNAL.

49.4.180 Messages.WM_CANCELMODE Constant

Pascal

```
WM_CANCELMODE = $001F;
```

C++

```
WM_CANCELMODE = 0x001F;
```

File

Messages

Description

This is constant Messages.WM_CANCELMODE.

49.4.181 Messages.WM_CAPTURECHANGED Constant

Pascal

```
WM_CAPTURECHANGED = 533;
```

C++

```
WM_CAPTURECHANGED = 533;
```

File

Messages

Description

This is constant Messages.WM_CAPTURECHANGED.

49.4.182 Messages.WM_CHANGECBCHAIN Constant

Pascal

```
WM_CHANGECBCHAIN = $030D;
```

C++

```
WM_CHANGECBCHAIN = 0x030D;
```

File

Messages

Description

This is constant Messages.WM_CHANGECBCHAIN.

49.4.183 Messages.WM_CHANGEUISTATE Constant

Pascal

```
WM_CHANGEUISTATE = $0127;
```

C++

```
WM_CHANGEUISTATE = 0x0127;
```

File

Messages

Description

This is constant Messages.WM_CHANGEUISTATE.

49.4.184 Messages.WM_CHAR Constant

Pascal

```
WM_CHAR = $0102;
```

C++

```
WM_CHAR = 0x0102;
```

File

Messages

Description

This is constant Messages.WM_CHAR.

49.4.185 Messages.WM_CHARTOITEM Constant

Pascal

```
WM_CHARTOITEM = $002F;
```

C++

```
WM_CHARTOITEM = 0x002F;
```

File

Messages

Description

This is constant Messages.WM_CHARTOITEM.

49.4.186 Messages.WM_CHILDACTIVATE Constant

Pascal

```
WM_CHILDACTIVATE = $0022;
```

C++

```
WM_CHILDACTIVATE = 0x0022;
```

File

Messages

Description

This is constant Messages.WM_CHILDACTIVATE.

49.4.187 Messages.WM_CLEAR Constant

Pascal

```
WM_CLEAR = $0303;
```

C++

```
WM_CLEAR = 0x0303;
```

File

Messages

Description

This is constant Messages.WM_CLEAR.

49.4.188 Messages.WM_CLOSE Constant

Pascal

```
WM_CLOSE = $0010;
```

C++

```
WM_CLOSE = 0x0010;
```

File

Messages

Description

This is constant Messages.WM_CLOSE.

49.4.189 Messages.WM_COALESCE_FIRST Constant

Pascal

```
WM_COALESCE_FIRST = $0390;
```

C++

```
WM_COALESCE_FIRST = 0x0390;
```

File

Messages

Description

This is constant Messages.WM_COALESCE_FIRST.

49.4.190 Messages.WM_COALESCE_LAST Constant

Pascal

```
WM_COALESCE_LAST = $039F;
```

C++

```
WM_COALESCE_LAST = 0x039F;
```

File

Messages

Description

This is constant Messages.WM_COALESCE_LAST.

49.4.191 Messages.WM_COMMAND Constant

Pascal

```
WM_COMMAND = $0111;
```

C++

```
WM_COMMAND = 0x0111;
```

File

Messages

Description

This is constant Messages.WM_COMMAND.

49.4.192 Messages.WM_COMMNOTIFY Constant

Pascal

```
WM_COMMNOTIFY = $0044;
```

C++

```
WM_COMMNOTIFY = 0x0044;
```

File

Messages

Description

This is constant Messages.WM_COMMNOTIFY.

49.4.193 Messages.WM_COMPACTING Constant

Pascal

```
WM_COMPACTING = $0041;
```

C++

```
WM_COMPACTING = 0x0041;
```

File

Messages

Description

This is constant Messages.WM_COMPACTING.

49.4.194 Messages.WM_COMPAREITEM Constant

Pascal

```
WM_COMPAREITEM = $0039;
```

C++

```
WM_COMPAREITEM = 0x0039;
```

File

Messages

Description

This is constant Messages.WM_COMPAREITEM.

49.4.195 Messages.WM_CONTEXTMENU Constant

Pascal

```
WM_CONTEXTMENU = $007B;
```

C++

```
WM_CONTEXTMENU = 0x007B;
```

File

Messages

Description

This is constant Messages.WM_CONTEXTMENU.

49.4.196 Messages.WM_COPY Constant

Pascal

```
WM_COPY = $0301;
```

C++

```
WM_COPY = 0x0301;
```

File

Messages

Description

This is constant Messages.WM_COPY.

49.4.197 Messages.WM_COPYDATA Constant

Pascal

```
WM_COPYDATA = $004A;
```

C++

```
WM_COPYDATA = 0x004A;
```

File

Messages

Description

This is constant Messages.WM_COPYDATA.

49.4.198 Messages.WM_CREATE Constant

Pascal

```
WM_CREATE = $0001;
```

C++

```
WM_CREATE = 0x0001;
```

File

Messages

Description

This is constant Messages.WM_CREATE.

49.4.199 Messages.WM_CTLCOLOR Constant

Pascal

```
WM_CTLCOLOR = $0019;
```

C++

```
WM_CTLCOLOR = 0x0019;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLOR.

49.4.200 Messages.WM_CTLCOLORBTN Constant

Pascal

```
WM_CTLCOLORBTN = $0135;
```

C++

```
WM_CTLCOLORBTN = 0x0135;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORBTN.

49.4.201 Messages.WM_CTLCOLORDLG Constant

Pascal

```
WM_CTLCOLORDLG = $0136;
```

C++

```
WM_CTLCOLORDLG = 0x0136;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORDLG.

49.4.202 Messages.WM_CTLCOLOREDIT Constant

Pascal

```
WM_CTLCOLOREDIT = $0133;
```

C++

```
WM_CTLCOLOREDIT = 0x0133;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLOREDIT.

49.4.203 Messages.WM_CTLCOLORLISTBOX Constant

Pascal

```
WM_CTLCOLORLISTBOX = $0134;
```

C++

```
WM_CTLCOLORLISTBOX = 0x0134;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORLISTBOX.

49.4.204 Messages.WM_CTLCOLORMSGBOX Constant

Pascal

```
WM_CTLCOLORMSGBOX = $0132;
```

C++

```
WM_CTLCOLORMSGBOX = 0x0132;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORMSGBOX.

49.4.205 Messages.WM_CTLCOLORSCROLLBAR Constant

Pascal

```
WM_CTLCOLORSCROLLBAR = $0137;
```

C++

```
WM_CTLCOLORSCROLLBAR = 0x0137;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORSCROLLBAR.

49.4.206 Messages.WM_CTLCOLORSTATIC Constant

Pascal

```
WM_CTLCOLORSTATIC = $0138;
```

C++

```
WM_CTLCOLORSTATIC = 0x0138;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORSTATIC.

49.4.207 Messages.WM_CUT Constant

Pascal

```
WM_CUT = $0300;
```

C++

```
WM_CUT = 0x0300;
```

File

Messages

Description

This is constant Messages.WM_CUT.

49.4.208 Messages.WM_DDE_ACK Constant

Pascal

```
WM_DDE_ACK = WM_DDE_FIRST + 4;
```

C++

```
WM_DDE_ACK = WM_DDE_FIRST + 4;
```

File

Messages

Description

This is constant Messages.WM_DDE_ACK.

49.4.209 Messages.WM_DDE_ADVISE Constant

Pascal

```
WM_DDE_ADVISE = WM_DDE_FIRST + 2;
```

C++

```
WM_DDE_ADVISE = WM_DDE_FIRST + 2;
```

File

Messages

Description

This is constant Messages.WM_DDE_ADVISE.

49.4.210 Messages.WM_DDE_DATA Constant

Pascal

```
WM_DDE_DATA = WM_DDE_FIRST + 5;
```

C++

```
WM_DDE_DATA = WM_DDE_FIRST + 5;
```

File

Messages

Description

This is constant Messages.WM_DDE_DATA.

49.4.211 Messages.WM_DDE_EXECUTE Constant

Pascal

```
WM_DDE_EXECUTE = WM_DDE_FIRST + 8;
```

C++

```
WM_DDE_EXECUTE = WM_DDE_FIRST + 8;
```

File

Messages

Description

This is constant Messages.WM_DDE_EXECUTE.

49.4.212 Messages.WM_DDE_FIRST Constant

Pascal

```
WM_DDE_FIRST = $03E0;
```

C++

```
WM_DDE_FIRST = 0x03E0;
```

File

Messages

Description

This is constant Messages.WM_DDE_FIRST.

49.4.213 Messages.WM_DDE_INITIATE Constant

Pascal

```
WM_DDE_INITIATE = WM_DDE_FIRST + 0;
```

C++

```
WM_DDE_INITIATE = WM_DDE_FIRST + 0;
```

File

Messages

Description

This is constant Messages.WM_DDE_INITIATE.

49.4.214 Messages.WM_DDE_LAST Constant

Pascal

```
WM_DDE_LAST = WM_DDE_FIRST + 8;
```

C++

```
WM_DDE_LAST = WM_DDE_FIRST + 8;
```

File

Messages

Description

This is constant Messages.WM_DDE_LAST.

49.4.215 Messages.WM_DDE_POKE Constant

Pascal

```
WM_DDE_POKE = WM_DDE_FIRST + 7;
```

C++

```
WM_DDE_POKE = WM_DDE_FIRST + 7;
```

File

Messages

Description

This is constant Messages.WM_DDE_POKE.

49.4.216 Messages.WM_DDE_REQUEST Constant

Pascal

```
WM_DDE_REQUEST = WM_DDE_FIRST + 6;
```

C++

```
WM_DDE_REQUEST = WM_DDE_FIRST + 6;
```

File

Messages

Description

This is constant Messages.WM_DDE_REQUEST.

49.4.217 Messages.WM_DDE_TERMINATE Constant

Pascal

```
WM_DDE_TERMINATE = WM_DDE_FIRST + 1;
```

C++

```
WM_DDE_TERMINATE = WM_DDE_FIRST + 1;
```

File

Messages

Description

This is constant Messages.WM_DDE_TERMINATE.

49.4.218 Messages.WM_DDE_UNADVISE Constant

Pascal

```
WM_DDE_UNADVISE = WM_DDE_FIRST + 3;
```

C++

```
WM_DDE_UNADVISE = WM_DDE_FIRST + 3;
```

File

Messages

Description

This is constant Messages.WM_DDE_UNADVISE.

49.4.219 Messages.WM_DEADCHAR Constant

Pascal

```
WM_DEADCHAR = $0103;
```

C++

```
WM_DEADCHAR = 0x0103;
```

File

Messages

Description

This is constant Messages.WM_DEADCHAR.

49.4.220 Messages.WM_DELETEITEM Constant

Pascal

```
WM_DELETEITEM = $002D;
```

C++

```
WM_DELETEITEM = 0x002D;
```

File

Messages

Description

This is constant Messages.WM_DELETEITEM.

49.4.221 Messages.WM_DESTROY Constant

Pascal

```
WM_DESTROY = $0002;
```

C++

```
WM_DESTROY = 0x0002;
```

File

Messages

Description

This is constant Messages.WM_DESTROY.

49.4.222 Messages.WM_DESTROYCLIPBOARD Constant

Pascal

```
WM_DESTROYCLIPBOARD = $0307;
```

C++

```
WM_DESTROYCLIPBOARD = 0x0307;
```

File

Messages

Description

This is constant Messages.WM_DESTROYCLIPBOARD.

49.4.223 Messages.WM_DEVICECHANGE Constant

Pascal

```
WM_DEVICECHANGE = 537;
```

C++

```
WM_DEVICECHANGE = 537;
```

File

Messages

Description

This is constant Messages.WM_DEVICECHANGE.

49.4.224 Messages.WM_DEVMODECHANGE Constant

Pascal

```
WM_DEVMODECHANGE = $001B;
```

C++

```
WM_DEVMODECHANGE = 0x001B;
```

File

Messages

Description

This is constant Messages.WM_DEVMODECHANGE.

49.4.225 Messages.WM_DISPLAYCHANGE Constant

Pascal

```
WM_DISPLAYCHANGE = $007E;
```

C++

```
WM_DISPLAYCHANGE = 0x007E;
```

File

Messages

Description

This is constant Messages.WM_DISPLAYCHANGE.

49.4.226 Messages.WM_DRAWCLIPBOARD Constant

Pascal

```
WM_DRAWCLIPBOARD = $0308;
```

C++

```
WM_DRAWCLIPBOARD = 0x0308;
```

File

Messages

Description

This is constant Messages.WM_DRAWCLIPBOARD.

49.4.227 Messages.WM_DRAWITEM Constant

Pascal

```
WM_DRAWITEM = $002B;
```

C++

```
WM_DRAWITEM = 0x002B;
```

File

Messages

Description

This is constant Messages.WM_DRAWITEM.

49.4.228 Messages.WM_DROPFILES Constant

Pascal

```
WM_DROPFILES = $0233;
```

C++

```
WM_DROPFILES = 0x0233;
```

File

Messages

Description

This is constant Messages.WM_DROPFILES.

49.4.229 Messages.WM_DWMCOLORIZATIONCOLORCHANGED Constant

Pascal

```
WM_DWMCOLORIZATIONCOLORCHANGED = $0320;
```

C++

```
WM_DWMCOLORIZATIONCOLORCHANGED = 0x0320;
```

File

Messages

Description

This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.

49.4.230 Messages.WM_DWMCOMPOSITIONCHANGED Constant

Pascal

```
WM_DWMCOMPOSITIONCHANGED = $031E;
```

C++

```
WM_DWMCOMPOSITIONCHANGED = 0x031E;
```

File

Messages

Description

This is constant Messages.WM_DWMCOMPOSITIONCHANGED.

49.4.231 Messages.WM_DWMNCRENDERINGCHANGED Constant

Pascal

```
WM_DWMNCRENDERINGCHANGED = $031F;
```

C++

```
WM_DWMNCRENDERINGCHANGED = 0x031F;
```

File

Messages

Description

This is constant Messages.WM_DWMNCRENDERINGCHANGED.

49.4.232 Messages.WM_DWMWINDOWMAXIMIZEDCHANGE Constant

Pascal

```
WM_DWMWINDOWMAXIMIZEDCHANGE = $0321;
```

C++

```
WM_DWMWINDOWMAXIMIZEDCHANGE = 0x0321;
```

File

Messages

Description

This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.

49.4.233 Messages.WM_ENABLE Constant

Pascal

```
WM_ENABLE = $000A;
```

C++

```
WM_ENABLE = 0x000A;
```

File

Messages

Description

This is constant Messages.WM_ENABLE.

49.4.234 Messages.WM_ENDSESSION Constant

Pascal

```
WM_ENDSESSION = $0016;
```

C++

```
WM_ENDSESSION = 0x0016;
```

File

Messages

Description

This is constant Messages.WM_ENDSESSION.

49.4.235 Messages.WM_ENTERIDLE Constant

Pascal

```
WM_ENTERIDLE = $0121;
```

C++

```
WM_ENTERIDLE = 0x0121;
```

File

Messages

Description

This is constant Messages.WM_ENTERIDLE.

49.4.236 Messages.WM_ENTERMENULOOP Constant

Pascal

```
WM_ENTERMENULOOP = $0211;
```

C++

```
WM_ENTERMENULOOP = 0x0211;
```

File

Messages

Description

This is constant Messages.WM_ENTERMENULOOP.

49.4.237 Messages.WM_ENTERSIZEMOVE Constant

Pascal

```
WM_ENTERSIZEMOVE = $0231;
```

C++

```
WM_ENTERSIZEMOVE = 0x0231;
```

File

Messages

Description

This is constant Messages.WM_ENTERSIZEMOVE.

49.4.238 Messages.WM_ERASEBKGND Constant

Pascal

```
WM_ERASEBKGND = $0014;
```

C++

```
WM_ERASEBKGND = 0x0014;
```

File

Messages

Description

This is constant Messages.WM_ERASEBKGND.

49.4.239 Messages.WM_EXITMENULOOP Constant

Pascal

```
WM_EXITMENULOOP = $0212;
```

C++

```
WM_EXITMENULOOP = 0x0212;
```

File

Messages

Description

This is constant Messages.WM_EXITMENULOOP.

49.4.240 Messages.WM_EXITSIZEMOVE Constant

Pascal

```
WM_EXITSIZEMOVE = $0232;
```

C++

```
WM_EXITSIZEMOVE = 0x0232;
```

File

Messages

Description

This is constant Messages.WM_EXITSIZEMOVE.

49.4.241 Messages.WM_FONTCHANGE Constant

Pascal

```
WM_FONTCHANGE = $001D;
```

C++

```
WM_FONTCHANGE = 0x001D;
```

File

Messages

Description

This is constant Messages.WM_FONTCHANGE.

49.4.242 Messages.WM_GETDLGCODE Constant

Pascal

```
WM_GETDLGCODE = $0087;
```

C++

```
WM_GETDLGCODE = 0x0087;
```

File

Messages

Description

This is constant Messages.WM_GETDLGCODE.

49.4.243 Messages.WM_GETFONT Constant

Pascal

```
WM_GETFONT = $0031;
```

C++

```
WM_GETFONT = 0x0031;
```

File

Messages

Description

This is constant Messages.WM_GETFONT.

49.4.244 Messages.WM_GETHOTKEY Constant

Pascal

```
WM_GETHOTKEY = $0033;
```

C++

```
WM_GETHOTKEY = 0x0033;
```

File

Messages

Description

This is constant Messages.WM_GETHOTKEY.

49.4.245 Messages.WM_GETICON Constant

Pascal

```
WM_GETICON = $007F;
```

C++

```
WM_GETICON = 0x007F;
```

File

Messages

Description

This is constant Messages.WM_GETICON.

49.4.246 Messages.WM_GETMINMAXINFO Constant

Pascal

```
WM_GETMINMAXINFO = $0024;
```

C++

```
WM_GETMINMAXINFO = 0x0024;
```

File

Messages

Description

This is constant Messages.WM_GETMINMAXINFO.

49.4.247 Messages.WM_GETOBJECT Constant

Pascal

```
WM_GETOBJECT = $003D;
```

C++

```
WM_GETOBJECT = 0x003D;
```

File

Messages

Description

This is constant Messages.WM_GETOBJECT.

49.4.248 Messages.WM_GETTEXT Constant

Pascal

```
WM_GETTEXT = $000D;
```

C++

```
WM_GETTEXT = 0x000D;
```

File

Messages

Description

This is constant Messages.WM_GETTEXT.

49.4.249 Messages.WM_GETTEXTLENGTH Constant

Pascal

```
WM_GETTEXTLENGTH = $000E;
```

C++

```
WM_GETTEXTLENGTH = 0x000E;
```

File

Messages

Description

This is constant Messages.WM_GETTEXTLENGTH.

49.4.250 Messages.WM_HANDHELDFIRST Constant

Pascal

```
WM_HANDHELDFIRST = 856;
```

C++

```
WM_HANDHELDFIRST = 856;
```

File

Messages

Description

This is constant Messages.WM_HANDHELDFIRST.

49.4.251 Messages.WM_HANDHELDLAST Constant

Pascal

```
WM_HANDHELDLAST = 863;
```

C++

```
WM_HANDHELDLAST = 863;
```

File

Messages

Description

This is constant Messages.WM_HANDHELDLAST.

49.4.252 Messages.WM_HELP Constant

Pascal

```
WM_HELP = $0053;
```

C++

```
WM_HELP = 0x0053;
```

File

Messages

Description

This is constant Messages.WM_HELP.

49.4.253 Messages.WM_HOTKEY Constant

Pascal

```
WM_HOTKEY = $0312;
```

C++

```
WM_HOTKEY = 0x0312;
```

File

Messages

Description

This is constant Messages.WM_HOTKEY.

49.4.254 Messages.WM_HSCROLL Constant

Pascal

```
WM_HSCROLL = $0114;
```

C++

```
WM_HSCROLL = 0x0114;
```

File

Messages

Description

This is constant Messages.WM_HSCROLL.

49.4.255 Messages.WM_HSCROLLCLIPBOARD Constant

Pascal

```
WM_HSCROLLCLIPBOARD = $030E;
```

C++

```
WM_HSCROLLCLIPBOARD = 0x030E;
```

File

Messages

Description

This is constant Messages.WM_HSCROLLCLIPBOARD.

49.4.256 Messages.WM_ICONERASEBKGND Constant

Pascal

```
WM_ICONERASEBKGND = $0027;
```

C++

```
WM_ICONERASEBKGND = 0x0027;
```

File

Messages

Description

This is constant Messages.WM_ICONERASEBKGND.

49.4.257 Messages.WM_IME_CHAR Constant

Pascal

```
WM_IME_CHAR = $0286;
```

C++

```
WM_IME_CHAR = 0x0286;
```

File

Messages

Description

This is constant Messages.WM_IME_CHAR.

49.4.258 Messages.WM_IME_COMPOSITION Constant

Pascal

```
WM_IME_COMPOSITION = $010F;
```

C++

```
WM_IME_COMPOSITION = 0x010F;
```

File

Messages

Description

This is constant Messages.WM_IME_COMPOSITION.

49.4.259 Messages.WM_IME_COMPOSITIONFULL Constant

Pascal

```
WM_IME_COMPOSITIONFULL = $0284;
```

C++

```
WM_IME_COMPOSITIONFULL = 0x0284;
```

File

Messages

Description

This is constant Messages.WM_IME_COMPOSITIONFULL.

49.4.260 Messages.WM_IME_CONTROL Constant

Pascal

```
WM_IME_CONTROL = $0283;
```

C++

```
WM_IME_CONTROL = 0x0283;
```

File

Messages

Description

This is constant Messages.WM_IME_CONTROL.

49.4.261 Messages.WM_IME_ENDCOMPOSITION Constant

Pascal

```
WM_IME_ENDCOMPOSITION = $010E;
```

C++

```
WM_IME_ENDCOMPOSITION = 0x010E;
```

File

Messages

Description

This is constant Messages.WM_IME_ENDCOMPOSITION.

49.4.262 Messages.WM_IME_KEYDOWN Constant

Pascal

```
WM_IME_KEYDOWN = $0290;
```

C++

```
WM_IME_KEYDOWN = 0x0290;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYDOWN.

49.4.263 Messages.WM_IME_KEYLAST Constant

Pascal

```
WM_IME_KEYLAST = $010F;
```

C++

```
WM_IME_KEYLAST = 0x010F;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYLAST.

49.4.264 Messages.WM_IME_KEYUP Constant

Pascal

```
WM_IME_KEYUP = $0291;
```

C++

```
WM_IME_KEYUP = 0x0291;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYUP.

49.4.265 Messages.WM_IME_NOTIFY Constant

Pascal

```
WM_IME_NOTIFY = $0282;
```

C++

```
WM_IME_NOTIFY = 0x0282;
```

File

Messages

Description

This is constant Messages.WM_IME_NOTIFY.

49.4.266 Messages.WM_IME_REQUEST Constant

Pascal

```
WM_IME_REQUEST = $0288;
```

C++

```
WM_IME_REQUEST = 0x0288;
```

File

Messages

Description

This is constant Messages.WM_IME_REQUEST.

49.4.267 Messages.WM_IME_SELECT Constant

Pascal

```
WM_IME_SELECT = $0285;
```

C++

```
WM_IME_SELECT = 0x0285;
```

File

Messages

Description

This is constant Messages.WM_IME_SELECT.

49.4.268 Messages.WM_IME_SETCONTEXT Constant

Pascal

```
WM_IME_SETCONTEXT = $0281;
```

C++

```
WM_IME_SETCONTEXT = 0x0281;
```

File

Messages

Description

This is constant Messages.WM_IME_SETCONTEXT.

49.4.269 Messages.WM_IME_STARTCOMPOSITION Constant

Pascal

```
WM_IME_STARTCOMPOSITION = $010D;
```

C++

```
WM_IME_STARTCOMPOSITION = 0x010D;
```

File

Messages

Description

This is constant Messages.WM_IME_STARTCOMPOSITION.

49.4.270 Messages.WM_INITDIALOG Constant

Pascal

```
WM_INITDIALOG = $0110;
```

C++

```
WM_INITDIALOG = 0x0110;
```

File

Messages

Description

This is constant Messages.WM_INITDIALOG.

49.4.271 Messages.WM_INITMENU Constant

Pascal

```
WM_INITMENU = $0116;
```

C++

```
WM_INITMENU = 0x0116;
```

File

Messages

Description

This is constant Messages.WM_INITMENU.

49.4.272 Messages.WM_INITMENUPOPUP Constant

Pascal

```
WM_INITMENUPOPUP = $0117;
```

C++

```
WM_INITMENUPOPUP = 0x0117;
```

File

Messages

Description

This is constant Messages.WM_INITMENUPOPUP.

49.4.273 Messages.WM_INPUT Constant

Pascal

```
WM_INPUT = $00FF;
```

C++

```
WM_INPUT = 0x00FF;
```

File

Messages

Description

This is constant Messages.WM_INPUT.

49.4.274 Messages.WM_INPUTLANGCHANGE Constant

Pascal

```
WM_INPUTLANGCHANGE = $0051;
```

C++

```
WM_INPUTLANGCHANGE = 0x0051;
```

File

Messages

Description

This is constant Messages.WM_INPUTLANGCHANGE.

49.4.275 Messages.WM_INPUTLANGCHANGEREQUEST Constant

Pascal

```
WM_INPUTLANGCHANGEREQUEST = $0050;
```

C++

```
WM_INPUTLANGCHANGEREQUEST = 0x0050;
```

File

Messages

Description

This is constant Messages.WM_INPUTLANGCHANGEREQUEST.

49.4.276 Messages.WM_KEYDOWN Constant

Pascal

```
WM_KEYDOWN = $0100;
```

C++

```
WM_KEYDOWN = 0x0100;
```

File

Messages

Description

This is constant Messages.WM_KEYDOWN.

49.4.277 Messages.WM_KEYFIRST Constant

Pascal

```
WM_KEYFIRST = $0100;
```

C++

```
WM_KEYFIRST = 0x0100;
```

File

Messages

Description

This is constant Messages.WM_KEYFIRST.

49.4.278 Messages.WM_KEYLAST Constant

Pascal

```
WM_KEYLAST = $0109;
```

C++

```
WM_KEYLAST = 0x0109;
```

File

Messages

Description

This is constant Messages.WM_KEYLAST.

49.4.279 Messages.WM_KEYUP Constant

Pascal

```
WM_KEYUP = $0101;
```

C++

```
WM_KEYUP = 0x0101;
```

File

Messages

Description

This is constant Messages.WM_KEYUP.

49.4.280 Messages.WM_KILLFOCUS Constant

Pascal

```
WM_KILLFOCUS = $0008;
```

C++

```
WM_KILLFOCUS = 0x0008;
```

File

Messages

Description

This is constant Messages.WM_KILLFOCUS.

49.4.281 Messages.WM_LBUTTONDBLCLK Constant

Pascal

```
WM_LBUTTONDBLCLK = $0203;
```

C++

```
WM_LBUTTONDBLCLK = 0x0203;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONDBLCLK.

49.4.282 Messages.WM_LBUTTONDOWN Constant

Pascal

```
WM_LBUTTONDOWN = $0201;
```

C++

```
WM_LBUTTONDOWN = 0x0201;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONDOWN.

49.4.283 Messages.WM_LBUTTONUP Constant

Pascal

```
WM_LBUTTONUP = $0202;
```

C++

```
WM_LBUTTONUP = 0x0202;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONUP.

49.4.284 Messages.WM_MBUTTONDBLCLK Constant

Pascal

```
WM_MBUTTONDBLCLK = $0209;
```

C++

```
WM_MBUTTONDBLCLK = 0x0209;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONDBLCLK.

49.4.285 Messages.WM_MBUTTONDOWN Constant

Pascal

```
WM_MBUTTONDOWN = $0207;
```

C++

```
WM_MBUTTONDOWN = 0x0207;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONDOWN.

49.4.286 Messages.WM_MBUTTONUP Constant

Pascal

```
WM_MBUTTONUP = $0208;
```

C++

```
WM_MBUTTONUP = 0x0208;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONUP.

49.4.287 Messages.WM_MDIACTIVATE Constant

Pascal

```
WM_MDIACTIVATE = $0222;
```

C++

```
WM_MDIACTIVATE = 0x0222;
```

File

Messages

Description

This is constant Messages.WM_MDIACTIVATE.

49.4.288 Messages.WM_MDICASCADE Constant

Pascal

```
WM_MDICASCADE = $0227;
```

C++

```
WM_MDICASCADE = 0x0227;
```

File

Messages

Description

This is constant Messages.WM_MDICASCADE.

49.4.289 Messages.WM_MDICREATE Constant

Pascal

```
WM_MDICREATE = $0220;
```

C++

```
WM_MDICREATE = 0x0220;
```

File

Messages

Description

This is constant Messages.WM_MDICREATE.

49.4.290 Messages.WM_MDIDESTROY Constant

Pascal

```
WM_MDIDESTROY = $0221;
```

C++

```
WM_MDIDESTROY = 0x0221;
```

File

Messages

Description

This is constant Messages.WM_MDIDESTROY.

49.4.291 Messages.WM_MDIGETACTIVE Constant

Pascal

```
WM_MDIGETACTIVE = $0229;
```

C++

```
WM_MDIGETACTIVE = 0x0229;
```

File

Messages

Description

This is constant Messages.WM_MDIGETACTIVE.

49.4.292 Messages.WM_MDIICONARRANGE Constant

Pascal

```
WM_MDIICONARRANGE = $0228;
```

C++

```
WM_MDIICONARRANGE = 0x0228;
```

File

Messages

Description

This is constant Messages.WM_MDIICONARRANGE.

49.4.293 Messages.WM_MDIMAXIMIZE Constant

Pascal

```
WM_MDIMAXIMIZE = $0225;
```

C++

```
WM_MDIMAXIMIZE = 0x0225;
```

File

Messages

Description

This is constant Messages.WM_MDIMAXIMIZE.

49.4.294 Messages.WM_MDINEXT Constant

Pascal

```
WM_MDINEXT = $0224;
```

C++

```
WM_MDINEXT = 0x0224;
```

File

Messages

Description

This is constant Messages.WM_MDINEXT.

49.4.295 Messages.WM_MDIREFRESHMENU Constant

Pascal

```
WM_MDIREFRESHMENU = $0234;
```

C++

```
WM_MDIREFRESHMENU = 0x0234;
```

File

Messages

Description

This is constant Messages.WM_MDIREFRESHMENU.

49.4.296 Messages.WM_MDIRESTORE Constant

Pascal

```
WM_MDIRESTORE = $0223;
```

C++

```
WM_MDIRESTORE = 0x0223;
```

File

Messages

Description

This is constant Messages.WM_MDIRESTORE.

49.4.297 Messages.WM_MDISETMENU Constant

Pascal

```
WM_MDISETMENU = $0230;
```

C++

```
WM_MDISETMENU = 0x0230;
```

File

Messages

Description

This is constant Messages.WM_MDISETMENU.

49.4.298 Messages.WM_MDITILE Constant

Pascal

```
WM_MDITILE = $0226;
```

C++

```
WM_MDITILE = 0x0226;
```

File

Messages

Description

This is constant Messages.WM_MDITILE.

49.4.299 Messages.WM_MEASUREITEM Constant

Pascal

```
WM_MEASUREITEM = $002C;
```

C++

```
WM_MEASUREITEM = 0x002C;
```

File

Messages

Description

This is constant Messages.WM_MEASUREITEM.

49.4.300 Messages.WM_MENUCHAR Constant

Pascal

```
WM_MENUCHAR = $0120;
```

C++

```
WM_MENUCHAR = 0x0120;
```

File

Messages

Description

This is constant Messages.WM_MENUCHAR.

49.4.301 Messages.WM_MENUCOMMAND Constant

Pascal

```
WM_MENUCOMMAND = $0126;
```

C++

```
WM_MENUCOMMAND = 0x0126;
```

File

Messages

Description

This is constant Messages.WM_MENUCOMMAND.

49.4.302 Messages.WM_MENUDRAG Constant

Pascal

```
WM_MENUDRAG = $0123;
```

C++

```
WM_MENUDRAG = 0x0123;
```

File

Messages

Description

This is constant Messages.WM_MENUDRAG.

49.4.303 Messages.WM_MENUGETOBJECT Constant

Pascal

```
WM_MENUGETOBJECT = $0124;
```

C++

```
WM_MENUGETOBJECT = 0x0124;
```

File

Messages

Description

This is constant Messages.WM_MENUGETOBJECT.

49.4.304 Messages.WM_MENURBUTTONUP Constant

Pascal

```
WM_MENURBUTTONUP = $0122;
```

C++

```
WM_MENURBUTTONUP = 0x0122;
```

File

Messages

Description

This is constant Messages.WM_MENURBUTTONUP.

49.4.305 Messages.WM_MENUSELECT Constant

Pascal

```
WM_MENUSELECT = $011F;
```

C++

```
WM_MENUSELECT = 0x011F;
```

File

Messages

Description

This is constant Messages.WM_MENUSELECT.

49.4.306 Messages.WM_MOUSEACTIVATE Constant

Pascal

```
WM_MOUSEACTIVATE = $0021;
```

C++

```
WM_MOUSEACTIVATE = 0x0021;
```

File

Messages

Description

This is constant Messages.WM_MOUSEACTIVATE.

49.4.307 Messages.WM_MOUSEFIRST Constant

Pascal

```
WM_MOUSEFIRST = $0200;
```

C++

```
WM_MOUSEFIRST = 0x0200;
```

File

Messages

Description

This is constant Messages.WM_MOUSEFIRST.

49.4.308 Messages.WM_MOUSEHOVER Constant

Pascal

```
WM_MOUSEHOVER = $02A1;
```

C++

```
WM_MOUSEHOVER = 0x02A1;
```

File

Messages

Description

This is constant Messages.WM_MOUSEHOVER.

49.4.309 Messages.WM_MOUSELAST Constant

Pascal

```
WM_MOUSELAST = $020A;
```

C++

```
WM_MOUSELAST = 0x020A;
```

File

Messages

Description

This is constant Messages.WM_MOUSELAST.

49.4.310 Messages.WM_MOUSELEAVE Constant

Pascal

```
WM_MOUSELEAVE = $02A3;
```

C++

```
WM_MOUSELEAVE = 0x02A3;
```

File

Messages

Description

This is constant Messages.WM_MOUSELEAVE.

49.4.311 Messages.WM_MOUSEMOVE Constant

Pascal

```
WM_MOUSEMOVE = $0200;
```

C++

```
WM_MOUSEMOVE = 0x0200;
```

File

Messages

Description

This is constant Messages.WM_MOUSEMOVE.

49.4.312 Messages.WM_MOUSEWHEEL Constant

Pascal

```
WM_MOUSEWHEEL = $020A;
```

C++

```
WM_MOUSEWHEEL = 0x020A;
```

File

Messages

Description

This is constant Messages.WM_MOUSEWHEEL.

49.4.313 Messages.WM_MOVE Constant

Pascal

```
WM_MOVE = $0003;
```

C++

```
WM_MOVE = 0x0003;
```

File

Messages

Description

This is constant Messages.WM_MOVE.

49.4.314 Messages.WM_MOVING Constant

Pascal

```
WM_MOVING = 534;
```

C++

```
WM_MOVING = 534;
```

File

Messages

Description

This is constant Messages.WM_MOVING.

49.4.315 Messages.WM_NCACTIVATE Constant

Pascal

```
WM_NCACTIVATE = $0086;
```

C++

```
WM_NCACTIVATE = 0x0086;
```

File

Messages

Description

This is constant Messages.WM_NCACTIVATE.

49.4.316 Messages.WM_NCCALCSIZE Constant

Pascal

```
WM_NCCALCSIZE = $0083;
```

C++

```
WM_NCCALCSIZE = 0x0083;
```

File

Messages

Description

This is constant Messages.WM_NCCALCSIZE.

49.4.317 Messages.WM_NCCREATE Constant

Pascal

```
WM_NCCREATE = $0081;
```

C++

```
WM_NCCREATE = 0x0081;
```

File

Messages

Description

This is constant Messages.WM_NCCREATE.

49.4.318 Messages.WM_NCDESTROY Constant

Pascal

```
WM_NCDESTROY = $0082;
```

C++

```
WM_NCDESTROY = 0x0082;
```

File

Messages

Description

This is constant Messages.WM_NCDESTROY.

49.4.319 Messages.WM_NCHITTEST Constant

Pascal

```
WM_NCHITTEST = $0084;
```

C++

```
WM_NCHITTEST = 0x0084;
```

File

Messages

Description

This is constant Messages.WM_NCHITTEST.

49.4.320 Messages.WM_NCLBUTTONDBLCLK Constant

Pascal

```
WM_NCLBUTTONDBLCLK = $00A3;
```

C++

```
WM_NCLBUTTONDBLCLK = 0x00A3;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONDBLCLK.

49.4.321 Messages.WM_NCLBUTTONDOWN Constant

Pascal

```
WM_NCLBUTTONDOWN = $00A1;
```

C++

```
WM_NCLBUTTONDOWN = 0x00A1;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONDOWN.

49.4.322 Messages.WM_NCLBUTTONUP Constant

Pascal

```
WM_NCLBUTTONUP = $00A2;
```

C++

```
WM_NCLBUTTONUP = 0x00A2;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONUP.

49.4.323 Messages.WM_NCMBUTTONDBLCLK Constant

Pascal

```
WM_NCMBUTTONDBLCLK = $00A9;
```

C++

```
WM_NCMBUTTONDBLCLK = 0x00A9;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONDBLCLK.

49.4.324 Messages.WM_NCMBUTTONDOWN Constant

Pascal

```
WM_NCMBUTTONDOWN = $00A7;
```

C++

```
WM_NCMBUTTONDOWN = 0x00A7;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONDOWN.

49.4.325 Messages.WM_NCMBUTTONUP Constant

Pascal

```
WM_NCMBUTTONUP = $00A8;
```

C++

```
WM_NCMBUTTONUP = 0x00A8;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONUP.

49.4.326 Messages.WM_NCMOUSEHOVER Constant

Pascal

```
WM_NCMOUSEHOVER = $02A0;
```

C++

```
WM_NCMOUSEHOVER = 0x02A0;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSEHOVER.

49.4.327 Messages.WM_NCMOUSELEAVE Constant

Pascal

```
WM_NCMOUSELEAVE = $02A2;
```

C++

```
WM_NCMOUSELEAVE = 0x02A2;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSELEAVE.

49.4.328 Messages.WM_NCMOUSEMOVE Constant

Pascal

```
WM_NCMOUSEMOVE = $00A0;
```

C++

```
WM_NCMOUSEMOVE = 0x00A0;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSEMOVE.

49.4.329 Messages.WM_NCPAINT Constant

Pascal

```
WM_NCPAINT = $0085;
```

C++

```
WM_NCPAINT = 0x0085;
```

File

Messages

Description

This is constant Messages.WM_NCPAINT.

49.4.330 Messages.WM_NCRBUTTONDBLCLK Constant

Pascal

```
WM_NCRBUTTONDBLCLK = $00A6;
```

C++

```
WM_NCRBUTTONDBLCLK = 0x00A6;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONDBLCLK.

49.4.331 Messages.WM_NCRBUTTONDOWN Constant

Pascal

```
WM_NCRBUTTONDOWN = $00A4;
```

C++

```
WM_NCRBUTTONDOWN = 0x00A4;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONDOWN.

49.4.332 Messages.WM_NCRBUTTONUP Constant

Pascal

```
WM_NCRBUTTONUP = $00A5;
```

C++

```
WM_NCRBUTTONUP = 0x00A5;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONUP.

49.4.333 Messages.WM_NCXBUTTONDBLCLK Constant

Pascal

```
WM_NCXBUTTONDBLCLK = $00AD;
```

C++

```
WM_NCXBUTTONDBLCLK = 0x00AD;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONDBLCLK.

49.4.334 Messages.WM_NCXBUTTONDOWN Constant

Pascal

```
WM_NCXBUTTONDOWN = $00AB;
```

C++

```
WM_NCXBUTTONDOWN = 0x00AB;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONDOWN.

49.4.335 Messages.WM_NCXBUTTONUP Constant

Pascal

```
WM_NCXBUTTONUP = $00AC;
```

C++

```
WM_NCXBUTTONUP = 0x00AC;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONUP.

49.4.336 Messages.WM_NEXTDLGCTL Constant

Pascal

```
WM_NEXTDLGCTL = $0028;
```

C++

```
WM_NEXTDLGCTL = 0x0028;
```

File

Messages

Description

This is constant Messages.WM_NEXTDLGCTL.

49.4.337 Messages.WM_NEXTMENU Constant

Pascal

```
WM_NEXTMENU = $0213;
```

C++

```
WM_NEXTMENU = 0x0213;
```

File

Messages

Description

This is constant Messages.WM_NEXTMENU.

49.4.338 Messages.WM_NOTIFY Constant

Pascal

```
WM_NOTIFY = $004E;
```

C++

```
WM_NOTIFY = 0x004E;
```

File

Messages

Description

This is constant Messages.WM_NOTIFY.

49.4.339 Messages.WM_NOTIFYFORMAT Constant

Pascal

```
WM_NOTIFYFORMAT = $0055;
```

C++

```
WM_NOTIFYFORMAT = 0x0055;
```

File

Messages

Description

This is constant Messages.WM_NOTIFYFORMAT.

49.4.340 Messages.WM_NULL Constant

Pascal

```
WM_NULL = $0000;
```

C++

```
WM_NULL = 0x0000;
```

File

Messages

Description

This is constant Messages.WM_NULL.

49.4.341 Messages.WM_PAINT Constant

Pascal

```
WM_PAINT = $000F;
```

C++

```
WM_PAINT = 0x000F;
```

File

Messages

Description

This is constant Messages.WM_PAINT.

49.4.342 Messages.WM_PAINTCLIPBOARD Constant

Pascal

```
WM_PAINTCLIPBOARD = $0309;
```

C++

```
WM_PAINTCLIPBOARD = 0x0309;
```

File

Messages

Description

This is constant Messages.WM_PAINTCLIPBOARD.

49.4.343 Messages.WM_PAINTICON Constant

Pascal

```
WM_PAINTICON = $0026;
```

C++

```
WM_PAINTICON = 0x0026;
```

File

Messages

Description

This is constant Messages.WM_PAINTICON.

49.4.344 Messages.WM_PALETTECHANGED Constant

Pascal

```
WM_PALETTECHANGED = $0311;
```

C++

```
WM_PALETTECHANGED = 0x0311;
```

File

Messages

Description

This is constant Messages.WM_PALETTECHANGED.

49.4.345 Messages.WM_PALETTEISCHANGING Constant

Pascal

```
WM_PALETTEISCHANGING = $0310;
```

C++

```
WM_PALETTEISCHANGING = 0x0310;
```

File

Messages

Description

This is constant Messages.WM_PALETTEISCHANGING.

49.4.346 Messages.WM_PARENTNOTIFY Constant

Pascal

```
WM_PARENTNOTIFY = $0210;
```

C++

```
WM_PARENTNOTIFY = 0x0210;
```

File

Messages

Description

This is constant Messages.WM_PARENTNOTIFY.

49.4.347 Messages.WM_PASTE Constant

Pascal

```
WM_PASTE = $0302;
```

C++

```
WM_PASTE = 0x0302;
```

File

Messages

Description

This is constant Messages.WM_PASTE.

49.4.348 Messages.WM_PENWINFIRST Constant

Pascal

```
WM_PENWINFIRST = $0380;
```

C++

```
WM_PENWINFIRST = 0x0380;
```

File

Messages

Description

This is constant Messages.WM_PENWINFIRST.

49.4.349 Messages.WM_PENWINLAST Constant

Pascal

```
WM_PENWINLAST = $038F;
```

C++

```
WM_PENWINLAST = 0x038F;
```

File

Messages

Description

This is constant Messages.WM_PENWINLAST.

49.4.350 Messages.WM_POWER Constant

Pascal

```
WM_POWER = $0048;
```

C++

```
WM_POWER = 0x0048;
```

File

Messages

Description

This is constant Messages.WM_POWER.

49.4.351 Messages.WM_POWERBROADCAST Constant

Pascal

```
WM_POWERBROADCAST = 536;
```

C++

```
WM_POWERBROADCAST = 536;
```

File

Messages

Description

This is constant Messages.WM_POWERBROADCAST.

49.4.352 Messages.WM_PRINT Constant

Pascal

```
WM_PRINT = 791;
```

C++

```
WM_PRINT = 791;
```

File

Messages

Description

This is constant Messages.WM_PRINT.

49.4.353 Messages.WM_PRINTCLIENT Constant

Pascal

```
WM_PRINTCLIENT = 792;
```

C++

```
WM_PRINTCLIENT = 792;
```

File

Messages

Description

This is constant Messages.WM_PRINTCLIENT.

49.4.354 Messages.WM_QUERYDRAGICON Constant

Pascal

```
WM_QUERYDRAGICON = $0037;
```

C++

```
WM_QUERYDRAGICON = 0x0037;
```

File

Messages

Description

This is constant Messages.WM_QUERYDRAGICON.

49.4.355 Messages.WM_QUERYENDSESSION Constant

Pascal

```
WM_QUERYENDSESSION = $0011;
```

C++

```
WM_QUERYENDSESSION = 0x0011;
```

File

Messages

Description

This is constant Messages.WM_QUERYENDSESSION.

49.4.356 Messages.WM_QUERYNEWPALETTE Constant

Pascal

```
WM_QUERYNEWPALETTE = $030F;
```

C++

```
WM_QUERYNEWPALETTE = 0x030F;
```

File

Messages

Description

This is constant Messages.WM_QUERYNEWPALETTE.

49.4.357 Messages.WM_QUERYOPEN Constant

Pascal

```
WM_QUERYOPEN = $0013;
```

C++

```
WM_QUERYOPEN = 0x0013;
```

File

Messages

Description

This is constant Messages.WM_QUERYOPEN.

49.4.358 Messages.WM_QUERYUISTATE Constant

Pascal

```
WM_QUERYUISTATE = $0129;
```

C++

```
WM_QUERYUISTATE = 0x0129;
```

File

Messages

Description

This is constant Messages.WM_QUERYUISTATE.

49.4.359 Messages.WM_QUEUESYNC Constant

Pascal

```
WM_QUEUESYNC = $0023;
```

C++

```
WM_QUEUESYNC = 0x0023;
```

File

Messages

Description

This is constant Messages.WM_QUEUESYNC.

49.4.360 Messages.WM_QUIT Constant

Pascal

```
WM_QUIT = $0012;
```

C++

```
WM_QUIT = 0x0012;
```

File

Messages

Description

This is constant Messages.WM_QUIT.

49.4.361 Messages.WM_RBUTTONDBLCLK Constant

Pascal

```
WM_RBUTTONDBLCLK = $0206;
```

C++

```
WM_RBUTTONDBLCLK = 0x0206;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONDBLCLK.

49.4.362 Messages.WM_RBUTTONDOWN Constant

Pascal

```
WM_RBUTTONDOWN = $0204;
```

C++

```
WM_RBUTTONDOWN = 0x0204;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONDOWN.

49.4.363 Messages.WM_RBUTTONUP Constant

Pascal

```
WM_RBUTTONUP = $0205;
```

C++

```
WM_RBUTTONUP = 0x0205;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONUP.

49.4.364 Messages.WM_RENDERALLFORMATS Constant

Pascal

```
WM_RENDERALLFORMATS = $0306;
```

C++

```
WM_RENDERALLFORMATS = 0x0306;
```

File

Messages

Description

This is constant Messages.WM_RENDERALLFORMATS.

49.4.365 Messages.WM_RENDERFORMAT Constant

Pascal

```
WM_RENDERFORMAT = $0305;
```

C++

```
WM_RENDERFORMAT = 0x0305;
```

File

Messages

Description

This is constant Messages.WM_RENDERFORMAT.

49.4.366 Messages.WM_SETCURSOR Constant

Pascal

```
WM_SETCURSOR = $0020;
```

C++

```
WM_SETCURSOR = 0x0020;
```

File

Messages

Description

This is constant Messages.WM_SETCURSOR.

49.4.367 Messages.WM_SETFOCUS Constant

Pascal

```
WM_SETFOCUS = $0007;
```

C++

```
WM_SETFOCUS = 0x0007;
```

File

Messages

Description

This is constant Messages.WM_SETFOCUS.

49.4.368 Messages.WM_SETFONT Constant

Pascal

```
WM_SETFONT = $0030;
```

C++

```
WM_SETFONT = 0x0030;
```

File

Messages

Description

This is constant Messages.WM_SETFONT.

49.4.369 Messages.WM_SETHOTKEY Constant

Pascal

```
WM_SETHOTKEY = $0032;
```

C++

```
WM_SETHOTKEY = 0x0032;
```

File

Messages

Description

This is constant Messages.WM_SETHOTKEY.

49.4.370 Messages.WM_SETICON Constant

Pascal

```
WM_SETICON = $0080;
```

C++

```
WM_SETICON = 0x0080;
```

File

Messages

Description

This is constant Messages.WM_SETICON.

49.4.371 Messages.WM_SETREDRAW Constant

Pascal

```
WM_SETREDRAW = $000B;
```

C++

```
WM_SETREDRAW = 0x000B;
```

File

Messages

Description

This is constant Messages.WM_SETREDRAW.

49.4.372 Messages.WM_SETTEXT Constant

Pascal

```
WM_SETTEXT = $000C;
```

C++

```
WM_SETTEXT = 0x000C;
```

File

Messages

Description

This is constant Messages.WM_SETTEXT.

49.4.373 Messages.WM_SETTINGCHANGE Constant

Pascal

```
WM_SETTINGCHANGE = WM_WININICHANGE;
```

C++

```
WM_SETTINGCHANGE = WM_WININICHANGE;
```

File

Messages

Description

This is constant Messages.WM_SETTINGCHANGE.

49.4.374 Messages.WM_SHOWWINDOW Constant

Pascal

```
WM_SHOWWINDOW = $0018;
```

C++

```
WM_SHOWWINDOW = 0x0018;
```

File

Messages

Description

This is constant Messages.WM_SHOWWINDOW.

49.4.375 Messages.WM_SIZE Constant

Pascal

```
WM_SIZE = $0005;
```

C++

```
WM_SIZE = 0x0005;
```

File

Messages

Description

This is constant Messages.WM_SIZE.

49.4.376 Messages.WM_SIZECLIPBOARD Constant

Pascal

```
WM_SIZECLIPBOARD = $030B;
```

C++

```
WM_SIZECLIPBOARD = 0x030B;
```

File

Messages

Description

This is constant Messages.WM_SIZECLIPBOARD.

49.4.377 Messages.WM_SIZING Constant

Pascal

```
WM_SIZING = 532;
```

C++

```
WM_SIZING = 532;
```

File

Messages

Description

This is constant Messages.WM_SIZING.

49.4.378 Messages.WM_SPOOLERSTATUS Constant

Pascal

```
WM_SPOOLERSTATUS = $002A;
```

C++

```
WM_SPOOLERSTATUS = 0x002A;
```

File

Messages

Description

This is constant Messages.WM_SPOOLERSTATUS.

49.4.379 Messages.WM_STYLECHANGED Constant

Pascal

```
WM_STYLECHANGED = $007D;
```

C++

```
WM_STYLECHANGED = 0x007D;
```

File

Messages

Description

This is constant Messages.WM_STYLECHANGED.

49.4.380 Messages.WM_STYLECHANGING Constant

Pascal

```
WM_STYLECHANGING = $007C;
```

C++

```
WM_STYLECHANGING = 0x007C;
```

File

Messages

Description

This is constant Messages.WM_STYLECHANGING.

49.4.381 Messages.WM_SYSCHAR Constant

Pascal

```
WM_SYSCHAR = $0106;
```

C++

```
WM_SYSCHAR = 0x0106;
```

File

Messages

Description

This is constant Messages.WM_SYSCHAR.

49.4.382 Messages.WM_SYSCOLORCHANGE Constant

Pascal

```
WM_SYSCOLORCHANGE = $0015;
```

C++

```
WM_SYSCOLORCHANGE = 0x0015;
```

File

Messages

Description

This is constant Messages.WM_SYSCOLORCHANGE.

49.4.383 Messages.WM_SYSCOMMAND Constant

Pascal

```
WM_SYSCOMMAND = $0112;
```

C++

```
WM_SYSCOMMAND = 0x0112;
```

File

Messages

Description

This is constant Messages.WM_SYSCOMMAND.

49.4.384 Messages.WM_SYSDEADCHAR Constant

Pascal

```
WM_SYSDEADCHAR = $0107;
```

C++

```
WM_SYSDEADCHAR = 0x0107;
```

File

Messages

Description

This is constant Messages.WM_SYSDEADCHAR.

49.4.385 Messages.WM_SYSKEYDOWN Constant

Pascal

```
WM_SYSKEYDOWN = $0104;
```

C++

```
WM_SYSKEYDOWN = 0x0104;
```

File

Messages

Description

This is constant Messages.WM_SYSKEYDOWN.

49.4.386 Messages.WM_SYSKEYUP Constant

Pascal

```
WM_SYSKEYUP = $0105;
```

C++

```
WM_SYSKEYUP = 0x0105;
```

File

Messages

Description

This is constant Messages.WM_SYSKEYUP.

49.4.387 Messages.WM_SYSTEMERROR Constant

Pascal

```
WM_SYSTEMERROR = $0017;
```

C++

```
WM_SYSTEMERROR = 0x0017;
```

File

Messages

Description

This is constant Messages.WM_SYSTEMERROR.

49.4.388 Messages.WM_TABLET_FIRST Constant

Pascal

```
WM_TABLET_FIRST = $02C0;
```

C++

```
WM_TABLET_FIRST = 0x02C0;
```

File

Messages

Description

This is constant Messages.WM_TABLET_FIRST.

49.4.389 Messages.WM_TABLET_LAST Constant

Pascal

```
WM_TABLET_LAST = $02DF;
```

C++

```
WM_TABLET_LAST = 0x02DF;
```

File

Messages

Description

This is constant Messages.WM_TABLET_LAST.

49.4.390 Messages.WM_TCARD Constant

Pascal

```
WM_TCARD = $0052;
```

C++

```
WM_TCARD = 0x0052;
```

File

Messages

Description

This is constant Messages.WM_TCARD.

49.4.391 Messages.WM_THEMECHANGED Constant

Pascal

```
WM_THEMECHANGED = $031A;
```

C++

```
WM_THEMECHANGED = 0x031A;
```

File

Messages

Description

This is constant Messages.WM_THEMECHANGED.

49.4.392 Messages.WM_TIMECHANGE Constant

Pascal

```
WM_TIMECHANGE = $001E;
```

C++

```
WM_TIMECHANGE = 0x001E;
```

File

Messages

Description

This is constant Messages.WM_TIMECHANGE.

49.4.393 Messages.WM_TIMER Constant

Pascal

```
WM_TIMER = $0113;
```

C++

```
WM_TIMER = 0x0113;
```

File

Messages

Description

This is constant Messages.WM_TIMER.

49.4.394 Messages.WM_UNDO Constant

Pascal

```
WM_UNDO = $0304;
```

C++

```
WM_UNDO = 0x0304;
```

File

Messages

Description

This is constant Messages.WM_UNDO.

49.4.395 Messages.WM_UNICHAR Constant

Pascal

```
WM_UNICHAR = $0109;
```

C++

```
WM_UNICHAR = 0x0109;
```

File

Messages

Description

This is constant Messages.WM_UNICHAR.

49.4.396 Messages.WM_UNINITMENUPOPUP Constant

Pascal

```
WM_UNINITMENUPOPUP = $0125;
```

C++

```
WM_UNINITMENUPOPUP = 0x0125;
```

File

Messages

Description

This is constant Messages.WM_UNINITMENUPOPUP.

49.4.397 Messages.WM_UPDATEUISTATE Constant

Pascal

```
WM_UPDATEUISTATE = $0128;
```

C++

```
WM_UPDATEUISTATE = 0x0128;
```

File

Messages

Description

This is constant Messages.WM_UPDATEUISTATE.

49.4.398 Messages.WM_USER Constant

Pascal

```
WM_USER = $0400;
```

C++

```
WM_USER = 0x0400;
```

File

Messages

Description

This is constant Messages.WM_USER.

49.4.399 Messages.WM_USERCHANGED Constant

Pascal

```
WM_USERCHANGED = $0054;
```

C++

```
WM_USERCHANGED = 0x0054;
```

File

Messages

Description

This is constant Messages.WM_USERCHANGED.

49.4.400 Messages.WM_VKEYTOITEM Constant

Pascal

```
WM_VKEYTOITEM = $002E;
```

C++

```
WM_VKEYTOITEM = 0x002E;
```

File

Messages

Description

This is constant Messages.WM_VKEYTOITEM.

49.4.401 Messages.WM_VSCROLL Constant

Pascal

```
WM_VSCROLL = $0115;
```

C++

```
WM_VSCROLL = 0x0115;
```

File

Messages

Description

This is constant Messages.WM_VSCROLL.

49.4.402 Messages.WM_VSCROLLCLIPBOARD Constant

Pascal

```
WM_VSCROLLCLIPBOARD = $030A;
```

C++

```
WM_VSCROLLCLIPBOARD = 0x030A;
```

File

Messages

Description

This is constant Messages.WM_VSCROLLCLIPBOARD.

49.4.403 Messages.WM_WINDOWPOSCHANGED Constant

Pascal

```
WM_WINDOWPOSCHANGED = $0047;
```

C++

```
WM_WINDOWPOSCHANGED = 0x0047;
```

File

Messages

Description

This is constant Messages.WM_WINDOWPOSCHANGED.

49.4.404 Messages.WM_WINDOWPOSCHANGING Constant

Pascal

```
WM_WINDOWPOSCHANGING = $0046;
```

C++

```
WM_WINDOWPOSCHANGING = 0x0046;
```

File

Messages

Description

This is constant Messages.WM_WINDOWPOSCHANGING.

49.4.405 Messages.WM_WININICHANGE Constant

Pascal

```
WM_WININICHANGE = $001A;
```

C++

```
WM_WININICHANGE = 0x001A;
```

File

Messages

Description

This is constant Messages.WM_WININICHANGE.

49.4.406 Messages.WM_WTSSESSION_CHANGE Constant

Pascal

```
WM_WTSSESSION_CHANGE = $02B1;
```

C++

```
WM_WTSSESSION_CHANGE = 0x02B1;
```

File

Messages

Description

This is constant Messages.WM_WTSSESSION_CHANGE.

50 Midas Namespace

This is namespace Midas.

Topics

Name	Description
Interfaces (see page 5222)	The following table lists interfaces in this documentation.
Constants (see page 5232)	The following table lists constants in this documentation.

Constants

Name	Description
IID_IAppServer (see page 5232)	This is constant Midas.IID_IAppServer.
LIBID_Midas (see page 5232)	This is constant Midas.LIBID_Midas.

Interfaces

	Name	Description
	IAppServer (see page 5222)	IAppServer is the interface that client datasets use to communicate with a dataset provider.
	IAppServerDisp (see page 5229)	This is class Midas.IAppServerDisp.

50.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAppServer (see page 5222)	IAppServer is the interface that client datasets use to communicate with a dataset provider.
	IAppServerDisp (see page 5229)	This is class Midas.IAppServerDisp.

50.1.1 IAppServer Interface

IAppServer is the interface that client datasets use to communicate with a dataset provider.

Class Hierarchy

Pascal

```
IAppServer = interface(IDispatch);
```

C++

```
__interface IAppServer : public IDispatch;
```

File

Midas

Description

Client datasets use an IAppServer interface to implement many of their properties and methods. This interface provides the basis of communication between the client dataset and a provider component from which it receives data and to which it applies updates.

If the provider is in the same application as the client dataset, the IAppServer interface is implemented by a hidden object that is created to manage the communication between the client dataset and the provider. In multi-tiered applications, client datasets obtain an IAppServer instance from a connection component that resides in the client application.

When client datasets use IAppServer to talk to a provider component, they pass the name of the desired provider to all method calls. By default, the interface is stateless, meaning that each method call functions independently, with no reliance on previous method calls. Because of this, IAppServer has no properties, which would represent persistent state information. However, there are cases where the client dataset relies on state information, such as the current database cursor position when performing incremental fetching using the AS_GetRecords (see page 5227) method. This state information is preserved only if multiple clients do not share an application server that implements IAppServer.

Note: Component Method declarations that use IAppServer use the _di_IAppServer type instead. This type is a DelphiInterface wrapper around the IAppServer interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IAppServer >  _di_IAppServer;
```

See Also

TClientDataSet

TDataSetProvider (see page 6510)

TCustomClientDataSet

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

50.1.1.1 IAppServer Members

The following tables list the members exposed by IAppServer.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 5223)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 5224)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 5224)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 5225)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 5226)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 5227)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 5228)	Returns information from a specified record of the provider's dataset.

50.1.1.2 IAppServer Methods

The methods of the IAppServer class are listed here.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 5223)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 5224)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 5224)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 5225)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 5226)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 5227)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 5228)	Returns information from a specified record of the provider's dataset.

50.1.1.2.1 IAppServer.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates

applyupdates (see page 6468)

BeforeApplyUpdates (see page 6487)

AfterApplyUpdates (see page 6483)

BeforeApplyUpdates

AfterApplyUpdates

AS_ApplyUpdates

as_applyupdates

Communicating with the Client Dataset

50.1.1.2.2 IAppServer.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest (see page 6490)

DataRequest

Communicating with the Client Dataset

50.1.1.2.3 IAppServer.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 6525)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

Params

Execute

Execute (see page 6474)

BeforeExecute

AfterExecute

Options (see page 6455)

PackageParams

UnpackParams

Communicating with the Client Dataset

50.1.1.2.4 IAppServer.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute (see page 5224) method instead.

See Also

DataSet (see page 6525)

Params

FetchParams

GetParams (see page 6475)

BeforeGetParams (see page 6488)

aftergetparams (see page 6485)

AS_Execute (see page 5224)

UnpackParams

Communicating with the Client Dataset

50.1.1.2.5 IAppServer.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that uses the IAppServer (see page 5222) interface of an application server. When calling any other IAppServer (see page 5222) method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

providers

ProviderName

Providers

Communicating with the Client Dataset

50.1.1.2.6 IAppServer.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data

Params

GetNextPacket

GetRecords (see page 6476)

BeforeGetRecords

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

Options (see page 6455)

PackageParams

UnpackParams

Communicating with the Client Dataset

50.1.1.2.7 IAppServer.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest (see page 6480)

BeforeRowRequest (see page 6490)

AfterRowRequest (see page 6486)

FetchBlobs

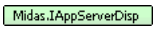
FetchDetails

RefreshRecord

Communicating with the Client Dataset

50.1.2 IAppServerDisp Interface

Class Hierarchy



Pascal

```
IAppServerDisp = dispinterface;
```

C++

```
class IAppServerDisp;
```

File

Midas

Description

This is class Midas.IAppServerDisp.

50.1.2.1 IAppServerDisp Members

The following tables list the members exposed by IAppServerDisp.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 5230)	This is AS_ApplyUpdates, a member of class IAppServerDisp.
	AS_DataRequest (see page 5230)	This is AS_DataRequest, a member of class IAppServerDisp.
	AS_Execute (see page 5230)	This is AS_Execute, a member of class IAppServerDisp.
	AS_GetParams (see page 5230)	This is AS_GetParams, a member of class IAppServerDisp.
	AS_GetProviderNames (see page 5231)	This is AS_GetProviderNames, a member of class IAppServerDisp.
	AS_GetRecords (see page 5231)	This is AS_GetRecords, a member of class IAppServerDisp.
	AS_RowRequest (see page 5231)	This is AS_RowRequest, a member of class IAppServerDisp.

50.1.2.2 IAppServerDisp Methods

The methods of the IAppServerDisp class are listed here.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 5230)	This is AS_ApplyUpdates, a member of class IAppServerDisp.
	AS_DataRequest (see page 5230)	This is AS_DataRequest, a member of class IAppServerDisp.
	AS_Execute (see page 5230)	This is AS_Execute, a member of class IAppServerDisp.
	AS_GetParams (see page 5230)	This is AS_GetParams, a member of class IAppServerDisp.
	AS_GetProviderNames (see page 5231)	This is AS_GetProviderNames, a member of class IAppServerDisp.
	AS_GetRecords (see page 5231)	This is AS_GetRecords, a member of class IAppServerDisp.
	AS_RowRequest (see page 5231)	This is AS_RowRequest, a member of class IAppServerDisp.

50.1.2.2.1 IAppServerDisp.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is AS_ApplyUpdates, a member of class IAppServerDisp.

50.1.2.2.2 IAppServerDisp.AS_DataRequest Method

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

This is AS_DataRequest, a member of class IAppServerDisp.

50.1.2.2.3 IAppServerDisp.AS_Execute Method

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant);
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

This is AS_Execute, a member of class IAppServerDisp.

50.1.2.2.4 IAppServerDisp.AS_GetParams Method

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

This is AS_GetParams, a member of class IAppServerDisp.

50.1.2.2.5 IAppServerDisp.AS_GetProviderNames Method

Pascal

```
function AS_GetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

This is AS_GetProviderNames, a member of class IAppServerDisp.

50.1.2.2.6 IAppServerDisp.AS_GetRecords Method

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class IAppServerDisp.

50.1.2.2.7 IAppServerDisp.AS_RowRequest Method

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

This is AS_RowRequest, a member of class IAppServerDisp.

50.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IID_IAppServer (see page 5232)	This is constant Midas.IID_IAppServer.
LIBID_Midas (see page 5232)	This is constant Midas.LIBID_Midas.

50.2.1 Midas.IID_IAppServer Constant

Pascal

```
IID_IAppServer: TGUID = '{1AEFCC20-7A24-11D2-98B0-C69BEB4B5B6D}';
```

C++

```
TGUID IID_IAppServer = '{1AEFCC20-7A24-11D2-98B0-C69BEB4B5B6D}';
```

File

Midas

Description

This is constant Midas.IID_IAppServer.

50.2.2 Midas.LIBID_Midas Constant

Pascal

```
LIBID_Midas: TGUID = '{83F57D68-CA9A-11D2-9088-00C04FA35CFA}';
```

C++

```
TGUID LIBID_Midas = '{83F57D68-CA9A-11D2-9088-00C04FA35CFA}';
```

File

Midas

Description

This is constant Midas.LIBID_Midas.

51 MidasCon Namespace

This is namespace MidasCon.

Topics

Name	Description
Classes (see page 5234)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 5241)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TMidasConnection (see page 5234)	TMidasConnection handles a server connection for the client of a multi-tiered application.
	TRemoteServer (see page 5239)	TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

Structs, Records, Enums

	Name	Description
	TConnectType (see page 5241)	This is record MidasCon.TConnectType.

51.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TMidasConnection (see page 5234)	TMidasConnection handles a server connection for the client of a multi-tiered application.
	TRemoteServer (see page 5239)	TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

51.1.1 TMidasConnection Class

TMidasConnection handles a server connection for the client of a multi-tiered application.

Class Hierarchy

Pascal

```
TMidasConnection = class(TDCOMConnection);
```

C++

```
class TMidasConnection : public TDCOMConnection;
```

File

MidasCon

Description

TMidasConnection is provided for backward compatibility. When writing client applications that connect to application servers using DCOM, use TDCOMConnection (see page 4790) instead. When writing client applications that connect to application servers using sockets, use TSocketConnection instead. When writing client applications that connect to application servers using HTTP, use TWebConnection instead.

See Also

TClientDataSet

TDCOMConnection (see page 4790)

TSocketConnection

TWebConnection

51.1.1.1 TMidasConnection Members

The following tables list the members exposed by TMidasConnection.

Protected Methods

	Name	Description
	DoConnect (see page 5236)	Connects to an application server.
	DoDisconnect (see page 5237)	Drops a connection to an application server.
	GetConnected (see page 4798)	Returns the value of the Connected (see page 4802) property.
	GetServerCLSID (see page 4799)	Returns the GUID for the application server.
	Notification (see page 4800)	Responds when other objects are created or destroyed.
	SetConnected (see page 4786)	Sets the Connected (see page 4802) property.
	SetConnected (see page 4800)	Sets the Connected (see page 4802) property.
	SetConnectType (see page 5237)	This is SetConnectType, a member of class TMidasConnection.
	SetUseBroker (see page 5237)	This is SetUseBroker, a member of class TMidasConnection.

Protected Properties

	Name	Description
	ObjectBroker (see page 4806)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Public Methods

	Name	Description
	Create (see page 5235)	Creates and initializes an instance of TMidasConnection (see page 5234).
	GetServer (see page 4799)	Returns the IAppServer (see page 5222) interface of the application server.
	GetServerList (see page 4800)	Returns a list of remote data modules to which the component can connect.

Published Events

	Name	Description
	AfterConnect (see page 4808)	Occurs after a connection is established.
	AfterDisconnect (see page 4808)	Occurs after the connection closes.
	BeforeConnect (see page 4809)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 4809)	Occurs immediately before the connection closes.
	OnGetUsername (see page 4810)	This is OnGetUsername, a member of class TDispatchConnection.
	OnLogin (see page 4810)	Occurs when a communication channel opens to the server.

Published Properties

	Name	Description
	ComputerName (see page 4793)	This is ComputerName, a member of class TDCOMConnection.
	Connected (see page 4802)	Indicates whether the TDispatchConnection (see page 4796) object is connected to an application server.
	ConnectType (see page 5238)	Specifies the protocol to use to establish the connection to the application server.
	LoginPrompt (see page 4802)	Specifies whether a login dialog appears immediately before opening a new connection.
	ObjectBroker (see page 4794)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	ServerGUID (see page 4807)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 4807)	Specifies the name of the application server with which to connect.
	ServerPort (see page 5238)	Specifies the port used by the listening socket on the server machine.
	UseBroker (see page 5239)	Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

51.1.1.2 TMidasConnection.Create Constructor

Creates and initializes an instance of TMidasConnection (see page 5234).

Pascal

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

C++

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

Description

Call Create to instantiate a TMidasConnection (see page 5234) component at runtime. TMidasConnection (see page 5234) components placed in forms or data modules at design time are created automatically.

Create initializes the value of the ServerPort (see page 5238) property to 211. This is also the default port number for ScktSrvr.exe and ScktSrvc.exe.

See Also

ServerPort (see page 5238)

Create (see page 4797)

Port

51.1.1.3 TMidasConnection Methods

The methods of the TMidasConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 5236)	Connects to an application server.
	DoDisconnect (see page 5237)	Drops a connection to an application server.
	SetConnectType (see page 5237)	This is SetConnectType, a member of class TMidasConnection.
	SetUseBroker (see page 5237)	This is SetUseBroker, a member of class TMidasConnection.

51.1.1.3.1 TMidasConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected property is set to true. DoConnect establishes a dual interface with the remote data module on the application server. This interface is used to locate providers on the application server.

To locate the server system, DoConnect uses the ComputerName property. If the ObjectBroker property is set, DoConnect gets this value from the object broker. Otherwise, the application must assign this property a value directly.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

computername (see page 4793)

ObjectBroker (see page 4806)

51.1.1.3.2 TMidasConnection.DoDisconnect Method

Drops a connection to an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected property is set to false. It drops the connection and changes the AppServer property to a NULL Variant.

See Also

AfterDisconnect

BeforeDisconnect

Connected (see page 4802)

AppServer

51.1.1.3.3 TMidasConnection.SetConnectType Method

Pascal

```
procedure SetConnectType(Value: TConnectType); virtual;
```

C++

```
virtual __fastcall SetConnectType(TConnectType Value);
```

Description

This is SetConnectType, a member of class TMidasConnection.

51.1.1.3.4 TMidasConnection.SetUseBroker Method

Pascal

```
procedure SetUseBroker(Value: Boolean); virtual;
```

C++

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

Description

This is SetUseBroker, a member of class TMidasConnection.

51.1.1.4 TMidasConnection Properties

The properties of the TMidasConnection class are listed here.

Published Properties

	Name	Description
	ConnectType (see page 5238)	Specifies the protocol to use to establish the connection to the application server.
	ServerPort (see page 5238)	Specifies the port used by the listening socket on the server machine.
	UseBroker (see page 5239)	Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

51.1.1.4.1 TMidasConnection.ConnectType Property

Specifies the protocol to use to establish the connection to the application server.

Pascal

```
property ConnectType: TConnectType;
```

C++

```
__property TConnectType ConnectType;
```

Description

Use ConnectType to specify the desired protocol for connecting with the application server.

See Also

ServerPort (see page 5238)

computername (see page 4793)

UseBroker (see page 5239)

ServerName (see page 4807)

51.1.1.4.2 TMidasConnection.ServerPort Property

Specifies the port used by the listening socket on the server machine.

Pascal

```
property ServerPort: Integer;
```

C++

```
__property int ServerPort;
```

Description

When establishing connections to the application server using TCP/IP sockets, the application server must have a socket dispatcher listening on a predefined port for requests from client applications. If the value of ConnectType (see page 5238) is ctSockets, set ServerPort to the port number used by Scktsrvr.exe or SckTSrvc.exe on the server system. TMidasConnection (see page 5234) attempts to connect to the application server by sending a message to the system specified by the ComputerName property at the port specified by ServerPort.

Note: ServerPort has no effect if ConnectType (see page 5238) is not ctSockets.

See Also

ConnectType (see page 5238)

computername (see page 4793)

Port

51.1.1.4.3 TMidasConnection.UseBroker Property

Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

Pascal

```
property UseBroker: Boolean;
```

C++

```
__property Boolean UseBroker;
```

Description

When establishing connections to the application server using OLEnterprise, UseBroker specifies whether the connection should use the Business Object Broker. If UseBroker is true, TMidasConnection (see page 5234) uses the Business Object Broker, and the ComputerName property specifies the name of the Business Object Broker. If UseBroker is false, TMidasConnection (see page 5234) uses OLEnterprise, but does not go through the Business Object Broker. When UseBroker is false, ComputerName specifies the name of the OLEnterprise server.

For more information about using OLEnterprise, see the OLEnterprise documentation.

Note: The value of UseBroker has no effect unless ConnectType (see page 5238) is ctOLEnterprise.

See Also

ConnectType (see page 5238)

computername (see page 4793)

51.1.2 TRemoteServer Class

TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

Class Hierarchy

Pascal

```
TRemoteServer = class(TDCOMConnection);
```

C++

```
class TRemoteServer : public TDCOMConnection;
```

File

MidasCon

Description

TRemoteServer is provided for backward compatibility. When writing client applications that connect to application servers using DCOM, use TDCOMConnection (see page 4790) instead.

See Also

TClientDataSet

TDCOMConnection (see page 4790)

TSocketConnection

TWebConnection

TDatasetprovider

51.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TConnectType (see page 5241)	This is record MidasCon.TConnectType.

51.2.1 MidasCon.TConnectType Enumeration

Pascal

```
TConnectType = (
  ctDCOM,
  ctSockets,
  ctOLEnterprise
);
```

C++

```
enum TConnectType {
  ctDCOM,
  ctSockets,
  ctOLEnterprise
};
```

File

MidasCon

Description

This is record MidasCon.TConnectType.

52 MidItems Namespace

This is namespace MidItems.

Topics

Name	Description
Classes (see page 5248)	The following table lists classes in this documentation.
Interfaces (see page 5644)	The following table lists interfaces in this documentation.
Functions (see page 5676)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5681)	The following table lists structs, records, enums in this documentation.
Types (see page 5684)	The following table lists types in this documentation.
Constants (see page 5686)	The following table lists constants in this documentation.

Classes

	Name	Description
	TApplyUpdatesButton (see page 5250)	TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.
	TCaptionAttributes (see page 5256)	TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.
	TCustomDataForm (see page 5260)	TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.
	TCustomDataGrid (see page 5263)	TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.
	TCustomDataNavigator (see page 5271)	TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.
	TCustomFieldGroup (see page 5276)	TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.
	TCustomLayoutGroup (see page 5282)	TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.
	TCustomQueryButtons (see page 5288)	TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.
	TCustomQueryForm (see page 5293)	TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.
	TDataForm (see page 5298)	TDataForm generates an HTML form that displays database information and lets users apply updates.
	TDataGrid (see page 5302)	TDataGrid generates an HTML table to display a dataset in an XML data packet.
	TDataNavigator (see page 5307)	TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.
	TDataSetButton (see page 5311)	TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.

	TDeleteButton (see page 5318)	TDeleteButton generates an HTML control that deletes a record from an XML data packet.
	TFieldGroup (see page 5320)	TFieldGroup arranges database information on an HTML form as a set of labeled fields.
	TFieldRadioGroup (see page 5326)	TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.
	TFieldSelectOptions (see page 5335)	TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.
	TFieldStatus (see page 5344)	TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.
	TFieldText (see page 5349)	TFieldText creates a single-line HTML input control to represent a field in an XML data packet.
	TFieldTextArea (see page 5355)	TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.
	TFirstButton (see page 5363)	TFirstButton generates an HTML control that moves to the first record in an XML data packet.
	TFormLayout (see page 5365)	TFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TGridAttributes (see page 5373)	TGridAttributes describes the options the TDataGrid (see page 5302) component adds to an HTML <TABLE> tag.
	TGridRowAttributes (see page 5379)	TGridRowAttributes describes the options the TDataGrid (see page 5302) component adds to its HTML row tags (<TR>).
	TInsertButton (see page 5385)	TInsertButton generates an HTML control that inserts a new record in an XML data packet.
	TLastButton (see page 5387)	TLastButton generates an HTML control that moves to the last record in an XML data packet.
	TLayoutGroup (see page 5389)	TLayoutGroup generates the HTML to arrange content produced by other Web items.
	TNextButton (see page 5392)	TNextButton generates an HTML control that moves to the next record in an XML data packet.
	TNextPageButton (see page 5394)	TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.
	TPostButton (see page 5396)	TPostButton generates an HTML control that posts changes to an XML delta packet.
	TPriorButton (see page 5399)	TPriorButton generates an HTML control that moves to the previous record in an XML data packet.
	TPriorPageButton (see page 5401)	TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.
	TQueryButton (see page 5403)	TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.
	TQueryButtons (see page 5412)	TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.
	TQueryFieldGroup (see page 5415)	TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.
	TQueryForm (see page 5421)	TQueryForm generates HTML forms for letting users enter application-defined values.
	TQueryRadioGroup (see page 5425)	TQueryRadioGroup creates an HTML radio group control to represent named parameter.
	TQuerySelectOptions (see page 5437)	TQuerySelectOptions creates an HTML list control to represent a named parameter.
	TQueryText (see page 5449)	TQueryText creates a single-line HTML input control to represent a named parameter.
	TQueryTextArea (see page 5458)	TQueryTextArea creates a multi-line HTML input control to represent named parameter.
	TResetQueryButton (see page 5469)	TResetQueryButton generates an HTML control that backs out changes to the values on a query form.
	TSelectOptionsColumn (see page 5472)	TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.
	TStatusColumn (see page 5481)	TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.
	TSubmitQueryButton (see page 5486)	TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.
	TTextAreaColumn (see page 5488)	TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.

	TTextColumn (see page 5495)	TTextColumn creates a single-line HTML input control to represent a field value in a data grid.
	TUndoButton (see page 5502)	TUndoButton generates an HTML control that backs out changes to the current record.
	TWebButton (see page 5504)	TWebButton is the base class for Web items that create HTML button controls.
	TWebControlGroup (see page 5510)	TWebControlGroup is the base class for Web items that arrange HTML controls.
	TWebDataDisplay (see page 5519)	TWebDataDisplay is the base class for Web items that create HTML display controls.
	TWebDataInput (see page 5532)	TWebDataInput is the base class for Web items that create HTML display controls for entering information.
	TWebDisplay (see page 5540)	This is class MidItems.TWebDisplay.
	TWebDisplayReferenceGroup (see page 5544)	TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.
	TWebForm (see page 5548)	TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.
	TWebListInput (see page 5558)	TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.
	TWebRadioGroupInput (see page 5567)	TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.
	TWebSelectOptionsInput (see page 5573)	TWebSelectOptionsInput is the base class for Web items that create HTML list controls.
	TWebStatus (see page 5578)	TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.
	TWebTextAreaInput (see page 5583)	TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.
	TWebTextInput (see page 5589)	TWebTextInput is the base class for Web items that create HTML single-line text controls.
	TXMLButton (see page 5595)	TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.
	TXMLData (see page 5601)	TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.
	TXMLDataParent (see page 5605)	This is class MidItems.TXMLDataParent.
	TXMLDataSet (see page 5608)	TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.
	TXMLDataSetParent (see page 5616)	This is class MidItems.TXMLDataSetParent.
	TXMLDisplayGroup (see page 5620)	TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.
	TXMLDisplayParent (see page 5632)	This is class MidItems.TXMLDisplayParent.
	TXMLDisplayReferenceButton (see page 5635)	TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.
	TXMLDisplayReferenceGroup (see page 5640)	TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

Constants

Name	Description
FormMethodNames (see page 5686)	This is constant MidItems.FormMethodNames.
HTMLAlignValues (see page 5687)	This is constant MidItems.HTMLAlignValues.
HTMLVAlignValues (see page 5687)	This is constant MidItems.HTMLVAlignValues.
sActionAttr (see page 5687)	This is constant MidItems.sActionAttr.
sAlignAttr (see page 5688)	This is constant MidItems.sAlignAttr.
sBgColorAttr (see page 5688)	This is constant MidItems.sBgColorAttr.
sBorderAttr (see page 5688)	This is constant MidItems.sBorderAttr.
sCellPaddingAttr (see page 5689)	This is constant MidItems.sCellPaddingAttr.
sCellSpacingAttr (see page 5689)	This is constant MidItems.sCellSpacingAttr.
sCheckedAttr (see page 5689)	This is constant MidItems.sCheckedAttr.
sClassAttr (see page 5690)	This is constant MidItems.sClassAttr.
sColsAttr (see page 5690)	This is constant MidItems.sColsAttr.
sColSpanAttr (see page 5690)	This is constant MidItems.sColSpanAttr.
sEncTypeAttr (see page 5691)	This is constant MidItems.sEncTypeAttr.
sImageHeightAttr (see page 5691)	This is constant MidItems.sImageHeightAttr.

sImageWidthAttr (see page 5691)	This is constant MidItems.sImageWidthAttr.
sMaxLengthAttr (see page 5692)	This is constant MidItems.sMaxLengthAttr.
sMethodAttr (see page 5692)	This is constant MidItems.sMethodAttr.
sMultipleAttr (see page 5692)	This is constant MidItems.sMultipleAttr.
sNameAttr (see page 5693)	This is constant MidItems.sNameAttr.
sOnClickAttr (see page 5693)	This is constant MidItems.sOnClickAttr.
sReadOnlyAttr (see page 5693)	This is constant MidItems.sReadOnlyAttr.
sRowsAttr (see page 5694)	This is constant MidItems.sRowsAttr.
sSelectedAttr (see page 5694)	This is constant MidItems.sSelectedAttr.
sSizeAttr (see page 5694)	This is constant MidItems.sSizeAttr.
sStyleAttr (see page 5695)	This is constant MidItems.sStyleAttr.
sTabIndexAttr (see page 5695)	This is constant MidItems.sTabIndexAttr.
sTypeAttr (see page 5695)	This is constant MidItems.sTypeAttr.
sVAlignAttr (see page 5696)	This is constant MidItems.sVAlignAttr.
sValueAttr (see page 5696)	This is constant MidItems.sValueAttr.
sWidthAttr (see page 5696)	This is constant MidItems.sWidthAttr.
sWrapAttr (see page 5697)	This is constant MidItems.sWrapAttr.
TextAreaWrap (see page 5697)	This is constant MidItems.TextAreaWrap.

Functions

	Name	Description
	AddBoolAttrib (see page 5676)	This is function MidItems.AddBoolAttrib.
	AddCustomAttrib (see page 5676)	This is function MidItems.AddCustomAttrib.
	AddIntAttrib (see page 5677)	This is function MidItems.AddIntAttrib.
	AddQuotedAttrib (see page 5677)	This is function MidItems.AddQuotedAttrib.
	AddStringAttrib (see page 5677)	This is function MidItems.AddStringAttrib.
	CanAddClassHelper (see page 5678)	This is function MidItems.CanAddClassHelper.
	CreateDefaultButtonClasses (see page 5678)	This is function MidItems.CreateDefaultButtonClasses.
	DeclareSubmitForm (see page 5678)	This is function MidItems.DeclareSubmitForm.
	FindDispatcher (see page 5679)	This is function MidItems.FindDispatcher.
	FindProducer (see page 5679)	This is function MidItems.FindProducer.
	FormatColumnData (see page 5679)	This is function MidItems.FormatColumnData.
	FormatColumnHeading (see page 5680)	This is function MidItems.FormatColumnHeading.
	GetItemValuesFromDataSet (see page 5680)	This is function MidItems.GetItemValuesFromDataSet.

Interfaces

	Name	Description
	IDataSetComponent (see page 5644)	IDataSetComponent defines the methods that bind a Web item to an XML broker.
	IDataSetField (see page 5648)	This is class MidItems.IDataSetField.
	IDataSetFields (see page 5650)	This is class MidItems.IDataSetFields.
	IFormatColumn (see page 5652)	This is class MidItems.IFormatColumn.
	IHTMLField (see page 5653)	IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.
	IHTMLForm (see page 5655)	IHTMLForm is an interface that must be implemented by Web items that generate forms.
	IInetXWebDataSet (see page 5657)	IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.
	IQueryField (see page 5661)	This is class MidItems.IQueryField.
	IQueryFields (see page 5664)	This is class MidItems.IQueryFields.
	IQueryForm (see page 5665)	This is class MidItems.IQueryForm.
	IRestoreDefaults (see page 5665)	This is class MidItems.IRestoreDefaults.
	IStatusField (see page 5666)	This is class MidItems.IStatusField.
	IValidateField (see page 5667)	This is class MidItems.IValidateField.
	IValidateFields (see page 5668)	This is class MidItems.IValidateFields.
	IWebDisplay (see page 5670)	

	IWebDisplayReference (see page 5671)	
	IXMLDisplay (see page 5673)	This is class MidItems.IXMLDisplay.

Structs, Records, Enums

	Name	Description
	TCaptionPosition (see page 5681)	TCaptionPosition indicates the relative position of a label to the HTML control it labels.
	TFormMethod (see page 5682)	TFormMethod represents the method attribute on the <form> tag of an HTML form.
	TLayoutState (see page 5682)	This is record MidItems.TLayoutState.
	TTextAreaWrap (see page 5683)	TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

Types

Name	Description
IMidasWebDataSet (see page 5684)	This is type MidItems.IMidasWebDataSet.
TWebButtonClass (see page 5684)	This is type MidItems.TWebButtonClass.
TWebDataDisplayClass (see page 5685)	This is type MidItems.TWebDataDisplayClass.
TWebDataInputClass (see page 5685)	This is type MidItems.TWebDataInputClass.
TWebStatusClass (see page 5685)	This is type MidItems.TWebStatusClass.

52.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TApplyUpdatesButton (see page 5250)	TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.
	TCaptionAttributes (see page 5256)	TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.
	TCustomDataForm (see page 5260)	TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.
	TCustomDataGrid (see page 5263)	TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.
	TCustomDataNavigator (see page 5271)	TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.
	TCustomFieldGroup (see page 5276)	TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.
	TCustomLayoutGroup (see page 5282)	TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.
	TCustomQueryButtons (see page 5288)	TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.
	TCustomQueryForm (see page 5293)	TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.
	TDataForm (see page 5298)	TDataForm generates an HTML form that displays database information and lets users apply updates.
	TDataGrid (see page 5302)	TDataGrid generates an HTML table to display a dataset in an XML data packet.
	TDataNavigator (see page 5307)	TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.
	TDataSetButton (see page 5311)	TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.
	TDeleteButton (see page 5318)	TDeleteButton generates an HTML control that deletes a record from an XML data packet.
	TFieldGroup (see page 5320)	TFieldGroup arranges database information on an HTML form as a set of labeled fields.
	TFieldRadioGroup (see page 5326)	TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.
	TFieldSelectOptions (see page 5335)	TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.
	TFieldStatus (see page 5344)	TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.
	TFieldText (see page 5349)	TFieldText creates a single-line HTML input control to represent a field in an XML data packet.
	TFieldTextArea (see page 5355)	TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.
	TFirstButton (see page 5363)	TFirstButton generates an HTML control that moves to the first record in an XML data packet.
	TFormLayout (see page 5365)	TFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TGridAttributes (see page 5373)	TGridAttributes describes the options the TDataGrid (see page 5302) component adds to an HTML <TABLE> tag.
	TGridRowAttributes (see page 5379)	TGridRowAttributes describes the options the TDataGrid (see page 5302) component adds to its HTML row tags (<TR>).
	TInsertButton (see page 5385)	TInsertButton generates an HTML control that inserts a new record in an XML data packet.
	TLastButton (see page 5387)	TLastButton generates an HTML control that moves to the last record in an XML data packet.

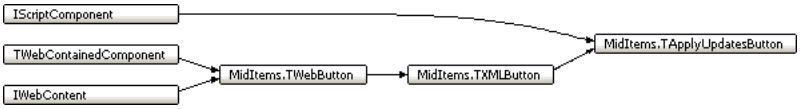
	TLayoutGroup (see page 5389)	TLayoutGroup generates the HTML to arrange content produced by other Web items.
	TNextButton (see page 5392)	TNextButton generates an HTML control that moves to the next record in an XML data packet.
	TNextPageButton (see page 5394)	TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.
	TPostButton (see page 5396)	TPostButton generates an HTML control that posts changes to an XML delta packet.
	TPriorButton (see page 5399)	TPriorButton generates an HTML control that moves to the previous record in an XML data packet.
	TPriorPageButton (see page 5401)	TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.
	TQueryButton (see page 5403)	TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.
	TQueryButtons (see page 5412)	TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.
	TQueryFieldGroup (see page 5415)	TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.
	TQueryForm (see page 5421)	TQueryForm generates HTML forms for letting users enter application-defined values.
	TQueryRadioGroup (see page 5425)	TQueryRadioGroup creates an HTML radio group control to represent named parameter.
	TQuerySelectOptions (see page 5437)	TQuerySelectOptions creates an HTML list control to represent a named parameter.
	TQueryText (see page 5449)	TQueryText creates a single-line HTML input control to represent a named parameter.
	TQueryTextArea (see page 5458)	TQueryTextArea creates a multi-line HTML input control to represent named parameter.
	TResetQueryButton (see page 5469)	TResetQueryButton generates an HTML control that backs out changes to the values on a query form.
	TSelectOptionsColumn (see page 5472)	TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.
	TStatusColumn (see page 5481)	TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.
	TSubmitQueryButton (see page 5486)	TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.
	TTextAreaColumn (see page 5488)	TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.
	TTextColumn (see page 5495)	TTextColumn creates a single-line HTML input control to represent a field value in a data grid.
	TUndoButton (see page 5502)	TUndoButton generates an HTML control that backs out changes to the current record.
	TWebButton (see page 5504)	TWebButton is the base class for Web items that create HTML button controls.
	TWebControlGroup (see page 5510)	TWebControlGroup is the base class for Web items that arrange HTML controls.
	TWebDataDisplay (see page 5519)	TWebDataDisplay is the base class for Web items that create HTML display controls.
	TWebDataInput (see page 5532)	TWebDataInput is the base class for Web items that create HTML display controls for entering information.
	TWebDisplay (see page 5540)	This is class MidItems.TWebDisplay.
	TWebDisplayReferenceGroup (see page 5544)	TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.
	TWebForm (see page 5548)	TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.
	TWebListInput (see page 5558)	TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.
	TWebRadioGroupInput (see page 5567)	TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.
	TWebSelectOptionsInput (see page 5573)	TWebSelectOptionsInput is the base class for Web items that create HTML list controls.
	TWebStatus (see page 5578)	TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.
	TWebTextAreaInput (see page 5583)	TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.

	TWebTextInput (see page 5589)	TWebTextInput is the base class for Web items that create HTML single-line text controls.
	TXMLButton (see page 5595)	TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.
	TXMLData (see page 5601)	TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.
	TXMLDataParent (see page 5605)	This is class MidItems.TXMLDataParent.
	TXMLDataSet (see page 5608)	TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.
	TXMLDataSetParent (see page 5616)	This is class MidItems.TXMLDataSetParent.
	TXMLDisplayGroup (see page 5620)	TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.
	TXMLDisplayParent (see page 5632)	This is class MidItems.TXMLDisplayParent.
	TXMLDisplayReferenceButton (see page 5635)	TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.
	TXMLDisplayReferenceGroup (see page 5640)	TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

52.1.1 TApplyUpdatesButton Class

TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.

Class Hierarchy



Pascal

```
TApplyUpdatesButton = class(TXMLButton, IScriptComponent);
```

C++

```
class TApplyUpdatesButton : public TXMLButton, public IScriptComponent;
```

File

MidItems

Description

Add TApplyUpdatesButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to submit changes made by end users. The resulting HTML button's action generates an HTTP message that forwards all changes to a specific XML data packet back to the XML broker on the InternetExpress application.

Use the properties of TApplyUpdatesButton to customize the appearance of the button and associate it with an XML broker. The associated XML broker adds the XML data packet to the HTML document and responds to update messages generated by the apply updates button on the HTML form.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

Using the Web Page Editor

52.1.1.1 TApplyUpdatesButton Members

The following tables list the members exposed by TApplyUpdatesButton.

Protected Methods

	Name	Description
	AddElements (see page 5252)	This is AddElements, a member of class TApplyUpdatesButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
V	GetCaption (see page 5597)	Returns the value of the Caption property.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetSubComponents (see page 5252)	This is GetSubComponents, a member of class TApplyUpdatesButton.
	GetXMLBroker (see page 5598)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 5598)	This is GetXMLUseParent, a member of class TXMLButton.
V	ImplAddElements (see page 5252)	This is ImplAddElements, a member of class TApplyUpdatesButton.
V	ImplContent (see page 5252)	Provides the underlying implementation for the Content method.
V	Notification (see page 5598)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetXMLBroker (see page 5599)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 5599)	This is SetXMLUseParent, a member of class TXMLButton.

Protected Properties

	Name	Description
	DefaultCaption (see page 5599)	This is DefaultCaption, a member of class TXMLButton.
R	XMLData (see page 5600)	This is XMLData, a member of class TXMLButton.

Public Methods

	Name	Description
V	Create (see page 5251)	Creates and initializes an instance of TApplyUpdatesButton (see page 5250).
V	Destroy (see page 5597)	Frees an instance of TXMLButton (see page 5595).

Published Properties

	Name	Description
	Caption (see page 5253)	Specifies the caption that appears on the button.
	Custom (see page 5254)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5254)	Adds style attributes to the generated button control.
	StyleRule (see page 5254)	Applies a style in the page producer's style sheet to the button.
	XMLBroker (see page 5255)	Associates the XML button with an XML broker.
	XMLUseParent (see page 5255)	Specifies whether the XML button is associated with its parent's XML broker.

52.1.1.2 TApplyUpdatesButton.Create Constructor

Creates and initializes an instance of TApplyUpdatesButton (see page 5250).

Pascal

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

C++

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

Description

Call Create to instantiate TApplyUpdatesButton (see page 5250) at runtime. Buttons added at design time using the Web

Page editor are instantiated automatically.

52.1.1.3 TApplyUpdatesButton Methods

The methods of the TApplyUpdatesButton class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 5252)	This is AddElements, a member of class TApplyUpdatesButton.
	GetSubComponents (see page 5252)	This is GetSubComponents, a member of class TApplyUpdatesButton.
V	ImplAddElements (see page 5252)	This is ImplAddElements, a member of class TApplyUpdatesButton.
V	ImplContent (see page 5252)	Provides the underlying implementation for the Content method.

52.1.1.3.1 TApplyUpdatesButton.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TApplyUpdatesButton.

52.1.1.3.2 TApplyUpdatesButton.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TApplyUpdatesButton.

52.1.1.3.3 TApplyUpdatesButton.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TApplyUpdatesButton.

52.1.1.3.4 TApplyUpdatesButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 5506)

ILayoutWebContent

52.1.1.4 TApplyUpdatesButton Properties

The properties of the TApplyUpdatesButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 5253)	Specifies the caption that appears on the button.
	Custom (see page 5254)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5254)	Adds style attributes to the generated button control.
	StyleRule (see page 5254)	Applies a style in the page producer's style sheet to the button.
	XMLBroker (see page 5255)	Associates the XML button with an XML broker.
	XMLUseParent (see page 5255)	Specifies whether the XML button is associated with its parent's XML broker.

52.1.1.4.1 TApplyUpdatesButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 5509)

GetCaption (see page 5507)

SetCaption (see page 5508)

52.1.1.4.2 TApplyUpdatesButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 5508)

Style (see page 5509)

52.1.1.4.3 TApplyUpdatesButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 5509)

StyleRule (see page 5510)

52.1.1.4.4 TApplyUpdatesButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 5509)

Styles (see page 5716)

StylesFile (see page 5717)

Styles

StylesFile

52.1.1.4.5 TApplyUpdatesButton.XMLBroker Property

Associates the XML button with an XML broker.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Set XMLBroker to specify the XML broker whose data packets the XML button manipulates.

When XML buttons are used by TDataNavigator, the data navigator already establishes a connection (indirectly) to an XML broker. If the XML button uses this same XML broker, you do not need to set the XMLBroker property. Instead, setting the XMLUseParent property to true (the default) automatically associates the XML button with the same broker as the one associated with the navigator.

See Also

XMLUseParent (see page 5600)

52.1.1.4.6 TApplyUpdatesButton.XMLUseParent Property

Specifies whether the XML button is associated with its parent's XML broker.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the XML button determines the XML broker whose data packets it manipulates. When XMLUseParent is false, the XMLBroker property must be set to indicate the XML broker. When XMLUseParent is true, the XML broker associated with the XML button's parent determines the associated XML broker.

The XML button's parent is the Web item (typically TDataNavigator) that uses the XML button to assemble its own HTML content. The parent's associated broker may be defined through an indirect association. For example, the data navigator acts on another component, such as a field group, which has an associated broker.

If the parent does not have an associated XML broker, setting XMLUseParent to true sets the XMLBroker property to nil (Delphi) or NULL (C++).

See Also

XMLBroker (see page 5599)

52.1.2 TCaptionAttributes Class

TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.

Class Hierarchy

Pascal

```
TCaptionAttributes = class(TPersistent);
```

C++

```
class TCaptionAttributes : public TPersistent;
```

File

MidItems

Description

TCaptionAttributes is the type of the CaptionAttributes property of Web items that generate display controls. It defines several properties that customize the appearance of the label generated by the Web item.

See Also

CaptionAttributes

52.1.2.1 TCaptionAttributes Members

The following tables list the members exposed by TCaptionAttributes.

Public Methods

	Name	Description
	Create (see page 5257)	Creates an instance of TCaptionAttributes (see page 5256).

Public Properties

	Name	Description
R	Parent (see page 5258)	Indicates the Web item that uses TCaptionAttributes (see page 5256) to customize the appearance of its generated caption.

Published Properties

	Name	Description
	Align (see page 5257)	Represents the Align option of an HTML tag.
	Custom (see page 5258)	Represents a tag option and its value.

	Style (see page 5258)	Adds style attributes to the generated caption.
	StyleRule (see page 5259)	Applies a style in the page producer's style sheet to the generated caption.
	VAlign (see page 5259)	Represents the VAlign option of an HTML tag.

52.1.2.2 TCaptionAttributes.Create Constructor

Creates an instance of TCaptionAttributes (see page 5256).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TCaptionAttributes(TComponent * AParent);
```

Description

Do not create instances of TCaptionAttributes (see page 5256). This object is instantiated by the Web item that uses it to implement its CaptionAttributes property.

The AParent parameter specifies the Web item that creates a TCaptionAttributes (see page 5256) instance for its CaptionAttributes property. This becomes the value of the Parent (see page 5258) property.

See Also

CaptionAttributes (see page 5530)

Parent (see page 5258)

52.1.2.3 TCaptionAttributes Properties

The properties of the TCaptionAttributes class are listed here.

Public Properties

	Name	Description
R	Parent (see page 5258)	Indicates the Web item that uses TCaptionAttributes (see page 5256) to customize the appearance of its generated caption.

Published Properties

	Name	Description
	Align (see page 5257)	Represents the Align option of an HTML tag.
	Custom (see page 5258)	Represents a tag option and its value.
	Style (see page 5258)	Adds style attributes to the generated caption.
	StyleRule (see page 5259)	Applies a style in the page producer's style sheet to the generated caption.
	VAlign (see page 5259)	Represents the VAlign option of an HTML tag.

52.1.2.3.1 TCaptionAttributes.Align Property

Represents the Align option of an HTML tag.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Use Align to add an Align option to the tag that surrounds the caption. The Align option influences the horizontal alignment of the caption text.

The Web item that generates the caption inserts the specified Align option into the HTML tag when it generates its content.

See Also

VAlign (see page 5259)

52.1.2.3.2 TCaptionAttributes.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the caption. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

The Web item that generates the caption inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 5258)

StyleRule (see page 5259)

52.1.2.3.3 TCaptionAttributes.Parent Property

Indicates the Web item that uses TCaptionAttributes (see page 5256) to customize the appearance of its generated caption.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Use Parent to access the Web item that uses the TCaptionAttributes (see page 5256) object.

See Also

Create (see page 5257)

52.1.2.3.4 TCaptionAttributes.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated caption. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the caption as the value of a STYLE option.

See Also

Custom (see page 5258)

StyleRule (see page 5259)

52.1.2.3.5 TCaptionAttributes.StyleRule Property

Applies a style in the page producer's style sheet to the generated caption.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated caption. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 5258) property instead.

See Also

Style (see page 5258)

Styles (see page 5716)

Styles

StylesFile (see page 5717)

StylesFile

52.1.2.3.6 TCaptionAttributes.VAlign Property

Represents the VAlign option of an HTML tag.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Use VAlign to add an VAlign option to the tag that surrounds the caption. The VAlign option influences the vertical alignment of the caption text.

The Web item that generates the caption inserts the specified VAlign option into the HTML tag when it generates its content.

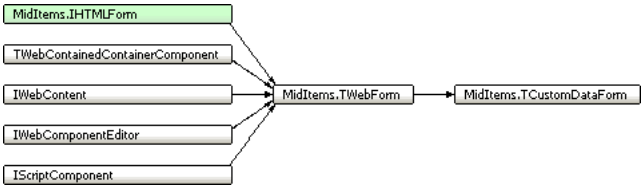
See Also

Align (see page 5257)

52.1.3 TCustomDataForm Class

TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.

Class Hierarchy



Pascal

```
TCustomDataForm = class(TWebForm);
```

C++

```
class TCustomDataForm : public TWebForm;
```

File

MidItems

Description

TCustomDataForm is the base class for components that generate HTML forms to display database information that users can view and edit. Such forms can be added to the items of an InternetExpress page producer (InetXPageProducer).

TCustomDataForm implements the ability to generate an HTML form that contains controls for displaying information from an XML data packet. Buttons on the form manipulate that data, and submit updates back to the XML broker that supplies the XML data packet.

TCustomDataForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that generate the contents of the form to components that generate data displays (TDataGrid (see page 5302) or TFieldGroup (see page 5320)), components that contain buttons which operate on those displays (TDataNavigator (see page 5307)), and components that manage the layout of HTML controls (TLayoutGroup (see page 5389)). At design time, you can add these Web items to the form using the Web page editor.

See Also

IWebContent

TInetXPageProducer (see page 5730)

52.1.3.1 TCustomDataForm Members

The following tables list the members exposed by TCustomDataForm.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 5262)	This is GetHTMLFormTag, a member of class TCustomDataForm.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5554)	Provides the underlying implementation for the AddElements (see page 5550) method.
V	ImplCanAddClass (see page 5262)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5262)	Provides the underlying implementation for the Content method.
V	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

Protected Properties

	Name	Description
R	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Methods

	Name	Description
V	Create (see page 5549)	Creates and initializes an instance of TWebForm (see page 5548).
V	Destroy (see page 5550)	Frees an instance of TWebForm (see page 5548).
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
	Custom (see page 5556)	Represents a tag option and its value in the generated FORM tag.
R	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.
	Style (see page 5556)	Adds style attributes to the generated form.
	StyleRule (see page 5557)	Applies a style in the page producer's style sheet to the button.
R	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

52.1.3.2 TCustomDataForm Methods

The methods of the TCustomDataForm class are listed here.

Protected Methods

	Name	Description
V	GetHTMLFormTag (see page 5262)	This is GetHTMLFormTag, a member of class TCustomDataForm.
V	ImplCanAddClass (see page 5262)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5262)	Provides the underlying implementation for the Content method.

52.1.3.2.1 TCustomDataForm.GetHTMLFormTag Method

Pascal

```
function GetHTMLFormTag(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetHTMLFormTag(TWebContentOptions Options);
```

Description

This is GetHTMLFormTag, a member of class TCustomDataForm.

52.1.3.2.2 TCustomDataForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebForm descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebForm instance to which the component will be added.

In TWebForm, ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 5551)

WebComponents (see page 5558)

52.1.3.2.3 TCustomDataForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm. TWebForm descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

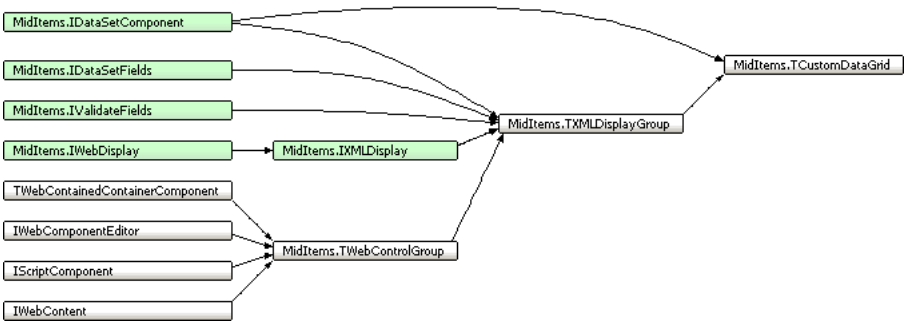
Content (see page 5551)

AddElements (see page 5550)

52.1.4 TCustomDataGrid Class

TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.

Class Hierarchy



Pascal

```
TCustomDataGrid = class(TXMLDisplayGroup, IDataSetComponent);
```

C++

```
class TCustomDataGrid : public TXMLDisplayGroup, public IDataSetComponent;
```

File

MidItems

Description

Use TCustomDataGrid as a base class for Web items that generate an HTML table to display multiple records from a dataset as a set of rows and columns. Each TCustomDataGrid object maintains a list of Web items that generate the individual fields (columns) of the table. These Web items can be from the following classes: TTextColumn (see page 5495), TSelectOptionColumn, TTextAreaColumn (see page 5488), and TStatusColumn (see page 5481).

You can bind the data grid to an XML data packet or dataset field within an XML data packet. The column Web items can then use that binding to determine the source of the data they display.

TCustomDataGrid introduces a number of properties that affect the appearance of the generated table.

In order to let other Web items use the HTML table that the data grid generates, TCustomDataGrid inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TCustomDataGrid. Instead, use a descendant of TCustomDataGrid such as TDataGrid (see page 5302).

See Also

TDataGrid (see page 5302)

IWebContent

TSelectOptionsColumn (see page 5472)

TStatusColumn (see page 5481)

TTextAreaColumn (see page 5488)

TTextColumn (see page 5495)

52.1.4.1 TCustomDataGrid Members

The following tables list the members exposed by TCustomDataGrid.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	FormatTable (see page 5266)	This is FormatTable, a member of class TCustomDataGrid.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 5267)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5267)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5267)	Provides the underlying implementation for the Content method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 5268)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetHeadingAttributes (see page 5268)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetRowAttributes (see page 5268)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 5269)	This is SetTableAttributes, a member of class TCustomDataGrid.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
V	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
V	Create (see page 5266)	Creates and initializes an instance of TCustomDataGrid (see page 5263).
V	Destroy (see page 5266)	Frees an instance of TCustomDataGrid (see page 5263).
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	DisplayRows (see page 5269)	Limits the number of rows in the HTML table.
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.
	HeadingAttributes (see page 5269)	Determines the display characteristics of the table's header.
R	IsMultipleRecordView (see page 5671)	
R	IsMultipleRecordView (see page 5674)	
	RowAttributes (see page 5270)	Determines the display characteristics of the table's body.
	TableAttributes (see page 5270)	Determines the basic display characteristics of the table.
R	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.4.2 TCustomDataGrid.Create Constructor

Creates and initializes an instance of TCustomDataGrid (see page 5263).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomDataGrid (see page 5263) at runtime. TCustomDataGrid (see page 5263) components added at design time using the Web Page editor are instantiated automatically.

52.1.4.3 TCustomDataGrid.Destroy Destructor

Frees an instance of TCustomDataGrid (see page 5263).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDataGrid();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object reference is nil before calling Destroy.

See Also

System::TObject::Free

52.1.4.4 TCustomDataGrid Methods

The methods of the TCustomDataGrid class are listed here.

Protected Methods

	Name	Description
	FormatTable (see page 5266)	This is FormatTable, a member of class TCustomDataGrid.
V	ImplAddElements (see page 5267)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5267)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5267)	Provides the underlying implementation for the Content method.
V	ImplIsMultipleRecordView (see page 5268)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
	SetHeadingAttributes (see page 5268)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetRowAttributes (see page 5268)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 5269)	This is SetTableAttributes, a member of class TCustomDataGrid.

52.1.4.4.1 TCustomDataGrid.FormatTable Method

Pascal

```
function FormatTable(Layout: TLayoutWebContent; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatTable(TLayoutWebContent Layout, TWebContentOptions Options);
```

Description

This is FormatTable, a member of class TCustomDataGrid.

52.1.4.4.2 TCustomDataGrid.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.4.4.3 TCustomDataGrid.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.4.4.4 TCustomDataGrid.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.4.4.5 TCustomDataGrid.ImplIsMultipleRecordView Method

Pascal

```
function ImplIsMultipleRecordView: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsMultipleRecordView();
```

Description

This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.

52.1.4.4.6 TCustomDataGrid.SetHeadingAttributes Method

Pascal

```
procedure SetHeadingAttributes(const Value: TGridRowAttributes);
```

C++

```
__fastcall SetHeadingAttributes(const TGridRowAttributes Value);
```

Description

This is SetHeadingAttributes, a member of class TCustomDataGrid.

52.1.4.4.7 TCustomDataGrid.SetRowAttributes Method

Pascal

```
procedure SetRowAttributes(const Value: TGridRowAttributes);
```

C++

```
__fastcall SetRowAttributes(const TGridRowAttributes Value);
```

Description

This is SetRowAttributes, a member of class TCustomDataGrid.

52.1.4.4.8 TCustomDataGrid.SetTableAttributes Method

Pascal

```
procedure SetTableAttributes(const Value: TGridAttributes);
```

C++

```
__fastcall SetTableAttributes(const TGridAttributes Value);
```

Description

This is SetTableAttributes, a member of class TCustomDataGrid.

52.1.4.5 TCustomDataGrid Properties

The properties of the TCustomDataGrid class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 5269)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 5269)	Determines the display characteristics of the table's header.
	RowAttributes (see page 5270)	Determines the display characteristics of the table's body.
	TableAttributes (see page 5270)	Determines the basic display characteristics of the table.

52.1.4.5.1 TCustomDataGrid.DisplayRows Property

Limits the number of rows in the HTML table.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify the number of rows that appear in the generated HTML table.

Note that the number of rows in the HTML table does not limit the number of records that end users can view and edit. When DisplayRows is less than the number of records in the data packet, the control generated by TNextPageButton (see page 5394) or TPriorPageButton (see page 5401) lets end users scroll through the records in the data packet.

See Also

MaxRecords

TNextPageButton (see page 5394)

TPriorPageButton (see page 5401)

52.1.4.5.2 TCustomDataGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes (see page 5270) or RowAttributes (see page 5270) property instead.

See Also

TableAttributes (see page 5270)

RowAttributes (see page 5270)

52.1.4.5.3 TCustomDataGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes (see page 5270) or HeadingAttributes (see page 5269) property instead.

See Also

TableAttributes (see page 5270)

HeadingAttributes (see page 5269)

52.1.4.5.4 TCustomDataGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any

custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes (see page 5269) property instead.

See Also

RowAttributes (see page 5270)

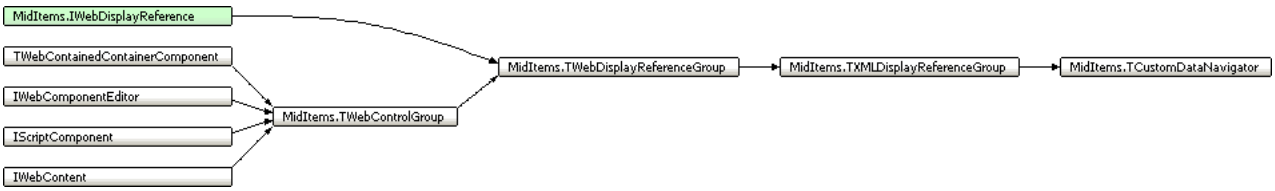
DisplayRows (see page 5269)

HeadingAttributes (see page 5269)

52.1.5 TCustomDataNavigator Class

TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.

Class Hierarchy



Pascal

```
TCustomDataNavigator = class(TXMLDisplayReferenceGroup);
```

C++

```
class TCustomDataNavigator : public TXMLDisplayReferenceGroup;
```

File

MidItems

Description

Use TCustomDataNavigator as a base class for Web items that generate the HTML to arrange a collection of button controls in a data form. Each TCustomDataNavigator object has an associated Web item that generates display controls in the form. The buttons arranged by the data navigator component manipulate the information these display controls show.

TCustomDataNavigator introduces properties that customize the appearance of the buttons it arranges.

Do not create instances of TCustomDataNavigator. TCustomDataNavigator is intended to act as a base class. Instead, instantiate a descendant class such as TDataNavigator (see page 5307).

See Also

TDataForm (see page 5298)

TFieldGroup (see page 5320)

TDataGrid (see page 5302)

IWebContent

52.1.5.1 TCustomDataNavigator Members

The following tables list the members exposed by TCustomDataNavigator.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultButtons (see page 5273)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetWebDisplayComponent (see page 5546)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplAddElements (see page 5642)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5273)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5273)	Provides the underlying implementation for the Content method.
V	ImplGetSubComponents (see page 5274)	Provides the underlying implementation for the GetSubComponents method.
V	Notification (see page 5546)	Forwards notification messages to all owned components.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
V	SetWebDisplayComponent (see page 5547)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	VisibleButtons (see page 5274)	This is VisibleButtons, a member of class TCustomDataNavigator.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
R	WebDisplay (see page 5547)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 5547)	

Public Methods

	Name	Description
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Create (see page 5545)	Creates an instance of TWebDisplayReferenceGroup (see page 5544).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5546)	Frees an instance of TWebDisplayReferenceGroup (see page 5544).
	GetWebDisplayComponent (see page 5672)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

Public Properties

	Name	Description
	Custom (see page 5275)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5275)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5276)	Applies a style in the page producer's style sheet to the group of buttons.
R	WebDisplayComponent (see page 5672)	
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLComponent (see page 5643)	Associates the group with a component that generates HTML display controls.

52.1.5.2 TCustomDataNavigator Methods

The methods of the TCustomDataNavigator class are listed here.

Protected Methods

	Name	Description
	GetDefaultButtons (see page 5273)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
	ImplCanAddClass (see page 5273)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 5273)	Provides the underlying implementation for the Content method.
	ImplGetSubComponents (see page 5274)	Provides the underlying implementation for the GetSubComponents method.
	VisibleButtons (see page 5274)	This is VisibleButtons, a member of class TCustomDataNavigator.

52.1.5.2.1 TCustomDataNavigator.GetDefaultButtons Method

Pascal

```
procedure GetDefaultButtons; virtual;
```

C++

```
virtual __fastcall GetDefaultButtons();
```

Description

This is GetDefaultButtons, a member of class TCustomDataNavigator.

52.1.5.2.2 TCustomDataNavigator.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.5.2.3 TCustomDataNavigator.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.5.2.4 TCustomDataNavigator.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents method.

Pascal

```
function ImplGetSubComponents: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents method calls ImplGetSubComponents to retrieve the value of WebFieldControls as a TObject. TWebControlGroup descendants override this method if the list is not simply the value of WebFieldControls. For example, some TWebControlGroup descendants supply a default value (DefaultWebComponents) when the WebFieldControls property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 5516)

DefaultWebComponents (see page 5518)

WebFieldControls (see page 5519)

52.1.5.2.5 TCustomDataNavigator.VisibleButtons Method

Pascal

```
function VisibleButtons: TWebComponentList;
```

C++

```
__fastcall TWebComponentList VisibleButtons();
```

Description

This is VisibleButtons, a member of class TCustomDataNavigator.

52.1.5.3 TCustomDataNavigator Properties

The properties of the TCustomDataNavigator class are listed here.

Public Properties

	Name	Description
	Custom (see page 5275)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5275)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5276)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.5.3.1 TCustomDataNavigator.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomDataNavigator (see page 5271) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5275)

StyleRule (see page 5276)

WebFieldControls (see page 5519)

52.1.5.3.2 TCustomDataNavigator.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomDataNavigator (see page 5271) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5275)

StyleRule (see page 5276)

WebFieldControls (see page 5519)

52.1.5.3.3 TCustomDataNavigator.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomDataNavigator (see page 5271) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 5275) property.

See Also

Style (see page 5275)

Styles (see page 5716)

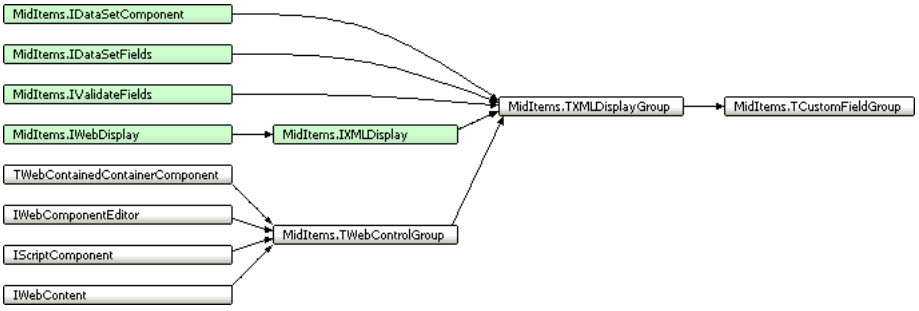
StylesFile (see page 5717)

WebFieldControls (see page 5519)

52.1.6 TCustomFieldGroup Class

TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.

Class Hierarchy



Pascal

```
TCustomFieldGroup = class(TXMLDisplayGroup);
```

C++

```
class TCustomFieldGroup : public TXMLDisplayGroup;
```

File

MidItems

Description

Use TCustomFieldGroup as a base class for Web items that display information in labeled fields on an HTML form. Each TCustomFieldGroup object maintains a list of Web items that generate the individual display fields. These Web items are restricted to classes that are appropriate for the specific TCustomFieldGroup descendant.

If the controls generated for TCustomFieldGroup by its Web items represent database information, you can bind the field group to an XML data packet or dataset field within an XML data packet. The Web items can use that binding to determine the source of the data they display.

TCustomFieldGroup introduces a number of properties that affect the appearance of the generated set of fields.

In order to let other Web items use the HTML that the TCustomFieldGroup object generates, TCustomFieldGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TCustomFieldGroup. Instead, use a descendant of TCustomFieldGroup such as TFieldGroup (see page 5320) or TQueryFieldGroup (see page 5415).

See Also

TFieldGroup (see page 5320)

IWebContent

TQueryFieldGroup (see page 5415)

52.1.6.1 TCustomFieldGroup Members

The following tables list the members exposed by TCustomFieldGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 5280)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.

	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplAddElements (see page 5626)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5280)	Provides the underlying implementation for the Content method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
V	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
V	Create (see page 5279)	Creates and initializes an instance of TCustomFieldGroup (see page 5276).
V	Destroy (see page 5279)	Frees an instance of TCustomFieldGroup (see page 5276).
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	Custom (see page 5281)	Represents a tag option and its value in the generated TABLE tag.
R	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.

	IsMultipleRecordView (see page 5671)	
	IsMultipleRecordView (see page 5674)	
	Style (see page 5281)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5282)	Applies a style in the page producer's style sheet to the table.
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.
	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.
	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.6.2 TCustomFieldGroup.Create Constructor

Creates and initializes an instance of TCustomFieldGroup (see page 5276).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomFieldGroup (see page 5276) at runtime. TCustomFieldGroup (see page 5276) components added at design time using the Web Page editor are instantiated automatically.

52.1.6.3 TCustomFieldGroup.Destroy Destructor

Frees an instance of TCustomFieldGroup (see page 5276).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomFieldGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object reference is not nil before calling Destroy.

See Also

System::TObject::Free

52.1.6.4 TCustomFieldGroup Methods

The methods of the TCustomFieldGroup class are listed here.

Protected Methods

	Name	Description
	GetLayoutAttributes (see page 5280)	Returns an object that specifies how the generated group of controls should look.
	ImplContent (see page 5280)	Provides the underlying implementation for the Content method.

52.1.6.4.1 TCustomFieldGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 5514)

52.1.6.4.2 TCustomFieldGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.6.5 TCustomFieldGroup Properties

The properties of the TCustomFieldGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 5281)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5281)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5282)	Applies a style in the page producer's style sheet to the table.

52.1.6.5.1 TCustomFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomFieldGroup (see page 5276) generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5281)

StyleRule (see page 5282)

52.1.6.5.2 TCustomFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomFieldGroup (see page 5276) generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5281)

StyleRule (see page 5282)

52.1.6.5.3 TCustomFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 5281) property.

See Also

Style (see page 5281)

Styles (see page 5716)

StylesFile (see page 5717)

52.1.7 TCustomLayoutGroup Class

TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.

Class Hierarchy

Pascal

```
TCustomLayoutGroup = class(TWebControlGroup);
```

C++

```
class TCustomLayoutGroup : public TWebControlGroup;
```

File

MidItems

Description

Use TCustomLayoutGroup as a base class for Web items that are used purely for layout. There are few restrictions on what Web items can use a layout group because it has no special content that it displays. It merely imposes a layout on content generated

by other Web items.

TCustomLayoutGroup maintains a list of Web items that display information in the final HTML document, but imposes no restrictions of its own on the possible types of Web items. Instead, it restricts those Web items to the classes that its parent allows. In order to let the parent component use the HTML that TCustomLayoutGroup generates, TCustomLayoutGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TCustomLayoutGroup. Instead, use a descendant of TCustomLayoutGroup such as TLayoutGroup (see page 5389).

See Also

TLayoutGroup (see page 5389)

IWebContent

52.1.7.1 TCustomLayoutGroup Members

The following tables list the members exposed by TCustomLayoutGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5284)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5285)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5285)	Provides the underlying implementation for the Content method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Methods

	Name	Description
V	Create (see page 5284)	Creates and initializes an instance of TCustomLayoutGroup (see page 5282).
V	Destroy (see page 5284)	Frees an instance of TCustomLayoutGroup (see page 5282).

Public Properties

	Name	Description
	Custom (see page 5286)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 5286)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 5287)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5287)	Applies a style in the page producer's style sheet to the group of buttons.
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

52.1.7.2 TCustomLayoutGroup.Create Constructor

Creates and initializes an instance of TCustomLayoutGroup (see page 5282).

Pascal

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

C++

```
virtual __fastcall TCustomLayoutGroup(TComponent * AComponent);
```

Description

Call Create to instantiate TCustomLayoutGroup (see page 5282) at runtime. TCustomLayoutGroup (see page 5282) components added at design time using the Web Page editor are instantiated automatically.

52.1.7.3 TCustomLayoutGroup.Destroy Destructor

Frees an instance of TCustomLayoutGroup (see page 5282).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomLayoutGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the layout group is nil before calling Destroy.

See Also

System::TObject::Free

52.1.7.4 TCustomLayoutGroup Methods

The methods of the TCustomLayoutGroup class are listed here.

Protected Methods

	Name	Description
	GetLayoutAttributes (see page 5284)	Returns an object that specifies how the generated group of controls should look.
	ImplCanAddClass (see page 5285)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 5285)	Provides the underlying implementation for the Content method.

52.1.7.4.1 TCustomLayoutGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 5514)

52.1.7.4.2 TCustomLayoutGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.7.4.3 TCustomLayoutGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.7.5 TCustomLayoutGroup Properties

The properties of the TCustomLayoutGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 5286)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 5286)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 5287)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5287)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.7.5.1 TCustomLayoutGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomLayoutGroup (see page 5282) generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls (see page 5519) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the group of controls and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5287)

StyleRule (see page 5287)

WebFieldControls (see page 5519)

DisplayColumns (see page 5286)

52.1.7.5.2 TCustomLayoutGroup.DisplayColumns Property

Specifies the number of columns the layout group uses to lay out its child controls.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

TCustomLayoutGroup (see page 5282) generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls (see page 5519) property. It adds the HTML generated by each Web item listed in WebFieldControls (see page 5519) to a cell of the table, filling the cells out row by row until it displays all the Web items. DisplayColumns determines the number of cells in each row.

See Also

WebFieldControls (see page 5519)

52.1.7.5.3 TCustomLayoutGroup.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomLayoutGroup (see page 5282) generates an HTML table to lay out the controls generated by the Web items in WebFieldControls (see page 5519). Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5286)

StyleRule (see page 5287)

WebFieldControls (see page 5519)

52.1.7.5.4 TCustomLayoutGroup.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomLayoutGroup (see page 5282) generates an HTML table to lay out the controls generated by its Web items. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 5287) property.

See Also

Style (see page 5287)

Styles

StylesFile

WebFieldControls (see page 5519)

52.1.8 TCustomQueryButtons Class

TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.

Class Hierarchy

Pascal

```
TCustomQueryButtons = class(TWebControlGroup);
```

C++

```
class TCustomQueryButtons : public TWebControlGroup;
```

File

MidItems

Description

Use TCustomQueryButtons as a base class for Web items that generate the HTML to arrange a collection of button controls in a query form. The buttons arranged by the query buttons component manipulate information shown by display controls in the parent Query form.

TCustomQueryButtons introduces properties that customize the appearance of the buttons it arranges.

Do not create instances of TCustomQueryButtons. TCustomQueryButtons is intended to act as a base class. Instead, instantiate a descendant class such as TQueryButtons (see page 5412).

See Also

TQueryForm (see page 5421)

TQueryFieldGroup (see page 5415)

IWebContent

52.1.8.1 TCustomQueryButtons Members

The following tables list the members exposed by TCustomQueryButtons.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultButtons (see page 5289)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5290)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5290)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5290)	Provides the underlying implementation for the Content method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	VisibleButtons (see page 5291)	This is VisibleButtons, a member of class TCustomQueryButtons.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Methods

	Name	Description
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).

Public Properties

	Name	Description
	Custom (see page 5291)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5292)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5292)	Applies a style in the page producer's style sheet to the group of buttons.
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

52.1.8.2 TCustomQueryButtons Methods

The methods of the TCustomQueryButtons class are listed here.

Protected Methods

	Name	Description
V	GetDefaultButtons (see page 5289)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
V	ImplAddElements (see page 5290)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5290)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5290)	Provides the underlying implementation for the Content method.
	VisibleButtons (see page 5291)	This is VisibleButtons, a member of class TCustomQueryButtons.

52.1.8.2.1 TCustomQueryButtons.GetDefaultButtons Method

Pascal

```
procedure GetDefaultButtons; virtual;
```

C++

```
virtual __fastcall GetDefaultButtons();
```

Description

This is GetDefaultButtons, a member of class TCustomQueryButtons.

52.1.8.2.2 TCustomQueryButtons.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.8.2.3 TCustomQueryButtons.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.8.2.4 TCustomQueryButtons.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.8.2.5 TCustomQueryButtons.VisibleButtons Method

Pascal

```
function VisibleButtons: TWebComponentList;
```

C++

```
__fastcall TWebComponentList VisibleButtons();
```

Description

This is VisibleButtons, a member of class TCustomQueryButtons.

52.1.8.3 TCustomQueryButtons Properties

The properties of the TCustomQueryButtons class are listed here.

Public Properties

	Name	Description
	Custom (see page 5291)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5292)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5292)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.8.3.1 TCustomQueryButtons.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomQueryButtons (see page 5288) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the query buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5292)

StyleRule (see page 5292)

WebFieldControls (see page 5519)

52.1.8.3.2 TCustomQueryButtons.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomQueryButtons (see page 5288) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5291)

StyleRule (see page 5292)

WebFieldControls (see page 5519)

52.1.8.3.3 TCustomQueryButtons.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomQueryButtons (see page 5288) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 5519) property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the

generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 5292) property.

See Also

Style (see page 5292)

Styles (see page 5716)

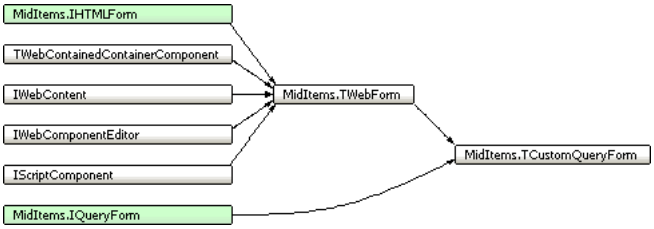
StylesFile (see page 5717)

WebFieldControls (see page 5519)

52.1.9 TCustomQueryForm Class

TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.

Class Hierarchy



Pascal

```
TCustomQueryForm = class(TWebForm, IQueryForm);
```

C++

```
class TCustomQueryForm : public TWebForm, public IQueryForm;
```

File

MidItems

Description

Use TCustomQueryForm as a base class for components that generate an HTML form that displays application-defined values and submits them to an InternetExpress application. TCustomQueryForm introduces properties for specifying the method and target URL of messages submitted by the form.

When building an HTML form, TCustomQueryForm makes use of content generated by the components listed in its WebComponents (see page 5558) property. These components can include Web items that display application-defined values (TQueryFieldGroup (see page 5415)) or that generate buttons to operate on those values (TQueryButtons (see page 5412)). They can also include components that manage the layout of other HTML controls (TLayoutGroup (see page 5389)).

In order to let an InternetExpress page producer use the generated HTML query form, TCustomQueryForm inherits support for the IWebContent interface from TWebForm (see page 5548).

Do not create instances of TCustomQueryForm. Instead, create descendants of TCustomQueryForm, such as TQueryForm (see page 5421).

See Also

TQueryForm (see page 5421)

IWebContent

TInetXPageProducer (see page 5730)

52.1.9.1 TCustomQueryForm Members

The following tables list the members exposed by TCustomQueryForm.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	AssignStringsCallback (see page 5295)	This is AssignStringsCallback, a member of class TCustomQueryForm.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 5296)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5296)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5296)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5297)	Provides the underlying implementation for the Content method.
	MethodString (see page 5297)	This is MethodString, a member of class TCustomQueryForm.
V	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

Protected Properties

	Name	Description
R	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Methods

	Name	Description
	AssignStrings (see page 5295)	This is AssignStrings, a member of class TCustomQueryForm.
V	Create (see page 5295)	Creates and initializes an instance of TCustomQueryForm (see page 5293).
V	Destroy (see page 5550)	Frees an instance of TWebForm (see page 5548).
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
	Action (see page 5298)	Specifies the path info portion of the URL to which the query submits request messages.
	Custom (see page 5556)	Represents a tag option and its value in the generated FORM tag.
R	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.
	Method (see page 5298)	This is Method, a member of class TCustomQueryForm.
	Style (see page 5556)	Adds style attributes to the generated form.
	StyleRule (see page 5557)	Applies a style in the page producer's style sheet to the button.
R	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

52.1.9.2 TCustomQueryForm.Create Constructor

Creates and initializes an instance of TCustomQueryForm (see page 5293).

Pascal

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

C++

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

Description

Call Create to instantiate a query form at runtime. Query forms added at design time using the Web Page editor are instantiated automatically.

52.1.9.3 TCustomQueryForm Methods

The methods of the TCustomQueryForm class are listed here.

Protected Methods

	Name	Description
	AssignStringsCallback (see page 5295)	This is AssignStringsCallback, a member of class TCustomQueryForm.
V	GetHTMLFormTag (see page 5296)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
V	ImplAddElements (see page 5296)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5296)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5297)	Provides the underlying implementation for the Content method.
	MethodString (see page 5297)	This is MethodString, a member of class TCustomQueryForm.

Public Methods

	Name	Description
	AssignStrings (see page 5295)	This is AssignStrings, a member of class TCustomQueryForm.

52.1.9.3.1 TCustomQueryForm.AssignStrings Method

Pascal

```
procedure AssignStrings(Value: TStrings);
```

C++

```
__fastcall AssignStrings(TStrings Value);
```

Description

This is AssignStrings, a member of class TCustomQueryForm.

52.1.9.3.2 TCustomQueryForm.AssignStringsCallback Method

Pascal

```
procedure AssignStringsCallback(AComponent: TComponent);
```

C++

```
__fastcall AssignStringsCallback(TComponent * AComponent);
```

Description

This is AssignStringsCallback, a member of class TCustomQueryForm.

52.1.9.3.3 TCustomQueryForm.GetHTMLFormTag Method

Pascal

```
function GetHTMLFormTag(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetHTMLFormTag(TWebContentOptions Options);
```

Description

This is GetHTMLFormTag, a member of class TCustomQueryForm.

52.1.9.3.4 TCustomQueryForm.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any javascript declarations by calling the AddIntf interface. These declarations allow the generated display control to call javascript methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebForm, ImplAddElements does nothing. TWebForm descendants can override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5550)

52.1.9.3.5 TCustomQueryForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebForm descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebForm instance to which the component will be added.

In TWebForm, ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 5551)

WebComponents (see page 5558)

52.1.9.3.6 TCustomQueryForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm. TWebForm descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

Content (see page 5551)

AddElements (see page 5550)

52.1.9.3.7 TCustomQueryForm.MethodString Method

Pascal

```
function MethodString: string;
```

C++

```
__fastcall UnicodeString MethodString();
```

Description

This is MethodString, a member of class TCustomQueryForm.

52.1.9.4 TCustomQueryForm Properties

The properties of the TCustomQueryForm class are listed here.

Public Properties

	Name	Description
	Action (see page 5298)	Specifies the path info portion of the URL to which the query submits request messages.
	Method (see page 5298)	This is Method, a member of class TCustomQueryForm.

52.1.9.4.1 TCustomQueryForm.Action Property

Specifies the path info portion of the URL to which the query submits request messages.

Pascal

```
property Action: string;
```

C++

```
__property UnicodeString Action;
```

Description

Use Action to specify the target URL for messages that the query form submits. Action supplies the PathInfo property of Web request messages from the query form. This value is matched to the PathInfo property of a Web action item when determining what object responds to those messages.

See Also

TCustomQueryForm (see page 5293)

PathInfo (see page 3382)

PathInfo (see page 3341)

52.1.9.4.2 TCustomQueryForm.Method Property

Pascal

```
property Method: TFormMethod;
```

C++

```
__property TFormMethod Method;
```

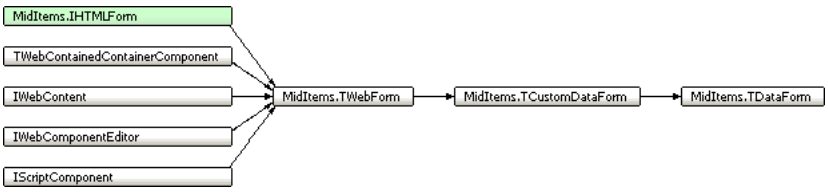
Description

This is Method, a member of class TCustomQueryForm.

52.1.10 TDataForm Class

TDataForm generates an HTML form that displays database information and lets users apply updates.

Class Hierarchy



Pascal

```
TDataForm = class(TCustomDataForm);
```

C++

```
class TDataForm : public TCustomDataForm;
```

File

MidItems

Description

Add TDataForm to the Web items of an Internet Express page producer (TInetXPageProducer (see page 5730)) to add an HTML form that displays database information that users can view and edit.

Data forms generate an HTML form that contains controls which display information from an XML data packet. Buttons on the form manipulate that data, and submit updates back to the XML broker that supplies the XML data packet.

TDataForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that generate the contents of the form to components that generate data displays (TDataGrid (see page 5302) or TFieldGroup (see page 5320)), components that contain buttons which operate on those displays (TDataNavigator (see page 5307)), and components that manage the layout of HTML controls (TLayoutGroup (see page 5389)). At design time, you can add these Web items to the form using the Web page editor.

See Also

TDataGrid (see page 5302)

TBasePageItemsProducer (see page 6390)

TInetXPageProducer (see page 5730)

TFieldGroup (see page 5320)

TlayoutGroup

TDataNavigator (see page 5307)

TApplyUpdatesButton (see page 5250)

Using the Web Page Editor

52.1.10.1 TDataForm Members

The following tables list the members exposed by TDataForm.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 5262)	This is GetHTMLFormTag, a member of class TCustomDataForm.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5554)	Provides the underlying implementation for the AddElements (see page 5550) method.
V	ImplCanAddClass (see page 5262)	Provides the underlying implementation for the CanAddClass method.
A	ImplCanAddClass (see page 5554)	Provides the underlying implementation for the CanAddClass (see page 5551) method.
V	ImplContent (see page 5262)	Provides the underlying implementation for the Content method.

	ImplContent (see page 5555)	Provides the underlying implementation for the Content (see page 5551) method.
	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

Protected Properties

	Name	Description
	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Methods

	Name	Description
	Create (see page 5549)	Creates and initializes an instance of TWebForm (see page 5548).
	Destroy (see page 5550)	Frees an instance of TWebForm (see page 5548).
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.
	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

Published Properties

	Name	Description
	Custom (see page 5300)	Represents a tag option and its value in the generated FORM tag.
	Style (see page 5301)	Adds style attributes to the generated form.
	StyleRule (see page 5301)	Applies a style in the page producer's style sheet to the button.

52.1.10.2 TDataForm Properties

The properties of the TDataForm class are listed here.

Published Properties

	Name	Description
	Custom (see page 5300)	Represents a tag option and its value in the generated FORM tag.
	Style (see page 5301)	Adds style attributes to the generated form.
	StyleRule (see page 5301)	Applies a style in the page producer's style sheet to the button.

52.1.10.2.1 TDataForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the

text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 5556)

52.1.10.2.2 TDataForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 5556)

StyleRule (see page 5557)

52.1.10.2.3 TDataForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5556)

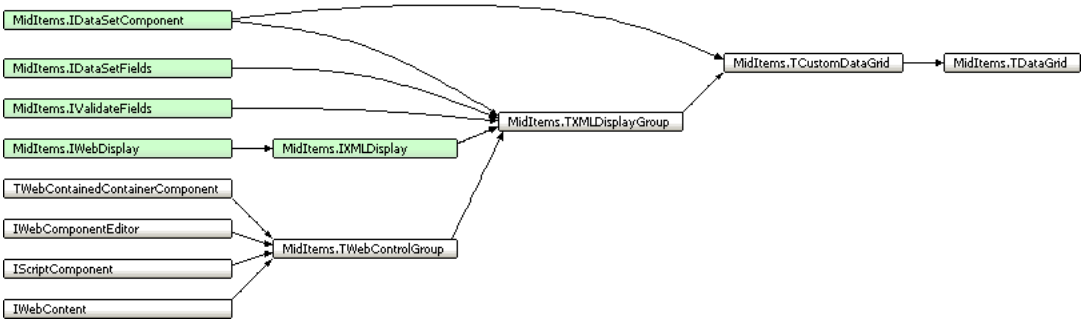
Styles (see page 5716)

StylesFile (see page 5717)

52.1.11 TDataGrid Class

TDataGrid generates an HTML table to display a dataset in an XML data packet.

Class Hierarchy



Pascal

```
TDataGrid = class(TCustomDataGrid);
```

C++

```
class TDataGrid : public TCustomDataGrid;
```

File

MidItems

Description

Add TDataGrid to the Web items of a TDataForm (see page 5298) component (or a layout group that is a child of a TDataForm (see page 5298) component.) TDataGrid generates an HTML table that displays multiple records from a dataset as a set of rows and columns.

Each TDataGrid object maintains a list of Web items that generate the individual fields (columns) of the table. These Web items can be from the following classes: TTextColumn (see page 5495), TSelectOptionColumn, TTextAreaColumn (see page 5488), and TStatusColumn (see page 5481). You can bind the data grid to an XML data packet or dataset field within an XML data packet. The column Web items can then use that binding to determine the source of the data they display.

You can also use the properties of TDataGrid to affect the appearance of the generated table.

See Also

TSelectOptionsColumn (see page 5472)

TStatusColumn (see page 5481)

TTextAreaColumn (see page 5488)

TTextColumn (see page 5495)

TDataForm (see page 5298)

TLayoutGroup (see page 5389)

TWebDataDisplay (see page 5519)

Using the Web Page Editor

52.1.11.1 TDataGrid Members

The following tables list the members exposed by TDataGrid.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	FormatTable (see page 5266)	This is FormatTable, a member of class TCustomDataGrid.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 5267)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplAddElements (see page 5626)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5267)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5267)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 5268)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
V	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetHeadingAttributes (see page 5268)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetRowAttributes (see page 5268)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 5269)	This is SetTableAttributes, a member of class TCustomDataGrid.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.

	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	Create (see page 5266)	Creates and initializes an instance of TCustomDataGrid (see page 5263).
	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
	Create (see page 5622)	Creates and initializes an instance of TXMLDisplayGroup (see page 5620).
	Destroy (see page 5266)	Frees an instance of TCustomDataGrid (see page 5263).
	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
	Destroy (see page 5622)	Frees an instance of TXMLDisplayGroup (see page 5620).
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.
	IsMultipleRecordView (see page 5671)	
	IsMultipleRecordView (see page 5674)	
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.
	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.
	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	DisplayRows (see page 5305)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 5305)	Determines the display characteristics of the table's header.
	RowAttributes (see page 5306)	Determines the display characteristics of the table's body.

	TableAttributes (see page 5306)	Determines the basic display characteristics of the table.
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.11.2 TDataGrid Properties

The properties of the TDataGrid class are listed here.

Published Properties

	Name	Description
	DisplayRows (see page 5305)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 5305)	Determines the display characteristics of the table's header.
	RowAttributes (see page 5306)	Determines the display characteristics of the table's body.
	TableAttributes (see page 5306)	Determines the basic display characteristics of the table.

52.1.11.2.1 TDataGrid.DisplayRows Property

Limits the number of rows in the HTML table.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify the number of rows that appear in the generated HTML table.

Note that the number of rows in the HTML table does not limit the number of records that end users can view and edit. When DisplayRows is less than the number of records in the data packet, the control generated by TNextPageButton or TPriorPageButton lets end users scroll through the records in the data packet.

See Also

MaxRecords

TNextPageButton (see page 5394)

TPriorPageButton (see page 5401)

52.1.11.2.2 TDataGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes or RowAttributes property instead.

See Also

TableAttributes (see page 5270)

RowAttributes (see page 5270)

52.1.11.2.3 TDataGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes or HeadingAttributes property instead.

See Also

TableAttributes (see page 5270)

HeadingAttributes (see page 5269)

52.1.11.2.4 TDataGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes property instead.

See Also

RowAttributes (see page 5270)

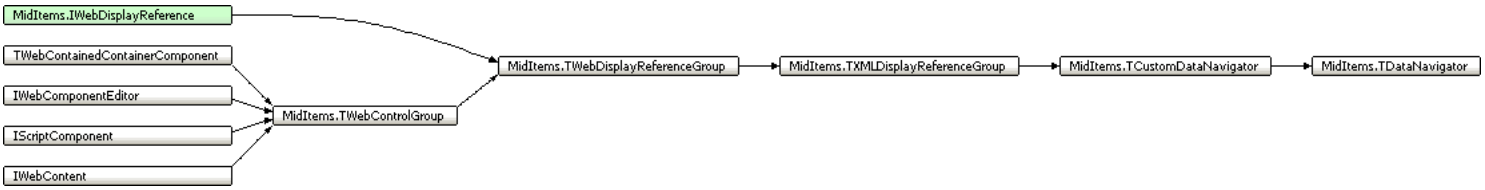
DisplayRows (see page 5269)

HeadingAttributes (see page 5269)

52.1.12 TDataNavigator Class

TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 5311) and TXMLButton (see page 5595) descendants.

Class Hierarchy



Pascal

```
TDataNavigator = class(TCustomDataNavigator);
```

C++

```
class TDataNavigator : public TCustomDataNavigator;
```

File

MidItems

Description

Add TDataNavigator as the child of a TDataForm (see page 5298) component (or a layout group that is a child of a TDataForm (see page 5298) component) to enable the generated HTML form to navigate the displayed data and apply updates that end users make to the data. TDataNavigator can be associated with the Web item that displays the data, such as TDataGrid (see page 5302) or TFieldGroup (see page 5320).

TDataNavigator generates HTML containing the controls generated by its children. These children must be descendants of TDataSetButton (see page 5311) or TXMLButton (see page 5595), such as TInsertButton (see page 5385) and TApplyUpdatesButton (see page 5250).

Use the properties of TDataNavigator to customize the appearance of the group of buttons.

See Also

TDataForm (see page 5298)

TFieldGroup (see page 5320)

TDataGrid (see page 5302)

Using the Web Page Editor

52.1.12.1 TDataNavigator Members

The following tables list the members exposed by TDataNavigator.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.

	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultButtons (see page 5273)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetWebDisplayComponent (see page 5546)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplAddElements (see page 5642)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5273)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5273)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5274)	Provides the underlying implementation for the GetSubComponents method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	Notification (see page 5546)	Forwards notification messages to all owned components.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
V	SetWebDisplayComponent (see page 5547)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	VisibleButtons (see page 5274)	This is VisibleButtons, a member of class TCustomDataNavigator.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
R	WebDisplay (see page 5547)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 5547)	

Public Methods

	Name	Description
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Create (see page 5545)	Creates an instance of TWebDisplayReferenceGroup (see page 5544).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5546)	Frees an instance of TWebDisplayReferenceGroup (see page 5544).
	GetWebDisplayComponent (see page 5672)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

Public Properties

	Name	Description
R	WebDisplayComponent (see page 5672)	
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 5309)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5309)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5310)	Applies a style in the page producer's style sheet to the group of buttons.
	XMLComponent (see page 5310)	Associates the group with a component that generates HTML display controls.

52.1.12.2 TDataNavigator Properties

The properties of the TDataNavigator class are listed here.

Published Properties

	Name	Description
	Custom (see page 5309)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5309)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5310)	Applies a style in the page producer's style sheet to the group of buttons.
	XMLComponent (see page 5310)	Associates the group with a component that generates HTML display controls.

52.1.12.2.1 TDataNavigator.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5275)

StyleRule (see page 5276)

WebFieldControls (see page 5519)

52.1.12.2.2 TDataNavigator.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5275)

StyleRule (see page 5276)

WebFieldControls (see page 5519)

52.1.12.2.3 TDataNavigator.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5275)

Styles (see page 5716)

StylesFile (see page 5717)

WebFieldControls (see page 5519)

52.1.12.2.4 TDataNavigator.XMLComponent Property

Associates the group with a component that generates HTML display controls.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Use XMLComponent to associate the group of buttons with a Web item that generates the display controls those buttons influence. Typical values of XMLComponent are TFieldGroup or TDataGrid.

If all the buttons arranged by the TXMLDisplayReferenceGroup object operate on the same set of display controls, it is most efficient to specify the Web item generating these controls as the value of XMLComponent. Then, each button uses this value to determine its associated display controls when its XMLUseParent property is true. Any buttons that do not operate on the display controls generated by XMLComponent can set their own XMLComponent property to another Web item.

See Also

XMLComponent (see page 5639)

XMLUseParent (see page 5640)

TFieldGroup (see page 5320)

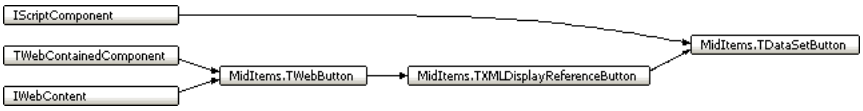
TDataGrid (see page 5302)

TQueryFieldGroup (see page 5415)

52.1.13 TDataSetButton Class

TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.

Class Hierarchy



Pascal

```
TDataSetButton = class(TXMLDisplayReferenceButton, IScriptComponent);
```

C++

```
class TDataSetButton : public TXMLDisplayReferenceButton, public IScriptComponent;
```

File

MidItems

Description

Use TDataSetButton as a base class for components that generate the HTML statements defining a specialized button control for working with the data displayed by another Web item. TDataSetButton objects are used by TDataNavigator (see page 5307), which organizes the generated buttons.

TDataSetButton defines properties and methods for

Generating the HTML code to define the button control.

Associating the button with another Web item that generates HTML for displaying the data that the button manipulates.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TDataSetButton inherits support for the IWebContent interface from TWebButton (see page 5504).

Do not create instances of TDataSetButton. TDataSetButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TDataSetButton, such as TNextButton (see page 5392), TPostButton (see page 5396), TInsertButton (see page 5385), and so on.

See Also

TFirstButton (see page 5363)

TNextButton (see page 5392)

TPriorButton (see page 5399)

TLastButton (see page 5387)

TNextPageButton (see page 5394)

TPriorPageButton (see page 5401)

TInsertButton (see page 5385)

TDeleteButton (see page 5318)

TUndoButton (see page 5502)

TPostButton (see page 5396)

IWebContent

TDataNavigator (see page 5307)

52.1.13.1 TDataSetButton Members

The following tables list the members exposed by TDataSetButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5506)	Creates and initializes an instance of TWebButton (see page 5504).
V	Create (see page 5637)	Creates and initializes an instance of TXMLDisplayReferenceButton (see page 5635).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.

	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.13.2 TDataSetButton Fields

The fields of the TDataSetButton class are listed here.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

52.1.13.2.1 TDataSetButton.DefaultCaption Field

Pascal

```
DefaultCaption: string;
```

C++

```
UnicodeString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TDataSetButton.

52.1.13.2.2 TDataSetButton.XMLMethodName Field

Pascal

```
XMLMethodName: string;
```

C++

```
UnicodeString XMLMethodName;
```

Description

This is XMLMethodName, a member of class TDataSetButton.

52.1.13.3 TDataSetButton Methods

The methods of the TDataSetButton class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	GetCaption (see page 5314)	Returns the value of the Caption property.
	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.

52.1.13.3.1 TDataSetButton.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TDataSetButton.

52.1.13.3.2 TDataSetButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 5508)

SetCaption (see page 5508)

52.1.13.3.3 TDataSetButton.GetMethodName Method

Pascal

```
function GetMethodName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetMethodName();
```

Description

This is GetMethodName, a member of class TDataSetButton.

52.1.13.3.4 TDataSetButton.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TDataSetButton.

52.1.13.3.5 TDataSetButton.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TDataSetButton.

52.1.13.3.6 TDataSetButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 5506)

ILayoutWebContent

52.1.13.4 TDataSetButton Properties

The properties of the TDataSetButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.13.4.1 TDataSetButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 5509)

52.1.13.4.2 TDataSetButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 5508)

Style (see page 5509)

52.1.13.4.3 TDataSetButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or

more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 5509)

StyleRule (see page 5510)

52.1.13.4.4 TDataSetButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 5509)

Styles (see page 5716)

StylesFile (see page 5717)

Styles

StylesFile

52.1.13.4.5 TDataSetButton.XMLComponent Property

Associates the button with the component that generates the display control it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates a display control that the button influences.

Note: Typically, descendants of TXMLDisplayReferenceButton are used by a descendant of TXMLDisplayReferenceGroup, which has its own XMLComponent property. If the button influences this same component, you need not set the XMLComponent property. Instead, setting the XMLUseParent property to true (the default) sets XMLComponent to the value of the parent's XMLComponent property.

See Also

XMLUseParent (see page 5640)

52.1.13.4.6 TDataSetButton.XMLUseParent Property

Specifies whether the button is associated with its parent's display object.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the dataset button determines the component that it influences. When XMLUseParent is false, the XMLComponent property must be set to indicate the display component. When XMLUseParent is true, the button influences the component specified by its parent's XMLComponent property.

The button's parent is the Web item that uses the button to assemble its own HTML content (typically a TDataNavigator or TQueryButtons object). If the parent does not have an associated display component, setting XMLUseParent to true sets the XMLComponent property to nil (Delphi) or NULL (C++).

See Also

XMLComponent (see page 5639)

XMLComponent (see page 5643)

52.1.14 TDeleteButton Class

TDeleteButton generates an HTML control that deletes a record from an XML data packet.

Class Hierarchy

Pascal

```
TDeleteButton = class(TDataSetButton);
```

C++

```
class TDeleteButton : public TDataSetButton;
```

File

MidItems

Description

Add TDeleteButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to delete records. The resulting HTML button's action adds the record deletion to the emerging delta packet, which can then be submitted using an apply updates button.

Use the properties of TDeleteButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the delete button can delete.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

TApplyUpdatesButton (see page 5250)

52.1.14.1 TDeleteButton Members

The following tables list the members exposed by TDeleteButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5320)	Creates and initializes an instance of TDeleteButton (see page 5318).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.

	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.14.2 TDeleteButton.Create Constructor

Creates and initializes an instance of TDeleteButton (see page 5318).

Pascal

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

C++

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

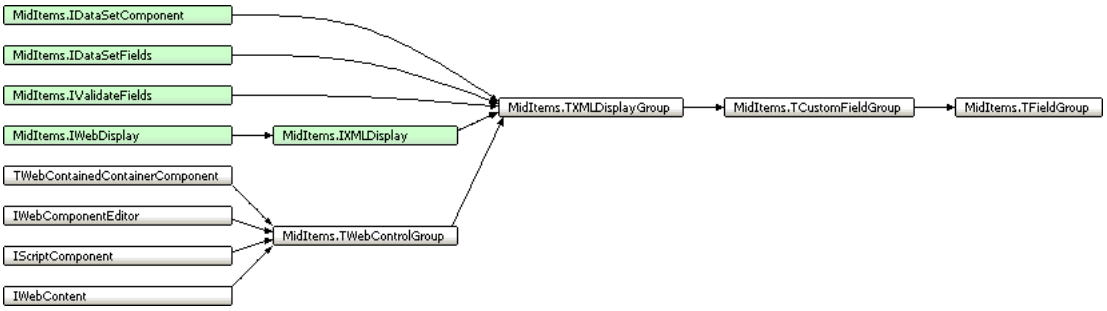
Description

Call Create to instantiate TDeleteButton (see page 5318) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.15 TFieldGroup Class

TFieldGroup arranges database information on an HTML form as a set of labeled fields.

Class Hierarchy



Pascal

```
TFieldGroup = class(TCustomFieldGroup);
```

C++

```
class TFieldGroup : public TCustomFieldGroup;
```

File

MidItems

Description

Add TFieldGroup to a data form component (TDataForm (see page 5298)) or a layout group (TLayoutGroup (see page 5389)) that is a child for a data form. TFieldGroup generates the HTML to display information from an XML data packet in labeled fields.

TFieldGroup maintains a list of Web items that generate the individual display fields. These Web items are restricted to the following classes: TFieldText (see page 5349), TFieldTextArea (see page 5355), TFieldSelectOptions (see page 5335), TFieldRadioGroup (see page 5326), and TFieldStatus (see page 5344). You can bind the field group to an XML data packet or dataset field within an XML data packet. The Web items can then use that binding to determine the source of the data they display.

You can also use the properties of TFieldGroup to affect the overall appearance of the set of fields.

See Also

TDataForm (see page 5298)

TLayoutGroup (see page 5389)

TFieldRadioGroup (see page 5326)

TFieldStatus (see page 5344)

TFieldSelectOptions (see page 5335)

TFieldText (see page 5349)

TFieldTextArea (see page 5355)

Using the Web Page Editor

52.1.15.1 TFieldGroup Members

The following tables list the members exposed by TFieldGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 5280)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.

	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 5323)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5324)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 5280)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
V	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
V	Create (see page 5279)	Creates and initializes an instance of TCustomFieldGroup (see page 5276).
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Create (see page 5622)	Creates and initializes an instance of TXMLDisplayGroup (see page 5620).
V	Destroy (see page 5279)	Frees an instance of TCustomFieldGroup (see page 5276).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5622)	Frees an instance of TXMLDisplayGroup (see page 5620).
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.
	IsMultipleRecordView (see page 5671)	
	IsMultipleRecordView (see page 5674)	
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.
	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.
	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Custom (see page 5324)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5325)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5325)	Applies a style in the page producer's style sheet to the table.
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.15.2 TFieldGroup Methods

The methods of the TFieldGroup class are listed here.

Protected Methods

	Name	Description
	ImplAddElements (see page 5323)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 5324)	Provides the underlying implementation for the CanAddClass method.

52.1.15.2.1 TFieldGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.15.2.2 TFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.15.3 TFieldGroup Properties

The properties of the TFieldGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 5324)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5325)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5325)	Applies a style in the page producer's style sheet to the table.

52.1.15.3.1 TFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomFieldGroup generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5281)

StyleRule (see page 5282)

52.1.15.3.2 TFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomFieldGroup generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5281)

StyleRule (see page 5282)

52.1.15.3.3 TFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5281)

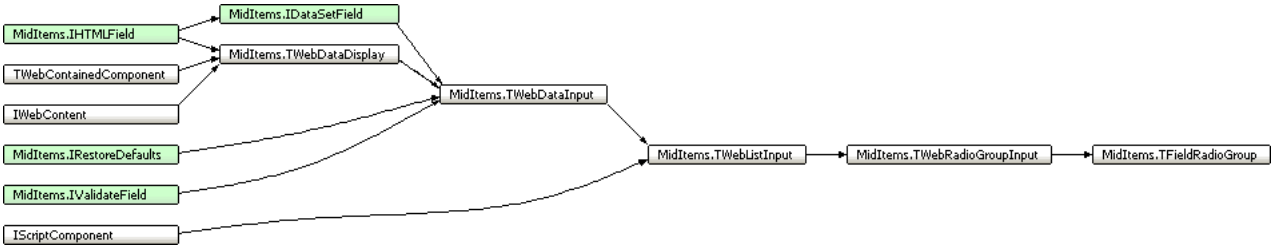
Styles (see page 5716)

StylesFile (see page 5717)

52.1.16 TFieldRadioGroup Class

TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.

Class Hierarchy



Pascal

```
TFieldRadioGroup = class(TWebRadioGroupInput);
```

C++

```
class TFieldRadioGroup : public TWebRadioGroupInput;
```

File

MidItems

Description

Add TFieldRadioGroup as the child of a TFieldGroup (see page 5320) component to add a radio group control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldRadioGroup to

Specify the field whose value the radio group displays.

Specify the possible values for the field from which the user can choose.

Specify a caption that describes the field.

Customize the appearance of the radio group.

Fit the radio group into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 5320)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.16.1 TFieldRadioGroup Members

The following tables list the members exposed by TFieldRadioGroup.

Protected Methods

	Name	Description
	AddAttributes (see page 5570)	This is AddAttributes, a member of class TWebRadioGroupInput.

	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5571)	This is EventContent, a member of class TWebRadioGroupInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	FormatInputs (see page 5571)	This is FormatInputs, a member of class TWebRadioGroupInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetCheckIndex (see page 5571)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 5571)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5571)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
	Create (see page 5570)	Creates and initializes an instance of TWebRadioGroupInput (see page 5567).
	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5329)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5329)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5330)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5330)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5330)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 5331)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5331)	Specifies the width of the radio group, in characters.
	FieldName (see page 5332)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5332)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5333)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ReadOnly (see page 5333)	Indicates whether end users can use the generated control to set values.
	Style (see page 5333)	Adds style attributes to the generated display control.
	StyleRule (see page 5334)	This is StyleRule, a member of class TFieldRadioGroup.
	TabIndex (see page 5334)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5334)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5335)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.16.2 TFieldRadioGroup Properties

The properties of the TFieldRadioGroup class are listed here.

Published Properties

	Name	Description
	Caption (see page 5329)	Specifies the text of a label that indicates what information the generated control displays.

	CaptionAttributes (see page 5329)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5330)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5330)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5330)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 5331)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5331)	Specifies the width of the radio group, in characters.
	FieldName (see page 5332)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5332)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5333)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ReadOnly (see page 5333)	Indicates whether end users can use the generated control to set values.
	Style (see page 5333)	Adds style attributes to the generated display control.
	StyleRule (see page 5334)	This is StyleRule, a member of class TFieldRadioGroup.
	TabIndex (see page 5334)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5334)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5335)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.16.2.1 TFieldRadioGroup.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.16.2.2 TFieldRadioGroup.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.16.2.3 TFieldRadioGroup.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.16.2.4 TFieldRadioGroup.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.16.2.5 TFieldRadioGroup.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.16.2.6 TFieldRadioGroup.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 5565)

Items (see page 5565)

DisplayWidth (see page 5572)

52.1.16.2.7 TFieldRadioGroup.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 5572)

Custom (see page 5530)

52.1.16.2.8 TFieldRadioGroup.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.16.2.9 TFieldRadioGroup.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.16.2.10 TFieldRadioGroup.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.16.2.11 TFieldRadioGroup.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

DisplayWidth (see page 5572)

Custom (see page 5530)

52.1.16.2.12 TFieldRadioGroup.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.16.2.13 TFieldRadioGroup.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TFieldRadioGroup.

52.1.16.2.14 TFieldRadioGroup.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.16.2.15 TFieldRadioGroup.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.16.2.16 TFieldRadioGroup.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

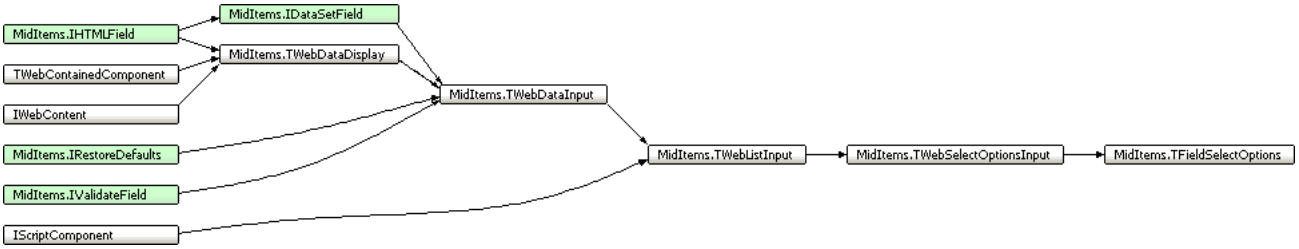
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.17 TFieldSelectOptions Class

TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.

Class Hierarchy



Pascal

```
TFieldSelectOptions = class(TWebSelectOptionsInput);
```

C++

```
class TFieldSelectOptions : public TWebSelectOptionsInput;
```

File

MidItems

Description

Add TFieldSelectOptions as the child of a TFieldGroup (see page 5320) component to add a list control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldSelectOptions to

Specify the field whose value the list box displays.

Specify the possible values for the field from which the user can choose.

Specify a caption that describes the field.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 5320)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.17.1 TFieldSelectOptions Members

The following tables list the members exposed by TFieldSelectOptions.

Protected Methods

	Name	Description
	AddAttributes (see page 5576)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.

	EventContent (see page 5577)	This is EventContent, a member of class TWebSelectOptionsInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	FormatInputs (see page 5577)	This is FormatInputs, a member of class TWebSelectOptionsInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSelectIndex (see page 5577)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
	Create (see page 5576)	Creates and initializes an instance of TWebSelectOptionsInput (see page 5573).

	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5339)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5339)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5339)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5340)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5340)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5341)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5341)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5342)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5342)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Style (see page 5342)	Adds style attributes to the generated display control.
	StyleRule (see page 5343)	This is StyleRule, a member of class TFieldSelectOptions.
	TabIndex (see page 5343)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5343)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5344)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.17.2 TFieldSelectOptions Properties

The properties of the TFieldSelectOptions class are listed here.

Published Properties

	Name	Description
	Caption (see page 5339)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5339)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5339)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5340)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5340)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5341)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5341)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5342)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5342)	Indicates which field in DataSet supplies the possible selections for the generated control.

	Style (see page 5342)	Adds style attributes to the generated display control.
	StyleRule (see page 5343)	This is StyleRule, a member of class TFieldSelectOptions.
	TabIndex (see page 5343)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5343)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5344)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.17.2.1 TFieldSelectOptions.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.17.2.2 TFieldSelectOptions.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.17.2.3 TFieldSelectOptions.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.17.2.4 TFieldSelectOptions.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.17.2.5 TFieldSelectOptions.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections

that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.17.2.6 TFieldSelectOptions.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 5565)

Items (see page 5565)

52.1.17.2.7 TFieldSelectOptions.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.17.2.8 TFieldSelectOptions.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.17.2.9 TFieldSelectOptions.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.17.2.10 TFieldSelectOptions.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.17.2.11 TFieldSelectOptions.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TFieldSelectOptions.

52.1.17.2.12 TFieldSelectOptions.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.17.2.13 TFieldSelectOptions.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.17.2.14 TFieldSelectOptions.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

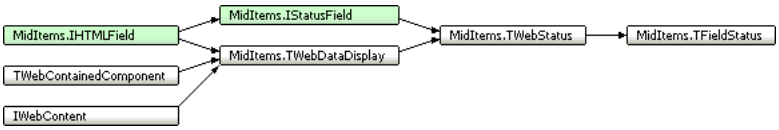
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.18 TFieldStatus Class

TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.

Class Hierarchy



Pascal

```
TFieldStatus = class(TWebStatus);
```

C++

```
class TFieldStatus : public TWebStatus;
```

File

MidItems

Description

Add TFieldStatus as the child of a TFieldGroup (see page 5320) component to provide feedback about whether the current record has been modified. The generated control displays an M if the user has edited the record, an I if the user has inserted the record, and blank if the user has made no changes to the record.

Use the properties and methods of TFieldStatus to

Specify a caption that labels the status control.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 5320)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.18.1 TFieldStatus Members

The following tables list the members exposed by TFieldStatus.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5580)	This is ControlContent, a member of class TWebStatus.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5581)	Returns the value of the Caption property.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.

	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5581)	Returns the name that other components can use to refer to the generated display control.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5580)	Creates and initializes an instance of TWebStatus (see page 5578).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	Identifier (see page 5581)	Returns the name under which record status information is stored in the XML data packet.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Public Properties

	Name	Description
	DisplayWidth (see page 5583)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

Published Properties

	Name	Description
	Caption (see page 5347)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5347)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5347)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5348)	Represents a tag option and its value in the tag for the display control.
	Style (see page 5348)	Adds style attributes to the generated display control.
	StyleRule (see page 5349)	This is StyleRule, a member of class TFieldStatus.

52.1.18.2 TFieldStatus Properties

The properties of the TFieldStatus class are listed here.

Published Properties

	Name	Description
	Caption (see page 5347)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5347)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5347)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5348)	Represents a tag option and its value in the tag for the display control.

	Style (see page 5348)	Adds style attributes to the generated display control.
	StyleRule (see page 5349)	This is StyleRule, a member of class TFieldStatus.

52.1.18.2.1 TFieldStatus.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.18.2.2 TFieldStatus.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.18.2.3 TFieldStatus.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.18.2.4 TFieldStatus.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.18.2.5 TFieldStatus.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.18.2.6 TFieldStatus.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

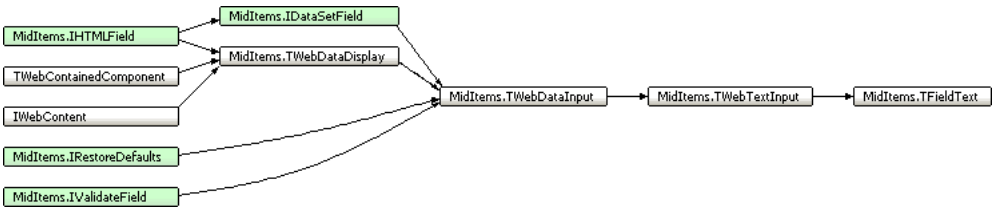
Description

This is StyleRule, a member of class TFieldStatus.

52.1.19 TFieldText Class

TFieldText creates a single-line HTML input control to represent a field in an XML data packet.

Class Hierarchy



Pascal

```
TFieldText = class(TWebTextInput);
```

C++

```
class TFieldText : public TWebTextInput;
```

File

MidItems

Description

Add TFieldText as the child of a TFieldGroup (see page 5320) component to add a single-line edit control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldText to

Specify the field whose value the control displays.

Specify a caption that describes the field.

Customize the appearance of the edit control.

Fit the control into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 5320)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.19.1 TFieldText Members

The following tables list the members exposed by TFieldText.

Protected Methods

	Name	Description
	AddAttributes (see page 5592)	This is AddAttributes, a member of class TWebTextInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5592)	This is ControlContent, a member of class TWebTextInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 5593)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5593)	This is RestoreFieldDefaults, a member of class TWebTextInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
	Create (see page 5592)	Creates and initializes an instance of TWebTextInput (see page 5589).
	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5352)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5352)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5352)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5353)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5353)	Specifies the horizontal size of the generated control.
	FieldName (see page 5354)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5354)	Indicates whether end users can use the generated control to set values.
	Style (see page 5354)	Adds style attributes to the generated display control.
	StyleRule (see page 5355)	This is StyleRule, a member of class TFieldText.
	TabIndex (see page 5355)	Determines the position of the generated control in the HTML form's tab order.

52.1.19.2 TFieldText Properties

The properties of the TFieldText class are listed here.

Published Properties

	Name	Description
	Caption (see page 5352)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5352)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5352)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5353)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5353)	Specifies the horizontal size of the generated control.
	FieldName (see page 5354)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5354)	Indicates whether end users can use the generated control to set values.

	Style (see page 5354)	Adds style attributes to the generated display control.
	StyleRule (see page 5355)	This is StyleRule, a member of class TFieldText.
	TabIndex (see page 5355)	Determines the position of the generated control in the HTML form's tab order.

52.1.19.2.1 TFieldText.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.19.2.2 TFieldText.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.19.2.3 TFieldText.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.19.2.4 TFieldText.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.19.2.5 TFieldText.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

52.1.19.2.6 TFieldText.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.19.2.7 TFieldText.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

52.1.19.2.8 TFieldText.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.19.2.9 TFieldText.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TFieldText.

52.1.19.2.10 TFieldText.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

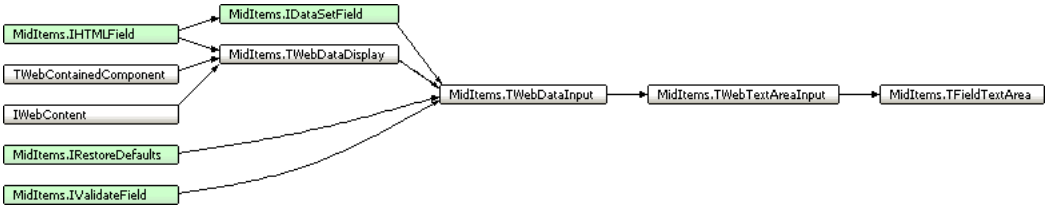
Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.20 TFieldTextArea Class

TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.

Class Hierarchy



Pascal

```
TFieldTextArea = class(TWebTextAreaInput);
```

C++

```
class TFieldTextArea : public TWebTextAreaInput;
```

File

MidItems

Description

Add TFieldTextArea as the child of a TFieldGroup (see page 5320) component to add a multi-line edit control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldTextArea to

Specify the field whose value the control displays.

Specify a caption that describes the field.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TQueryTextArea (see page 5458)

TTextAreaColumn (see page 5488)

52.1.20.1 TFieldTextArea Members

The following tables list the members exposed by TFieldTextArea.

Protected Methods

	Name	Description
	AddAttributes (see page 5586)	This is AddAttributes, a member of class TWebTextAreaInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5586)	This is ControlContent, a member of class TWebTextAreaInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5587)	This is EventContent, a member of class TWebTextAreaInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.

	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 5587)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5587)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5586)	Creates and initializes an instance of TWebTextAreaInput (see page 5583).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5358)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5359)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5359)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5359)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5360)	Specifies the height of the generated control.
	DisplayWidth (see page 5360)	Specifies the horizontal size of the generated control.
	FieldName (see page 5360)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5361)	Indicates whether end users can use the generated control to set values.
	Style (see page 5361)	Adds style attributes to the generated display control.
	StyleRule (see page 5362)	This is StyleRule, a member of class TFieldTextArea.
	TabIndex (see page 5362)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5362)	Specifies how the generated control treats lines that are longer than the display width.

52.1.20.2 TFieldTextArea Properties

The properties of the TFieldTextArea class are listed here.

Published Properties

	Name	Description
	Caption (see page 5358)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5359)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5359)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5359)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5360)	Specifies the height of the generated control.
	DisplayWidth (see page 5360)	Specifies the horizontal size of the generated control.
	FieldName (see page 5360)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5361)	Indicates whether end users can use the generated control to set values.
	Style (see page 5361)	Adds style attributes to the generated display control.
	StyleRule (see page 5362)	This is StyleRule, a member of class TFieldTextArea.
	TabIndex (see page 5362)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5362)	Specifies how the generated control treats lines that are longer than the display width.

52.1.20.2.1 TFieldTextArea.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.20.2.2 TFieldTextArea.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.20.2.3 TFieldTextArea.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.20.2.4 TFieldTextArea.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to

add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.20.2.5 TFieldTextArea.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 5588)

52.1.20.2.6 TFieldTextArea.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 5588)

52.1.20.2.7 TFieldTextArea.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.20.2.8 TFieldTextArea.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

52.1.20.2.9 TFieldTextArea.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.20.2.10 TFieldTextArea.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TFieldTextArea.

52.1.20.2.11 TFieldTextArea.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.20.2.12 TFieldTextArea.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 5588)

52.1.21 TFirstButton Class

TFirstButton generates an HTML control that moves to the first record in an XML data packet.

Class Hierarchy

Pascal

```
TFirstButton = class(TDataSetButton);
```

C++

```
class TFirstButton : public TDataSetButton;
```

File

MidItems

Description

Add TFirstButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to move quickly to the first record in a dataset. The resulting HTML button's action changes the associated display control so that it shows the first record as current.

Use the properties of TFirstButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the first button navigates.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

52.1.21.1 TFirstButton Members

The following tables list the members exposed by TFirstButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5365)	Creates and initializes an instance of TFirstButton (see page 5363).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.21.2 TFirstButton.Create Constructor

Creates and initializes an instance of TFirstButton (see page 5363).

Pascal

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

C++

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

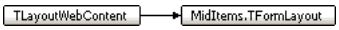
Description

Call Create to instantiate TFirstButton (see page 5363) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.22 TFormLayout Class

TFormLayout lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
TFormLayout = class(TLayoutWebContent);
```

C++

```
class TFormLayout : public TLayoutWebContent;
```

File

MidItems

Description

TFormLayout is the default implementation of the ILayoutWebContent interface for InternetExpress applications. The Web items that belong to an InetXPageProducer object pass a TFormLayout instance to their child Web items as the ParentLayout parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure.

Use the TFormLayout properties to configure how the ILayoutWebContent methods convert HTML that child Web items generate into a final form.

See Also

TLayout

IWebContent

ILayoutWebContent

TInetXPageProducer (see page 5730)

52.1.22.1 TFormLayout Members

The following tables list the members exposed by TFormLayout.

Protected Methods

	Name	Description
	EndButtons (see page 5367)	This is EndButtons, a member of class TFormLayout.
	EndFields (see page 5367)	This is EndFields, a member of class TFormLayout.
	EndRow (see page 5368)	This is EndRow, a member of class TFormLayout.
	EndTable (see page 5368)	This is EndTable, a member of class TFormLayout.
V	ImplLayoutButton (see page 5368)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 5369)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 5369)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 5370)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 5370)	This is NextColumn, a member of class TFormLayout.
	StartButtons (see page 5370)	This is StartButtons, a member of class TFormLayout.
	StartFields (see page 5371)	This is StartFields, a member of class TFormLayout.
	StartRow (see page 5371)	This is StartRow, a member of class TFormLayout.
	StartTable (see page 5371)	This is StartTable, a member of class TFormLayout.

Public Methods

	Name	Description
	Create (see page 5366)	Creates a new TFormLayout (see page 5365) object.
	EndLayout (see page 5367)	Returns the string of HTML to insert after laying out all child items.

Public Properties

	Name	Description
	BreakButtons (see page 5372)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.
	ColumnCount (see page 5372)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
	TableHeader (see page 5372)	Specifies the HTML to generate at the beginning of a table that lays out child elements.

52.1.22.2 TFormLayout.Create Constructor

Creates a new TFormLayout (see page 5365) object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TFormLayout(TLayout AParentLayout);
```

Description

Call Create to instantiate a TFormLayout (see page 5365) object to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout property.

See Also

ParentLayout

52.1.22.3 TFormLayout Methods

The methods of the TFormLayout class are listed here.

Protected Methods

	Name	Description
	EndButtons (see page 5367)	This is EndButtons, a member of class TFormLayout.
	EndFields (see page 5367)	This is EndFields, a member of class TFormLayout.
	EndRow (see page 5368)	This is EndRow, a member of class TFormLayout.
	EndTable (see page 5368)	This is EndTable, a member of class TFormLayout.
V	ImplLayoutButton (see page 5368)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 5369)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 5369)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 5370)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 5370)	This is NextColumn, a member of class TFormLayout.
	StartButtons (see page 5370)	This is StartButtons, a member of class TFormLayout.
	StartFields (see page 5371)	This is StartFields, a member of class TFormLayout.
	StartRow (see page 5371)	This is StartRow, a member of class TFormLayout.
	StartTable (see page 5371)	This is StartTable, a member of class TFormLayout.

Public Methods

	Name	Description
	EndLayout (see page 5367)	Returns the string of HTML to insert after laying out all child items.

52.1.22.3.1 TFormLayout.EndButtons Method

Pascal

```
function EndButtons: string;
```

C++

```
__fastcall UnicodeString EndButtons();
```

Description

This is EndButtons, a member of class TFormLayout.

52.1.22.3.2 TFormLayout.EndFields Method

Pascal

```
function EndFields: string;
```

C++

```
__fastcall UnicodeString EndFields();
```

Description

This is EndFields, a member of class TFormLayout.

52.1.22.3.3 TFormLayout.EndLayout Method

Returns the string of HTML to insert after laying out all child items.

Pascal

```
function EndLayout: string;
```

C++

```
__fastcall UnicodeString EndLayout();
```

Description

EndLayout generates the tags that must be inserted after all the content that child Web items generate. TFormLayout (see page 5365) keeps track of the ILayoutWebContent method calls that are made so that it can know what tags need to be closed.

See Also

ILayoutWebContent

52.1.22.3.4 TFormLayout.EndRow Method

Pascal

```
function EndRow: string;
```

C++

```
__fastcall UnicodeString EndRow();
```

Description

This is EndRow, a member of class TFormLayout.

52.1.22.3.5 TFormLayout.EndTable Method

Pascal

```
function EndTable: string;
```

C++

```
__fastcall UnicodeString EndTable();
```

Description

This is EndTable, a member of class TFormLayout.

52.1.22.3.6 TFormLayout.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutButton(const UnicodeString HTMLButton,
TLayoutAttributes Attributes);
```

Description

The LayoutButton method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton method.

See Also

LayoutButton

52.1.22.3.7 TFormLayout.ImplLayoutField Method

Provides the underlying implementation for the LayoutField method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutField(const UnicodeString HTMLField,
TLayoutAttributes Attributes);
```

Description

The LayoutField method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField method.

See Also

LayoutField

52.1.22.3.8 TFormLayout.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

The LayoutLabelAndField method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField method.

See Also

LayoutLabelAndField

52.1.22.3.9 TFormLayout.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutTable(const UnicodeString HTMLTable,
TLayoutAttributes Attributes);
```

Description

The LayoutTable method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable method.

See Also

layouttable

52.1.22.3.10 TFormLayout.NextColumn Method

Pascal

```
function NextColumn(SubItemCount: Integer): string;
```

C++

```
__fastcall UnicodeString NextColumn(int SubItemCount);
```

Description

This is NextColumn, a member of class TFormLayout.

52.1.22.3.11 TFormLayout.StartButtons Method

Pascal

```
function StartButtons: string;
```

C++

```
__fastcall UnicodeString StartButtons();
```

Description

This is StartButtons, a member of class TFormLayout.

52.1.22.3.12 TFormLayout.StartFields Method

Pascal

```
function StartFields(ColCount: Integer): string;
```

C++

```
__fastcall UnicodeString StartFields(int ColCount);
```

Description

This is StartFields, a member of class TFormLayout.

52.1.22.3.13 TFormLayout.StartRow Method

Pascal

```
function StartRow: string;
```

C++

```
__fastcall UnicodeString StartRow();
```

Description

This is StartRow, a member of class TFormLayout.

52.1.22.3.14 TFormLayout.StartTable Method

Pascal

```
function StartTable: string;
```

C++

```
__fastcall UnicodeString StartTable();
```

Description

This is StartTable, a member of class TFormLayout.

52.1.22.4 TFormLayout Properties

The properties of the TFormLayout class are listed here.

Public Properties

	Name	Description
	BreakButtons (see page 5372)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.
	ColumnCount (see page 5372)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
	TableHeader (see page 5372)	Specifies the HTML to generate at the beginning of a table that lays out child elements.

52.1.22.4.1 TFormLayout.BreakButtons Property

Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.

Pascal

```
property BreakButtons: Boolean;
```

C++

```
__property Boolean BreakButtons;
```

Description

Set BreakButtons to true if the Web item that supplies this object to its child Web items uses an HTML table to position HTML buttons that those child Web items generate. Set BreakButtons to false if the Web item does not use an HTML table to position child buttons.

BreakButtons affects the whether the LayoutButton method on this object's ILayoutWebContent interface adds <TD> tags to the HTML that child Web items generate for HTML buttons.

See Also

layoutbutton

52.1.22.4.2 TFormLayout.ColumnCount Property

Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.

Pascal

```
property ColumnCount: Integer;
```

C++

```
__property int ColumnCount;
```

Description

Set ColumnCount to the number of child elements that appear in a single row when laying out child controls by rows.

ColumnCount affects where the LayoutField or LayoutButton method on this object's ILayoutWebContent interface adds <TR> and </TR> tags to the HTML that child Web items generate.

Note: ColumnCount only affects the LayoutButton method if the value of the BreakButtons (see page 5372) property is true.

See Also

layoutbutton

layoutfield

BreakButtons (see page 5372)

52.1.22.4.3 TFormLayout.TableHeader Property

Specifies the HTML to generate at the beginning of a table that lays out child elements.

Pascal

```
property TableHeader: string;
```

C++

```
__property UnicodeString TableHeader;
```

Description

Set TableHeader to the HTML tag that appears at the beginning of the HTML table used to control the layout of child elements. TableHeader is a string that includes the <table> tag, along with any attributes (such as alignment) that affect the layout of child items.

See Also

layoutbutton

layoutfield

BreakButtons (see page 5372)

52.1.23 TGridAttributes Class

TGridAttributes describes the options the TDataGrid (see page 5302) component adds to an HTML <TABLE> tag.

Class Hierarchy

Pascal

```
TGridAttributes = class(TPersistent);
```

C++

```
class TGridAttributes : public TPersistent;
```

File

MidItems

Description

TGridAttributes implements the TableAttributes property of a TDataGrid (see page 5302) or TAdapterGrid component. It describes display attributes that apply to an HTML table generated by TDataGrid (see page 5302) or TAdapterGrid. The properties of TGridAttributes specify the options added to the generated HTML <TABLE> tag, as well as any styles that should be applied.

See Also

TCustomDataGrid (see page 5263)

TCustomAdapterGrid

52.1.23.1 TGridAttributes Members

The following tables list the members exposed by TGridAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 5374)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	Create (see page 5374)	Creates an instance of TGridAttributes (see page 5373).

Public Properties

	Name	Description
	Parent (see page 5378)	Indicates the TDataGrid (see page 5302) object that uses this TGridAttributes (see page 5373) to describe its attributes.

Published Properties

	Name	Description
	Align (see page 5375)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 5376)	Specifies the background color of the HTML table.
	Border (see page 5376)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 5377)	This is CellPadding, a member of class TGridAttributes.
	CellSpacing (see page 5377)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 5377)	Represents a tag option and its value in an HTML tag.
	Style (see page 5378)	Adds style attributes to the generated table.
	StyleRule (see page 5379)	This is StyleRule, a member of class TGridAttributes.

52.1.23.2 TGridAttributes.Create Constructor

Creates an instance of TGridAttributes (see page 5373).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TGridAttributes(TComponent * AParent);
```

Description

Most applications have no need to instantiate TGridAttributes (see page 5373). TGridAttributes (see page 5373) is instantiated by the TDataGrid (see page 5302) component that uses it.

The AParent parameter is the TDataGrid (see page 5302) or TAdapterGrid component that uses the TGridAttributes (see page 5373) instance. This parameter becomes the value of the Parent (see page 5378) property.

See Also

Parent (see page 5378)

TDataGrid (see page 5302)

TAdapterGrid

52.1.23.3 TGridAttributes Methods

The methods of the TGridAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 5374)	Copies the properties of an object to a destination object.

52.1.23.3.1 TGridAttributes.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

52.1.23.4 TGridAttributes Properties

The properties of the TGridAttributes class are listed here.

Public Properties

	Name	Description
R	Parent (see page 5378)	Indicates the TDataGrid (see page 5302) object that uses this TGridAttributes (see page 5373) to describe its attributes.

Published Properties

	Name	Description
	Align (see page 5375)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 5376)	Specifies the background color of the HTML table.
	Border (see page 5376)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 5377)	This is CellPadding, a member of class TGridAttributes.
	CellSpacing (see page 5377)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 5377)	Represents a tag option and its value in an HTML tag.
	Style (see page 5378)	Adds style attributes to the generated table.
	StyleRule (see page 5379)	This is StyleRule, a member of class TGridAttributes.

52.1.23.4.1 TGridAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the table within the HTML document, or within the cell of a surrounding HTML table in a layout group. Align indicates the desired value of the Align option of the HTML table tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the HTML table where it wants.

See Also

cellpadding (see page 5377)

Border (see page 5376)

52.1.23.4.2 TGridAttributes.BgColor Property

Specifies the background color of the HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuchsia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Note: The row or heading attributes can override this background color for a portion of the HTML table.

See Also

BgColor (see page 5382)

52.1.23.4.3 TGridAttributes.Border Property

Specifies the thickness of the border drawn around the HTML table.

Pascal

```
property Border: Integer;
```

C++

```
__property int Border;
```

Description

Set Border to specify whether a border should be drawn around the table, and if so, how thick it should be. The thickness specified by any particular value of border depends on the web browser that views the HTML table. For any browser, however, larger numbers indicate thicker borders. A value of -1 indicates that no border should be drawn around the table.

See Also

CellSpacing (see page 5377)

52.1.23.4.4 TGridAttributes.CellPadding Property

Pascal

```
property CellPadding: Integer;
```

C++

```
__property int CellPadding;
```

Description

This is CellPadding, a member of class TGridAttributes.

52.1.23.4.5 TGridAttributes.CellSpacing Property

Specifies the amount of space to leave between cells in the HTML table.

Pascal

```
property CellSpacing: Integer;
```

C++

```
__property int CellSpacing;
```

Description

Set CellSpacing to specify how many pixels of whitespace should appear between the cells in the table. A value of -1 indicates that the web browser should decide how to separate the cells of the HTML table.

Note: CellSpacing refers to the whitespace between cells in an HTML table, not the thickness of the lines drawn to separate rows and columns.

See Also

Border (see page 5376)

cellpadding (see page 5377)

52.1.23.4.6 TGridAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options to be added to the HTML <TABLE> tag. Each option and its value should have the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to hide the side borders and add row dividers, you can set Custom to

```
FRAME=HSIDES RULES=ROWS
```

See Also

Style (see page 5378)

stylerule (see page 5379)

52.1.23.4.7 TGridAttributes.Parent Property

Indicates the TDataGrid (see page 5302) object that uses this TGridAttributes (see page 5373) to describe its attributes.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to access the TDataGrid (see page 5302) that uses this TGridAttributes (see page 5373) object. The value of Parent is assigned by the constructor.

See Also

TDataGrid (see page 5302)

Create (see page 5374)

TAdapterGrid

52.1.23.4.8 TGridAttributes.Style Property

Adds style attributes to the generated table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the entire table. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the <TABLE> tag as the value of a STYLE option.

See Also

Custom (see page 5377)

stylerule (see page 5379)

52.1.23.4.9 TGridAttributes.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TGridAttributes.

52.1.24 TGridRowAttributes Class

TGridRowAttributes describes the options the TDataGrid (see page 5302) component adds to its HTML row tags (<TR>).

Class Hierarchy

Pascal

```
TGridRowAttributes = class(TPersistent);
```

C++

```
class TGridRowAttributes : public TPersistent;
```

File

MidItems

Description

TGridRowAttributes implements the RowAttributes and HeadingAttributes properties of a TDataGrid (see page 5302) or TAdapterGrid component. It describes display attributes that apply to the generated HTML table's rows. The properties of TGridRowAttributes specify the options added to the generated HTML <TR> tag, as well as any styles that should be applied to the header or body of the table.

See Also

TCustomDataGrid (see page 5263)

TCustomDataGrid (see page 5263)

TCustomAdapterGrid

TCustomAdapterGrid

52.1.24.1 TGridRowAttributes Members

The following tables list the members exposed by TGridRowAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 5381)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	Create (see page 5380)	Creates an instance of TGridRowAttributes (see page 5379).

Public Properties

	Name	Description
R	Parent (see page 5383)	Indicates the TDataGrid (see page 5302) or TAdapterGrid object that uses this TGridRowAttributes (see page 5379) to describe its header or row attributes.

Published Properties

	Name	Description
	Align (see page 5382)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 5382)	Specifies the background color of the rows or header of an HTML table.
	Custom (see page 5382)	Represents a tag option and its value in an HTML tag.
	Style (see page 5383)	Adds style attributes to the generated table body or header.
	StyleRule (see page 5384)	Applies a style in the page producer's style sheet to the generated table body or header.
	VAlign (see page 5384)	Specifies the vertical alignment within the cells of the HTML table body or header.

52.1.24.2 TGridRowAttributes.Create Constructor

Creates an instance of TGridRowAttributes (see page 5379).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TGridRowAttributes(TComponent * AParent);
```

Description

Most applications have no need to instantiate TGridRowAttributes (see page 5379). TGridRowAttributes (see page 5379) is instantiated by the TDataGrid (see page 5302) or TAdapterGrid component that uses it.

The AParent parameter is the TDataGrid (see page 5302) or TAdapterGrid component that uses the TGridRowAttributes (see page 5379) instance. This parameter becomes the value of the Parent (see page 5383) property.

See Also

Parent (see page 5383)

TDataGrid (see page 5302)

TAdapterGrid

52.1.24.3 TGridRowAttributes Methods

The methods of the TGridRowAttributes class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 5381)	Copies the properties of an object to a destination object.

52.1.24.3.1 TGridRowAttributes.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError

52.1.24.4 TGridRowAttributes Properties

The properties of the TGridRowAttributes class are listed here.

Public Properties

	Name	Description
	Parent (see page 5383)	Indicates the TDataGrid (see page 5302) or TAdapterGrid object that uses this TGridRowAttributes (see page 5379) to describe its header or row attributes.

Published Properties

	Name	Description
	Align (see page 5382)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 5382)	Specifies the background color of the rows or header of an HTML table.
	Custom (see page 5382)	Represents a tag option and its value in an HTML tag.
	Style (see page 5383)	Adds style attributes to the generated table body or header.
	StyleRule (see page 5384)	Applies a style in the page producer's style sheet to the generated table body or header.
	VAlign (see page 5384)	Specifies the vertical alignment within the cells of the HTML table body or header.

52.1.24.4.1 TGridRowAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the contents of each cell in the table row or header. Align indicates the desired value of the Align option of the <TR> tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the cell contents wherever it wants horizontally.

See Also

VAlign (see page 5384)

52.1.24.4.2 TGridRowAttributes.BgColor Property

Specifies the background color of the rows or header of an HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the rows or heading of an HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuchsia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Note: This property overrides the background color option of the HTML table itself.

See Also

BgColor (see page 5376)

52.1.24.4.3 TGridRowAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options to be added to the HTML <TR> tag. Each option and its value should have the form

`OPTIONNAME=Value`

To specify multiple options, separate the Name/value pairs with spaces.

See Also

Style (see page 5383)

StyleRule (see page 5384)

52.1.24.4.4 TGridRowAttributes.Parent Property

Indicates the TDataGrid (see page 5302) or TAdapterGrid object that uses this TGridRowAttributes (see page 5379) to describe its header or row attributes.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to access the TDataGrid (see page 5302) or TAdapterGrid that uses this TGridRowAttributes (see page 5379) object. The value of Parent is assigned by the constructor.

See Also

TDataGrid (see page 5302)

TAdapterGrid

Create (see page 5380)

52.1.24.4.5 TGridRowAttributes.Style Property

Adds style attributes to the generated table body or header.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the rows or header of the HTML table. The Style option consists of one or more attribute descriptions such as

`font-family: arial; font-weight: bold; font-size: 10px`

These attributes are added to the <TR> tag as the value of a STYLE option.

See Also

Custom (see page 5382)

StyleRule (see page 5384)

52.1.24.4.6 TGridRowAttributes.StyleRule Property

Applies a style in the page producer's style sheet to the generated table body or header.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table body or header. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 5383) property instead.

See Also

Style (see page 5383)

Styles (see page 5716)

StylesFile (see page 5717)

Styles

StylesFile

52.1.24.4.7 TGridRowAttributes.VAlign Property

Specifies the vertical alignment within the cells of the HTML table body or header.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Set VAlign to indicate the desired vertical alignment of the contents of each cell. VAlign indicates the desired value of the VAlign option of the <TR> tag for the table rows or header.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document positions the contents of cells in the table element wherever it wishes vertically.

See Also

Align (see page 5382)

52.1.25 TInsertButton Class

TInsertButton generates an HTML control that inserts a new record in an XML data packet.

Class Hierarchy

Pascal

```
TInsertButton = class(TDataSetButton);
```

C++

```
class TInsertButton : public TDataSetButton;
```

File

MidItems

Description

Add TInsertButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to add records to an XML data packet. The resulting HTML button's action adds a new blank record to the emerging XML delta packet and causes the associated display control to display that new record.

Use the properties of TInsertButton to customize the appearance of the button and associate it with a Web item. The associated Web item generates the HTML control that displays the newly inserted record.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

TApplyUpdatesButton (see page 5250)

52.1.25.1 TInsertButton Members

The following tables list the members exposed by TInsertButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.

	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5386)	Creates and initializes an instance of TInsertButton (see page 5385).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.25.2 TInsertButton.Create Constructor

Creates and initializes an instance of TInsertButton (see page 5385).

Pascal

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

C++

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

Description

Call Create to instantiate TInsertButton (see page 5385) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.26 TLastButton Class

TLastButton generates an HTML control that moves to the last record in an XML data packet.

Class Hierarchy

Pascal

```
TLastButton = class(TDataSetButton);
```

C++

```
class TLastButton : public TDataSetButton;
```

File

MidItems

Description

Add TLastButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to move quickly to the last record in a dataset. The resulting HTML button's action changes the associated display control so that it shows the last record as current.

Use the properties of TLastButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the last button navigates.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

52.1.26.1 TLastButton Members

The following tables list the members exposed by TLastButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.

	GetCaption (see page 5314)	Returns the value of the Caption property.
	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
	Notification (see page 5638)	Forwards notification messages to all owned components.
	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	Create (see page 5388)	Creates and initializes an instance of TLastButton (see page 5387).
	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.26.2 TLastButton.Create Constructor

Creates and initializes an instance of TLastButton (see page 5387).

Pascal

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

C++

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

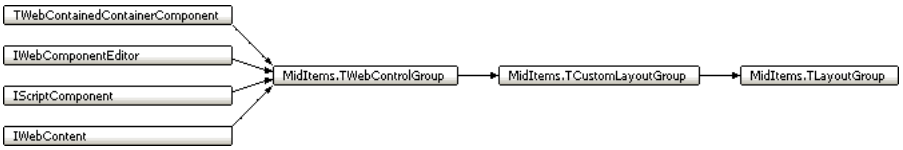
Description

Call Create to instantiate TLastButton (see page 5387) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.27 TLayoutGroup Class

TLayoutGroup generates the HTML to arrange content produced by other Web items.

Class Hierarchy



Pascal

```
TLayoutGroup = class(TCustomLayoutGroup);
```

C++

```
class TLayoutGroup : public TCustomLayoutGroup;
```

File

MidItems

Description

Use TLayoutGroup to adjust the layout of items in an HTML document. There are few restrictions on what Web items can use a layout group because it has no special content that it displays. It merely imposes a layout on content generated by other Web items.

TLayoutGroup maintains a list of Web items that display information in the final HTML document, but imposes no restrictions of its own on the possible types of Web items. Instead, it restricts those Web items to the classes that its parent allows. In order to let the parent component use the HTML that TLayoutGroup generates, TLayoutGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

See Also

IWebContent

Using the Web Page Editor

52.1.27.1 TLayoutGroup Members

The following tables list the members exposed by TLayoutGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.

	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5284)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5285)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5285)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Methods

	Name	Description
V	Create (see page 5284)	Creates and initializes an instance of TCustomLayoutGroup (see page 5282).
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5284)	Frees an instance of TCustomLayoutGroup (see page 5282).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).

Public Properties

	Name	Description
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 5391)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 5391)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 5391)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5392)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.27.2 TLayoutGroup Properties

The properties of the TLayoutGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 5391)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 5391)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 5391)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5392)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.27.2.1 TLayoutGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the group of controls and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5287)

StyleRule (see page 5287)

WebFieldControls (see page 5519)

DisplayColumns (see page 5286)

52.1.27.2.2 TLayoutGroup.DisplayColumns Property

Specifies the number of columns the layout group uses to lay out its child controls.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls property. It adds the HTML generated by each Web item listed in WebFieldControls to a cell of the table, filling the cells out row by row until it displays all the Web items. DisplayColumns determines the number of cells in each row.

See Also

WebFieldControls (see page 5519)

52.1.27.2.3 TLayoutGroup.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in WebFieldControls. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5286)

StyleRule (see page 5287)

WebFieldControls (see page 5519)

52.1.27.2.4 TLayoutGroup.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by its Web items. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5287)

Styles

StylesFile

WebFieldControls (see page 5519)

52.1.28 TNextButton Class

TNextButton generates an HTML control that moves to the next record in an XML data packet.

Class Hierarchy

Pascal

```
TNextButton = class(TDataSetButton);
```

C++

```
class TNextButton : public TDataSetButton;
```

File

MidItems

Description

Add TNextButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable an associated control to move to the next record in a dataset. The resulting HTML button's action changes the current record displayed by the associated control.

Use the properties of TNextButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the next button navigates.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

52.1.28.1 TNextButton Members

The following tables list the members exposed by TNextButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	Create (see page 5394)	Creates and initializes an instance of TNextButton (see page 5392).
	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.28.2 TNextButton.Create Constructor

Creates and initializes an instance of TNextButton (see page 5392).

Pascal

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

C++

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

Description

Call Create to instantiate TNextButton (see page 5392) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.29 TNextPageButton Class

TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.

Class Hierarchy

Pascal

```
TNextPageButton = class(TDataSetButton);
```

C++

```
class TNextPageButton : public TDataSetButton;
```

File

MidItems

Description

Add TNextPageButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to display the next group of records in a dataset. If the button is associated with a grid (TDataGrid (see page 5302)), the grid scrolls by the number of rows in the grid. If the button is associated with a collection of fields (TFieldGroup (see page 5320)), the current record changes to the next record in the data packet.

Use the properties of TNextPageButton to customize the appearance of the button and associate it with a display component.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

52.1.29.1 TNextPageButton Members

The following tables list the members exposed by TNextPageButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.

	Notification (see page 5638)	Forwards notification messages to all owned components.
	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	Create (see page 5396)	Creates and initializes an instance of TNextPageButton (see page 5394).
	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.29.2 TNextPageButton.Create Constructor

Creates and initializes an instance of TNextPageButton (see page 5394).

Pascal

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

C++

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

Description

Call Create to instantiate TNextPageButton (see page 5394) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.30 TPostButton Class

TPostButton generates an HTML control that posts changes to an XML delta packet.

Class Hierarchy

Pascal

```
TPostButton = class(TDataSetButton);
```

C++

```
class TPostButton : public TDataSetButton;
```

File

MidItems

Description

Add TPostButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to commit the edits on the current record of an associated display control. The resulting HTML button's action posts all edits to the current record to the emerging delta packet, which can then be submitted using an apply updates button.

Use the properties of TPostButton to customize the appearance of the button and associate it with a Web item that generates the display control.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

TApplyUpdatesButton (see page 5250)

52.1.30.1 TPostButton Members

The following tables list the members exposed by TPostButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.

	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
	Notification (see page 5638)	Forwards notification messages to all owned components.
	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	Create (see page 5398)	Creates and initializes an instance of TPostButton (see page 5396).
	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.30.2 TPostButton.Create Constructor

Creates and initializes an instance of TPostButton (see page 5396).

Pascal

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

C++

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

Description

Call Create to instantiate TPostButton (see page 5396) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.31 TPriorButton Class

TPriorButton generates an HTML control that moves to the previous record in an XML data packet.

Class Hierarchy

Pascal

```
TPriorButton = class(TDataSetButton);
```

C++

```
class TPriorButton : public TDataSetButton;
```

File

MidItems

Description

Add TPriorButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable an associated control to move to the previous record in a dataset. The resulting HTML button's action changes the current record displayed by the associated control.

Use the properties of TPriorButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the prior button navigates.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

52.1.31.1 TPriorButton Members

The following tables list the members exposed by TPriorButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
	Notification (see page 5638)	Forwards notification messages to all owned components.
	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	Create (see page 5400)	Creates and initializes an instance of TPriorButton (see page 5399).
	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.31.2 TPriorButton.Create Constructor

Creates and initializes an instance of TPriorButton (see page 5399).

Pascal

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

C++

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

Description

Call Create to instantiate TPriorButton (see page 5399) at runtime. Buttons added at design time using the Web Page editor

are instantiated automatically.

52.1.32 TPriorPageButton Class

TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.

Class Hierarchy

Pascal

```
TPriorPageButton = class(TDataSetButton);
```

C++

```
class TPriorPageButton : public TDataSetButton;
```

File

MidItems

Description

Add TPriorPageButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to display the previous group of records in a dataset. If the button is associated with a grid (TDataGrid (see page 5302)), the grid scrolls by the number of rows in the grid. If the button is associated with a collection of fields (TFieldGroup (see page 5320)), the current record changes to the previous record in the data packet.

Use the properties of TNextPageButton (see page 5394) to customize the appearance of the button and associate it with a display component.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

52.1.32.1 TPriorPageButton Members

The following tables list the members exposed by TPriorPageButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5403)	Creates and initializes an instance of TPriorPageButton (see page 5401).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.32.2 TPriorPageButton.Create Constructor

Creates and initializes an instance of TPriorPageButton (see page 5401).

Pascal

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

C++

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

Description

Call Create to instantiate TPriorPageButton (see page 5401) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.33 TQueryButton Class

TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.

Class Hierarchy

Pascal

```
TQueryButton = class(TWebButton);
```

C++

```
class TQueryButton : public TWebButton;
```

File

MidItems

Description

Use TQueryButton as a base class in an InternetExpress application for components that generate the HTML statements defining a specialized button control for managing the application-defined input controls. TQueryButton objects are used by TQueryForm (see page 5421), which organizes the generated buttons and the input controls they manipulate.

TQueryButton defines properties and methods for

Generating the HTML code to define the button control.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Associating the button with a Query form whose controls it influences.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TQueryButton inherits support for the IWebContent interface from TWebButton (see page 5504).

Do not create instances of TQueryButton. TQueryButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TQueryButton, such as TResetQueryButton (see page 5469) or TSubmitQueryButton (see page 5486).

See Also

TResetQueryButton (see page 5469)

TSubmitQueryButton (see page 5486)

IWebContent

TQueryForm (see page 5421)

52.1.33.1 TQueryButton Members

The following tables list the members exposed by TQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5405)	This is DefaultCaption, a member of class TQueryButton.
	InputType (see page 5405)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	EventContent (see page 5406)	This is EventContent, a member of class TQueryButton.
V	GetCaption (see page 5406)	Returns the value of the Caption property.
V	GetHTMLControlName (see page 5406)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 5407)	This is GetHTMLForm, a member of class TQueryButton.
V	GetInputType (see page 5407)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetXMLComponent (see page 5407)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 5407)	Provides the underlying implementation for the Content method.
V	Notification (see page 5408)	Forwards notification messages to all owned components.
	ParentXMLComponent (see page 5408)	This is ParentXMLComponent, a member of class TQueryButton.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetXMLComponent (see page 5408)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 5409)	This is SetXMLUseParent, a member of class TQueryButton.

Public Methods

	Name	Description
V	Create (see page 5404)	Creates and initializes an instance of TQueryButton (see page 5403).

Published Properties

	Name	Description
	Caption (see page 5409)	Specifies the caption that appears on the button.
	Custom (see page 5409)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5410)	Adds style attributes to the generated button control.
	StyleRule (see page 5410)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5411)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 5411)	Specifies whether the button is associated with its parent's query form.

52.1.33.2 TQueryButton.Create Constructor

Creates and initializes an instance of TQueryButton (see page 5403).

Pascal

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

C++

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

Description

Call Create to instantiate a dataset button at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

AOwner is the component that is responsible for freeing the query button. Typically, this is the Web module that contains the InterenetExpress page producer that uses the query button's generated HTML.

52.1.33.3 TQueryButton Fields

The fields of the TQueryButton class are listed here.

Protected Fields

	Name	Description
	DefaultCaption (see page 5405)	This is DefaultCaption, a member of class TQueryButton.
	InputType (see page 5405)	This is InputType, a member of class TQueryButton.

52.1.33.3.1 TQueryButton.DefaultCaption Field

Pascal

```
DefaultCaption: string;
```

C++

```
UnicodeString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TQueryButton.

52.1.33.3.2 TQueryButton.InputType Field

Pascal

```
InputType: string;
```

C++

```
UnicodeString InputType;
```

Description

This is InputType, a member of class TQueryButton.

52.1.33.4 TQueryButton Methods

The methods of the TQueryButton class are listed here.

Protected Methods

	Name	Description
	EventContent (see page 5406)	This is EventContent, a member of class TQueryButton.

	GetCaption (see page 5406)	Returns the value of the Caption property.
	GetHTMLControlName (see page 5406)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 5407)	This is GetHTMLForm, a member of class TQueryButton.
	GetInputType (see page 5407)	This is GetInputType, a member of class TQueryButton.
	GetXMLComponent (see page 5407)	This is GetXMLComponent, a member of class TQueryButton.
	ImplContent (see page 5407)	Provides the underlying implementation for the Content method.
	Notification (see page 5408)	Forwards notification messages to all owned components.
	ParentXMLComponent (see page 5408)	This is ParentXMLComponent, a member of class TQueryButton.
	SetXMLComponent (see page 5408)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 5409)	This is SetXMLUseParent, a member of class TQueryButton.

52.1.33.4.1 TQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TQueryButton.

52.1.33.4.2 TQueryButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 5508)

SetCaption (see page 5508)

52.1.33.4.3 TQueryButton.GetHTMLControlName Method

Pascal

```
function GetHTMLControlName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetHTMLControlName();
```

Description

This is GetHTMLControlName, a member of class TQueryButton.

52.1.33.4.4 TQueryButton.GetHTMLForm Method

Pascal

```
function GetHTMLForm: IHTMLForm;
```

C++

```
__fastcall IHTMLForm GetHTMLForm();
```

Description

This is GetHTMLForm, a member of class TQueryButton.

52.1.33.4.5 TQueryButton.GetInputType Method

Pascal

```
function GetInputType: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetInputType();
```

Description

This is GetInputType, a member of class TQueryButton.

52.1.33.4.6 TQueryButton.GetXMLComponent Method

Pascal

```
function GetXMLComponent: TComponent;
```

C++

```
__fastcall TComponent * GetXMLComponent();
```

Description

This is GetXMLComponent, a member of class TQueryButton.

52.1.33.4.7 TQueryButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 5506)

ILayoutWebContent

52.1.33.4.8 TQueryButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.33.4.9 TQueryButton.ParentXMLComponent Method

Pascal

```
function ParentXMLComponent: TComponent;
```

C++

```
__fastcall TComponent * ParentXMLComponent();
```

Description

This is ParentXMLComponent, a member of class TQueryButton.

52.1.33.4.10 TQueryButton.SetXMLComponent Method

Pascal

```
procedure SetXMLComponent(const Value: TComponent);
```

C++

```
__fastcall SetXMLComponent(const TComponent * Value);
```

Description

This is SetXMLComponent, a member of class TQueryButton.

52.1.33.4.11 TQueryButton.SetXMLUseParent Method

Pascal

```
procedure SetXMLUseParent(const Value: Boolean);
```

C++

```
__fastcall SetXMLUseParent(const Boolean Value);
```

Description

This is SetXMLUseParent, a member of class TQueryButton.

52.1.33.5 TQueryButton Properties

The properties of the TQueryButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 5409)	Specifies the caption that appears on the button.
	Custom (see page 5409)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5410)	Adds style attributes to the generated button control.
	StyleRule (see page 5410)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5411)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 5411)	Specifies whether the button is associated with its parent's query form.

52.1.33.5.1 TQueryButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 5509)

52.1.33.5.2 TQueryButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 5508)

Style (see page 5509)

52.1.33.5.3 TQueryButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 5509)

StyleRule (see page 5510)

52.1.33.5.4 TQueryButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 5509)

Styles (see page 5716)

StylesFile (see page 5717)

Styles

StylesFile

52.1.33.5.5 TQueryButton.XMLComponent Property

Associates the button with the query form whose display controls it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates the form whose controls the query button influences. If this query form contains the TQueryButtons (see page 5412) object that formats the query button with its siblings, you need not set the XMLComponent property. Instead, setting the XMLUseParent (see page 5411) property to true (the default) sets XMLComponent to the Query form that contains the parent TQueryButtons (see page 5412) object.

See Also

XMLUseParent (see page 5411)

TQueryForm (see page 5421)

52.1.33.5.6 TQueryButton.XMLUseParent Property

Specifies whether the button is associated with its parent's query form.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the query button determines the form whose controls it influences. When XMLUseParent is false, the XMLComponent (see page 5411) property must be set to indicate the query form. When XMLUseParent is true, the button influences controls on the query form that contains its parent TQueryButtons (see page 5412) component.

See Also

XMLComponent (see page 5411)

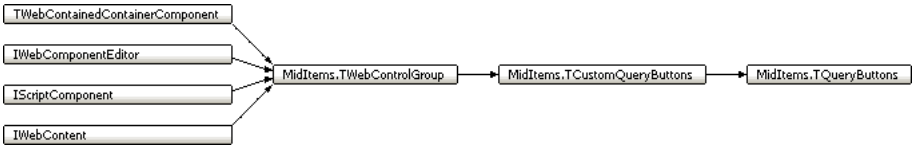
TQueryForm (see page 5421)

TQueryButtons (see page 5412)

52.1.34 TQueryButtons Class

TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 5403) descendants.

Class Hierarchy



Pascal

```
TQueryButtons = class(TCustomQueryButtons);
```

C++

```
class TQueryButtons : public TCustomQueryButtons;
```

File

MidItems

Description

Add TQueryButtons as the child of a TQueryForm (see page 5421) component (or a layout group that is a child of a TQueryForm (see page 5421) component) to enable the generated HTML form to undo or submit changes made by end users. TQueryButtons generates HTML containing the controls generated by its children. These children must be descendants of TQueryButton (see page 5403), such as TSubmitQueryButton (see page 5486) and TResetQueryButton (see page 5469).

In order to let other Web items to use the HTML that defines the organized collection of controls, TQueryButtons inherits the IWebContent interface from TWebControlGroup (see page 5510).

Use the properties of TQueryButtons to customize the appearance of the group of buttons.

See Also

TQueryForm (see page 5421)

TResetQueryButton (see page 5469)

TSubmitQueryButton (see page 5486)

IWebContent

TQueryFieldGroup (see page 5415)

Using the Web Page Editor

52.1.34.1 TQueryButtons Members

The following tables list the members exposed by TQueryButtons.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.

	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultButtons (see page 5289)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5290)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5290)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5290)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	VisibleButtons (see page 5291)	This is VisibleButtons, a member of class TCustomQueryButtons.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Methods

	Name	Description
V	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
V	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).

Public Properties

	Name	Description
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 5413)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5414)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5414)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.34.2 TQueryButtons Properties

The properties of the TQueryButtons class are listed here.

Published Properties

	Name	Description
	Custom (see page 5413)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5414)	Adds style attributes to the generated HTML table.
	StyleRule (see page 5414)	Applies a style in the page producer's style sheet to the group of buttons.

52.1.34.2.1 TQueryButtons.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the query buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5292)

StyleRule (see page 5292)

WebFieldControls (see page 5519)

52.1.34.2.2 TQueryButtons.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5291)

StyleRule (see page 5292)

WebFieldControls (see page 5519)

52.1.34.2.3 TQueryButtons.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5292)

Styles (see page 5716)

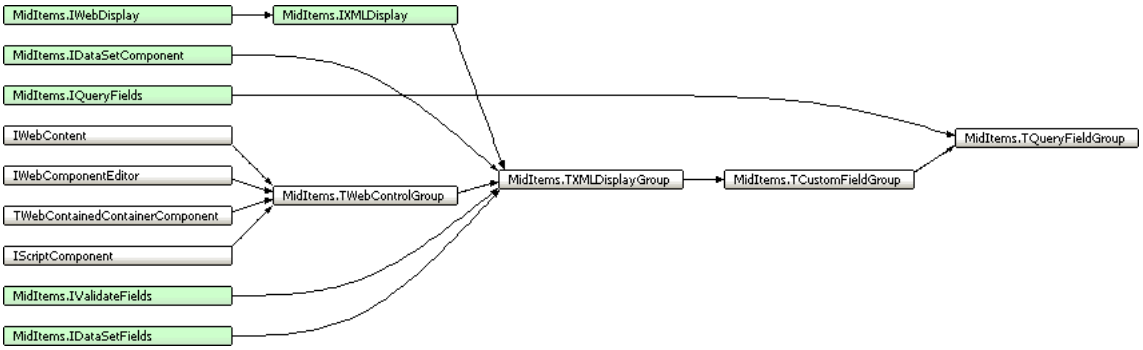
StylesFile (see page 5717)

WebFieldControls (see page 5519)

52.1.35 TQueryFieldGroup Class

TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.

Class Hierarchy



Pascal

```
TQueryFieldGroup = class(TCustomFieldGroup, IQueryFields);
```

C++

```
class TQueryFieldGroup : public TCustomFieldGroup, public IQueryFields;
```

File

MidItems

Description

Add TQueryFieldGroup to a query form component (TQueryForm (see page 5421)) or a layout group (TLayoutGroup (see page 5389)) that is a child for a query form. TQueryFieldGroup generates the HTML to display information in labeled fields. By default, TQueryFieldGroup displays information about the parameters for an XML broker, but it can be used to display any application-defined named values.

TQueryFieldGroup maintains a list of Web items that generate the individual display fields. These Web items are restricted to the following classes: TQueryText (see page 5449), TQueryTextArea (see page 5458), TQuerySelectOptions (see page 5437), and TQueryRadioGroup (see page 5425). If the query field group displays parameter values, you can bind the query field group to an XML data packet or dataset field within an XML data packet. The Web items can then use that binding to

determine the source of the parameters they display.

You can also use the properties of TQueryFieldGroup to affect the overall appearance of the set of fields.

See Also

TQueryForm (see page 5421)

TLayoutGroup (see page 5389)

TQueryRadioGroup (see page 5425)

TQuerySelectOptions (see page 5437)

TQueryText (see page 5449)

TQueryTextArea (see page 5458)

TXMLBroker

TWebDataDisplay (see page 5519)

Using the Web Page Editor

52.1.35.1 TQueryFieldGroup Members

The following tables list the members exposed by TQueryFieldGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	FindParam (see page 5418)	This is FindParam, a member of class TQueryFieldGroup.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 5280)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetParamsList (see page 5418)	This is GetParamsList, a member of class TQueryFieldGroup.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplAddElements (see page 5626)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 5419)	Provides the underlying implementation for the CanAddClass method.

	ImplContent (see page 5280)	Provides the underlying implementation for the Content method.
	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	IsParamInUse (see page 5419)	This is IsParamInUse, a member of class TQueryFieldGroup.
	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	Create (see page 5279)	Creates and initializes an instance of TCustomFieldGroup (see page 5276).
	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
	Create (see page 5622)	Creates and initializes an instance of TXMLDisplayGroup (see page 5620).
	Destroy (see page 5279)	Frees an instance of TCustomFieldGroup (see page 5276).
	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
	Destroy (see page 5622)	Frees an instance of TXMLDisplayGroup (see page 5620).
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateDataSet (see page 5665)	This is ValidateDataSet, a member of class IQueryFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.
	IsMultipleRecordView (see page 5671)	

	IsMultipleRecordView (see page 5674)	
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.
	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.
	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Custom (see page 5419)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5420)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5420)	Applies a style in the page producer's style sheet to the table.
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.35.2 TQueryFieldGroup Methods

The methods of the TQueryFieldGroup class are listed here.

Protected Methods

	Name	Description
	FindParam (see page 5418)	This is FindParam, a member of class TQueryFieldGroup.
	GetParamsList (see page 5418)	This is GetParamsList, a member of class TQueryFieldGroup.
	ImplCanAddClass (see page 5419)	Provides the underlying implementation for the CanAddClass method.
	IsParamInUse (see page 5419)	This is IsParamInUse, a member of class TQueryFieldGroup.

52.1.35.2.1 TQueryFieldGroup.FindParam Method

Pascal

```
function FindParam(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindParam(const UnicodeString AName);
```

Description

This is FindParam, a member of class TQueryFieldGroup.

52.1.35.2.2 TQueryFieldGroup.GetParamsList Method

Pascal

```
procedure GetParamsList(List: TStrings);
```

C++

```
__fastcall GetParamsList(TStrings List);
```

Description

This is GetParamsList, a member of class TQueryFieldGroup.

52.1.35.2.3 TQueryFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.35.2.4 TQueryFieldGroup.IsParamInUse Method

Pascal

```
function IsParamInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsParamInUse(UnicodeString AName);
```

Description

This is IsParamInUse, a member of class TQueryFieldGroup.

52.1.35.3 TQueryFieldGroup Properties

The properties of the TQueryFieldGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 5419)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 5420)	Adds style attributes to the generated set of fields.
	StyleRule (see page 5420)	Applies a style in the page producer's style sheet to the table.

52.1.35.3.1 TQueryFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TCustomFieldGroup generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 5281)

StyleRule (see page 5282)

52.1.35.3.2 TQueryFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

TCustomFieldGroup generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 5281)

StyleRule (see page 5282)

52.1.35.3.3 TQueryFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5281)

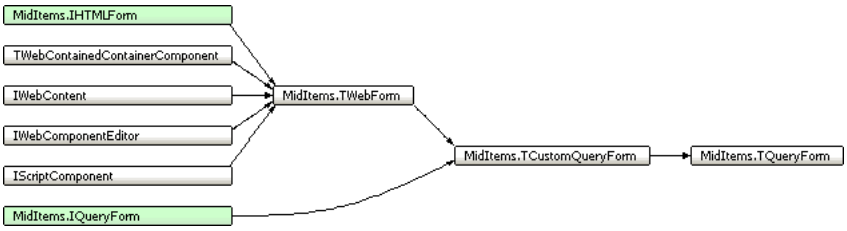
Styles (see page 5716)

StylesFile (see page 5717)

52.1.36 TQueryForm Class

TQueryForm generates HTML forms for letting users enter application-defined values.

Class Hierarchy



Pascal

```
TQueryForm = class(TCustomQueryForm);
```

C++

```
class TQueryForm : public TCustomQueryForm;
```

File

MidItems

Description

Add TQueryForm to the Web items of an InternetExpress page producer to add an HTML form that displays application-defined values that users can view and edit. Query forms submit these values to the InternetExpress application using a specified URL and method type.

TQueryForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that generate the contents of the form to components that display application-defined values (TQueryFieldGroup (see page 5415)), components that contain buttons to operate on those displays (TQueryButtons (see page 5412)), and components that manage the layout of HTML controls (TLayoutGroup (see page 5389)). At design time, you can add these Web items to the form using the Web page editor.

See Also

TQueryButtons (see page 5412)

TBasePageItemsProducer (see page 6390)

TInetXPageProducer (see page 5730)

TQueryFieldGroup (see page 5415)

TLayoutGroup (see page 5389)

Using the Web Page Editor

52.1.36.1 TQueryForm Members

The following tables list the members exposed by TQueryForm.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	AssignStringsCallback (see page 5295)	This is AssignStringsCallback, a member of class TCustomQueryForm.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 5296)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5296)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 5554)	Provides the underlying implementation for the AddElements (see page 5550) method.
V	ImplCanAddClass (see page 5296)	Provides the underlying implementation for the CanAddClass method.
A	ImplCanAddClass (see page 5554)	Provides the underlying implementation for the CanAddClass (see page 5551) method.
V	ImplContent (see page 5297)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5555)	Provides the underlying implementation for the Content (see page 5551) method.
	MethodString (see page 5297)	This is MethodString, a member of class TCustomQueryForm.
V	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

Protected Properties

	Name	Description
R	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Methods

	Name	Description
	AssignStrings (see page 5295)	This is AssignStrings, a member of class TCustomQueryForm.
V	Create (see page 5295)	Creates and initializes an instance of TCustomQueryForm (see page 5293).
V	Create (see page 5549)	Creates and initializes an instance of TWebForm (see page 5548).
V	Destroy (see page 5550)	Frees an instance of TWebForm (see page 5548).
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
R	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.
R	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

Published Properties

	Name	Description
	Action (see page 5423)	Specifies the path info portion of the URL to which the query submits request messages.

	Custom (see page 5423)	Represents a tag option and its value in the generated FORM tag.
	Method (see page 5424)	This is Method, a member of class TQueryForm.
	Style (see page 5424)	Adds style attributes to the generated form.
	StyleRule (see page 5424)	Applies a style in the page producer's style sheet to the button.

52.1.36.2 TQueryForm Properties

The properties of the TQueryForm class are listed here.

Published Properties

	Name	Description
	Action (see page 5423)	Specifies the path info portion of the URL to which the query submits request messages.
	Custom (see page 5423)	Represents a tag option and its value in the generated FORM tag.
	Method (see page 5424)	This is Method, a member of class TQueryForm.
	Style (see page 5424)	Adds style attributes to the generated form.
	StyleRule (see page 5424)	Applies a style in the page producer's style sheet to the button.

52.1.36.2.1 TQueryForm.Action Property

Specifies the path info portion of the URL to which the query submits request messages.

Pascal

```
property Action: string;
```

C++

```
__property UnicodeString Action;
```

Description

Use Action to specify the target URL for messages that the query form submits. Action supplies the PathInfo property of Web request messages from the query form. This value is matched to the PathInfo property of a Web action item when determining what object responds to those messages.

See Also

TCustomQueryForm (see page 5293)

PathInfo (see page 3382)

PathInfo (see page 3341)

52.1.36.2.2 TQueryForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 5556)

52.1.36.2.3 TQueryForm.Method Property

Pascal

```
property Method: TFormMethod;
```

C++

```
__property TFormMethod Method;
```

Description

This is Method, a member of class TQueryForm.

52.1.36.2.4 TQueryForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 5556)

StyleRule (see page 5557)

52.1.36.2.5 TQueryForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of

StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 5556)

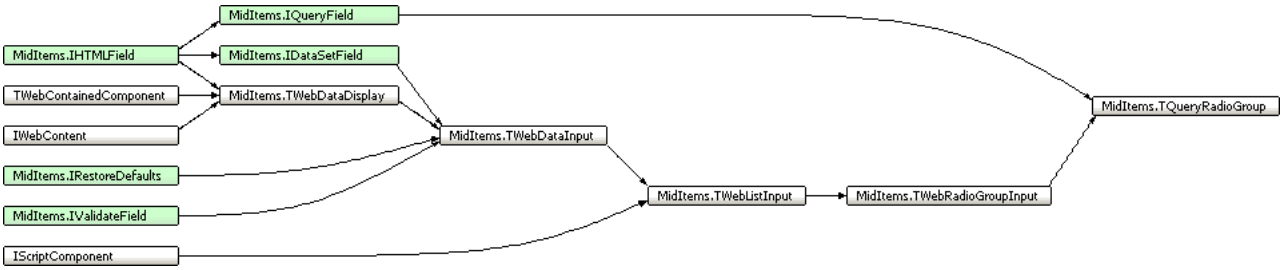
Styles (see page 5716)

StylesFile (see page 5717)

52.1.37 TQueryRadioGroup Class

TQueryRadioGroup creates an HTML radio group control to represent named parameter.

Class Hierarchy



Pascal

```
TQueryRadioGroup = class(TWebRadioGroupInput, IQueryField);
```

C++

```
class TQueryRadioGroup : public TWebRadioGroupInput, public IQueryField;
```

File

MidItems

Description

Add TQueryRadioGroup as the child of a TQueryFieldGroup (see page 5415) component to add a radio group control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 5415) represent the parameters of an XML broker. However, TQueryRadioGroup can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryRadioGroup to

Specify the field or parameter whose value the radio group displays.

Specify the possible values from which the user can choose.

Initialize the value of the radio group.

Specify a caption that describes the value of the radio group.

Customize the appearance of the radio group.

Fit the radio group into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 5415)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

TXMLBroker

52.1.37.1 TQueryRadioGroup Members

The following tables list the members exposed by TQueryRadioGroup.

Protected Methods

	Name	Description
	AddAttributes (see page 5428)	This is AddAttributes, a member of class TQueryRadioGroup.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5571)	This is EventContent, a member of class TWebRadioGroupInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	FormatInputs (see page 5571)	This is FormatInputs, a member of class TWebRadioGroupInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetCheckIndex (see page 5428)	This is GetCheckIndex, a member of class TQueryRadioGroup.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetText (see page 5429)	This is GetText, a member of class TQueryRadioGroup.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 5571)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.

	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5571)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetText (see page 5429)	This is SetText, a member of class TQueryRadioGroup.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
	Create (see page 5570)	Creates and initializes an instance of TWebRadioGroupInput (see page 5567).
	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5429)	Indicates whether the component generates a control that can appear in a query form.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5430)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5430)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5431)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5431)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5431)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 5432)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5432)	Specifies the width of the radio group, in characters.
	FieldName (see page 5433)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5433)	Lists the possible selections that appear in the generated control.

	ItemsField (see page 5434)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 5434)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5435)	Indicates whether end users can use the generated control to set values.
	Style (see page 5435)	Adds style attributes to the generated display control.
	StyleRule (see page 5435)	This is StyleRule, a member of class TQueryRadioGroup.
	TabIndex (see page 5436)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5436)	Specifies the initial value of the radio group.
	Values (see page 5436)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5437)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.37.2 TQueryRadioGroup Methods

The methods of the TQueryRadioGroup class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 5428)	This is AddAttributes, a member of class TQueryRadioGroup.
	GetCheckIndex (see page 5428)	This is GetCheckIndex, a member of class TQueryRadioGroup.
	GetText (see page 5429)	This is GetText, a member of class TQueryRadioGroup.
	SetText (see page 5429)	This is SetText, a member of class TQueryRadioGroup.

Public Methods

	Name	Description
	IsQueryField (see page 5429)	Indicates whether the component generates a control that can appear in a query form.

52.1.37.2.1 TQueryRadioGroup.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TQueryRadioGroup.

52.1.37.2.2 TQueryRadioGroup.GetCheckIndex Method

Pascal

```
function GetCheckIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; override;
```

C++

```
virtual __fastcall int GetCheckIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetCheckIndex, a member of class TQueryRadioGroup.

52.1.37.2.3 TQueryRadioGroup.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall UnicodeString GetText();
```

Description

This is GetText, a member of class TQueryRadioGroup.

52.1.37.2.4 TQueryRadioGroup.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 5421)). IsQueryField returns true, because TQueryRadioGroup (see page 5425) is designed specifically for use by TQueryFieldGroup (see page 5415).

See Also

IsColumn (see page 5527)

TQueryFieldGroup (see page 5415)

TQueryForm (see page 5421)

52.1.37.2.5 TQueryRadioGroup.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const UnicodeString Value);
```

Description

This is SetText, a member of class TQueryRadioGroup.

52.1.37.3 TQueryRadioGroup Properties

The properties of the TQueryRadioGroup class are listed here.

Published Properties

	Name	Description
	Caption (see page 5430)	Specifies the text of a label that indicates what information the generated control displays.

	CaptionAttributes (see page 5430)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5431)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5431)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5431)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 5432)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5432)	Specifies the width of the radio group, in characters.
	FieldName (see page 5433)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5433)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5434)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 5434)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5435)	Indicates whether end users can use the generated control to set values.
	Style (see page 5435)	Adds style attributes to the generated display control.
	StyleRule (see page 5435)	This is StyleRule, a member of class TQueryRadioGroup.
	TabIndex (see page 5436)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5436)	Specifies the initial value of the radio group.
	Values (see page 5436)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5437)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.37.3.1 TQueryRadioGroup.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.37.3.2 TQueryRadioGroup.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.37.3.3 TQueryRadioGroup.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.37.3.4 TQueryRadioGroup.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.37.3.5 TQueryRadioGroup.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.37.3.6 TQueryRadioGroup.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 5565)

Items (see page 5565)

DisplayWidth (see page 5572)

52.1.37.3.7 TQueryRadioGroup.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 5572)

Custom (see page 5530)

52.1.37.3.8 TQueryRadioGroup.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.37.3.9 TQueryRadioGroup.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.37.3.10 TQueryRadioGroup.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.37.3.11 TQueryRadioGroup.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;__property UnicodeString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 5539)

IsQueryField (see page 5528)

Params

xmlbroker (see page 5631)

Setting Web Item Properties

52.1.37.3.12 TQueryRadioGroup.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

DisplayWidth (see page 5572)

Custom (see page 5530)

52.1.37.3.13 TQueryRadioGroup.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.37.3.14 TQueryRadioGroup.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TQueryRadioGroup.

52.1.37.3.15 TQueryRadioGroup.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.37.3.16 TQueryRadioGroup.Text Property

Specifies the initial value of the radio group.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text to initialize the value that appears in the generated radio group. This should be one of the values listed in the Items (see page 5433) property or in the field referenced by the ItemsField (see page 5434) property.

See Also

Items (see page 5565)

ItemsField (see page 5565)

52.1.37.3.17 TQueryRadioGroup.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the

DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.37.3.18 TQueryRadioGroup.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

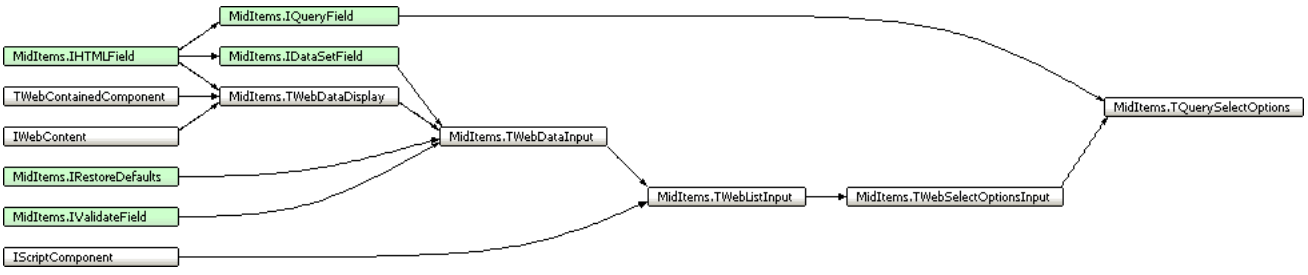
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.38 TQuerySelectOptions Class

TQuerySelectOptions creates an HTML list control to represent a named parameter.

Class Hierarchy



Pascal

```
TQuerySelectOptions = class(TWebSelectOptionsInput, IQueryField);
```

C++

```
class TQuerySelectOptions : public TWebSelectOptionsInput, public IQueryField;
```

File

MidItems

Description

Add TQuerySelectOptions as the child of a TQueryFieldGroup (see page 5415) component to add a list control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 5415) represent the parameters of an XML broker. However, TQuerySelectOptions can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQuerySelectOptions to

Specify the field or parameter whose value the list box displays.

Specify the possible values from which the user can choose.

Assign an initial value to the list box.

Specify a caption that describes the value of the list box.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 5415)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

TXMLBroker

52.1.38.1 TQuerySelectOptions Members

The following tables list the members exposed by TQuerySelectOptions.

Protected Methods

	Name	Description
	AddAttributes (see page 5576)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5577)	This is EventContent, a member of class TWebSelectOptionsInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	FormatInputs (see page 5577)	This is FormatInputs, a member of class TWebSelectOptionsInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.

	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	GetSelectIndex (see page 5441)	This is GetSelectIndex, a member of class TQuerySelectOptions.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetText (see page 5441)	This is GetText, a member of class TQuerySelectOptions.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetText (see page 5441)	This is SetText, a member of class TQuerySelectOptions.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
V	Create (see page 5576)	Creates and initializes an instance of TWebSelectOptionsInput (see page 5573).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
V	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.

	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5441)	Indicates whether the component generates a control that can appear in a query form.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5442)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5442)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5443)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5443)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5444)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5444)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5445)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5445)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5446)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 5446)	Specifies the name of the parameter whose value the generated control represents.
	Style (see page 5447)	Adds style attributes to the generated display control.
	StyleRule (see page 5447)	This is StyleRule, a member of class TQuerySelectOptions.
	TabIndex (see page 5447)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5448)	Specifies the initial value selected in the list box.
	Values (see page 5448)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5448)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.38.2 TQuerySelectOptions Methods

The methods of the TQuerySelectOptions class are listed here.

Protected Methods

	Name	Description
	GetSelectIndex (see page 5441)	This is GetSelectIndex, a member of class TQuerySelectOptions.
	GetText (see page 5441)	This is GetText, a member of class TQuerySelectOptions.
	SetText (see page 5441)	This is SetText, a member of class TQuerySelectOptions.

Public Methods

	Name	Description
	IsQueryField (see page 5441)	Indicates whether the component generates a control that can appear in a query form.

52.1.38.2.1 TQuerySelectOptions.GetSelectIndex Method

Pascal

```
function GetSelectIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; override;
```

C++

```
virtual __fastcall int GetSelectIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetSelectIndex, a member of class TQuerySelectOptions.

52.1.38.2.2 TQuerySelectOptions.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall UnicodeString GetText();
```

Description

This is GetText, a member of class TQuerySelectOptions.

52.1.38.2.3 TQuerySelectOptions.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 5421)). IsQueryField returns true, because TQuerySelectOptions (see page 5437) is designed specifically for use by TQueryFieldGroup (see page 5415).

See Also

IsColumn (see page 5527)

TQueryFieldGroup (see page 5415)

TQueryForm (see page 5421)

52.1.38.2.4 TQuerySelectOptions.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const UnicodeString Value);
```

Description

This is SetText, a member of class TQuerySelectOptions.

52.1.38.3 TQuerySelectOptions Properties

The properties of the TQuerySelectOptions class are listed here.

Published Properties

	Name	Description
	Caption (see page 5442)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5442)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5443)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5443)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5444)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5444)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5445)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5445)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5446)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 5446)	Specifies the name of the parameter whose value the generated control represents.
	Style (see page 5447)	Adds style attributes to the generated display control.
	StyleRule (see page 5447)	This is StyleRule, a member of class TQuerySelectOptions.
	TabIndex (see page 5447)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5448)	Specifies the initial value selected in the list box.
	Values (see page 5448)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5448)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.38.3.1 TQuerySelectOptions.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.38.3.2 TQuerySelectOptions.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.38.3.3 TQuerySelectOptions.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.38.3.4 TQuerySelectOptions.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.38.3.5 TQuerySelectOptions.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.38.3.6 TQuerySelectOptions.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 5565)

Items (see page 5565)

52.1.38.3.7 TQuerySelectOptions.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.38.3.8 TQuerySelectOptions.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.38.3.9 TQuerySelectOptions.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.38.3.10 TQuerySelectOptions.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;__property UnicodeString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 5539)

IsQueryField (see page 5528)

Params

xmlbroker (see page 5631)

Setting Web Item Properties

52.1.38.3.11 TQuerySelectOptions.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.38.3.12 TQuerySelectOptions.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TQuerySelectOptions.

52.1.38.3.13 TQuerySelectOptions.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.38.3.14 TQuerySelectOptions.Text Property

Specifies the initial value selected in the list box.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text to initialize the value of the generated list control. Text should specify one of the values listed in the Items (see page 5445) property or referenced by the ItemsField (see page 5446) property.

See Also

Items (see page 5565)

ItemsField (see page 5565)

52.1.38.3.15 TQuerySelectOptions.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.38.3.16 TQuerySelectOptions.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

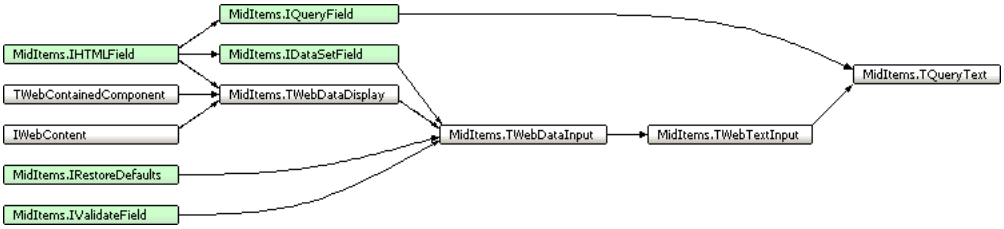
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.39 TQueryText Class

TQueryText creates a single-line HTML input control to represent a named parameter.

Class Hierarchy



Pascal

```
TQueryText = class(TWebTextInput, IQueryField);
```

C++

```
class TQueryText : public TWebTextInput, public IQueryField;
```

File

MidItems

Description

Add TQueryText as the child of a TQueryFieldGroup (see page 5415) component to add an edit control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 5415) represent the parameters of an XML broker. However, TQueryText can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryText to

Specify the field or parameter whose value the control displays.

Assign an initial value to the control.

Specify a caption that describes the value of the control.

Customize the appearance of the input control.

Fit the control into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 5415)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

TXMLBroker

52.1.39.1 TQueryText Members

The following tables list the members exposed by TQueryText.

Protected Methods

	Name	Description
V	AddAttributes (see page 5452)	This is AddAttributes, a member of class TQueryText.
	Content (see page 5523)	Returns the HTML that defines the display control.
A	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 5592)	This is ControlContent, a member of class TWebTextInput.
V	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
V	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetText (see page 5452)	This is GetText, a member of class TQueryText.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 5593)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5593)	This is RestoreFieldDefaults, a member of class TWebTextInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetText (see page 5453)	This is SetText, a member of class TQueryText.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5451)	Creates and initializes an instance of TQueryText (see page 5449).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5452)	Indicates whether the component generates a control that can appear in a query form.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5453)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5454)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5454)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5455)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5455)	Specifies the horizontal size of the generated control.
	FieldName (see page 5455)	Specifies the name of the field whose value the generated control represents.
	MaxWidth (see page 5456)	Limits the length of text users can type into the control.
	ParamName (see page 5456)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5457)	Indicates whether end users can use the generated control to set values.
	Style (see page 5457)	Adds style attributes to the generated display control.
	StyleRule (see page 5458)	This is StyleRule, a member of class TQueryText.
	TabIndex (see page 5458)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5458)	Specifies the initial value of the generated control.

52.1.39.2 TQueryText.Create Constructor

Creates and initializes an instance of TQueryText (see page 5449).

Pascal

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

C++

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

Description

Call Create to instantiate TQueryText (see page 5449) at runtime. TQueryText (see page 5449) components added at design time using the Web Page editor are instantiated automatically.

52.1.39.3 TQueryText Methods

The methods of the TQueryText class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 5452)	This is AddAttributes, a member of class TQueryText.
	GetText (see page 5452)	This is GetText, a member of class TQueryText.
	SetText (see page 5453)	This is SetText, a member of class TQueryText.

Public Methods

	Name	Description
	IsQueryField (see page 5452)	Indicates whether the component generates a control that can appear in a query form.

52.1.39.3.1 TQueryText.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TQueryText.

52.1.39.3.2 TQueryText.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall UnicodeString GetText();
```

Description

This is GetText, a member of class TQueryText.

52.1.39.3.3 TQueryText.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 5421)). IsQueryField returns true, because TQueryText (see page 5449) is designed specifically for TQueryFieldGroup (see page 5415) to use.

See Also

IsColumn (see page 5527)

TQueryFieldGroup (see page 5415)

TQueryForm (see page 5421)

52.1.39.3.4 TQueryText.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const UnicodeString Value);
```

Description

This is SetText, a member of class TQueryText.

52.1.39.4 TQueryText Properties

The properties of the TQueryText class are listed here.

Published Properties

	Name	Description
	Caption (see page 5453)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5454)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5454)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5455)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5455)	Specifies the horizontal size of the generated control.
	FieldName (see page 5455)	Specifies the name of the field whose value the generated control represents.
	MaxWidth (see page 5456)	Limits the length of text users can type into the control.
	ParamName (see page 5456)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5457)	Indicates whether end users can use the generated control to set values.
	Style (see page 5457)	Adds style attributes to the generated display control.
	StyleRule (see page 5458)	This is StyleRule, a member of class TQueryText.
	TabIndex (see page 5458)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 5458)	Specifies the initial value of the generated control.

52.1.39.4.1 TQueryText.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.39.4.2 TQueryText.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.39.4.3 TQueryText.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.39.4.4 TQueryText.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.39.4.5 TQueryText.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

52.1.39.4.6 TQueryText.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.39.4.7 TQueryText.MaxWidth Property

Limits the length of text users can type into the control.

Pascal

```
property MaxWidth: Integer;
```

C++

```
__property int MaxWidth;
```

Description

Use MaxWidth to limit the length of strings (in characters) that users can type into the generated control.

Note: MaxWidth has no effect on the size of the generated control. To limit the size of the control, use the DisplayWidth (see page 5455) property.

See Also

DisplayWidth (see page 5594)

52.1.39.4.8 TQueryText.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;__property UnicodeString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 5539)

IsQueryField (see page 5528)

Params

xmlbroker (see page 5631)

Setting Web Item Properties

52.1.39.4.9 TQueryText.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

52.1.39.4.10 TQueryText.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.39.4.11 TQueryText.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TQueryText.

52.1.39.4.12 TQueryText.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.39.4.13 TQueryText.Text Property

Specifies the initial value of the generated control.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text to initialize the value that appears in the generated display control.

See Also

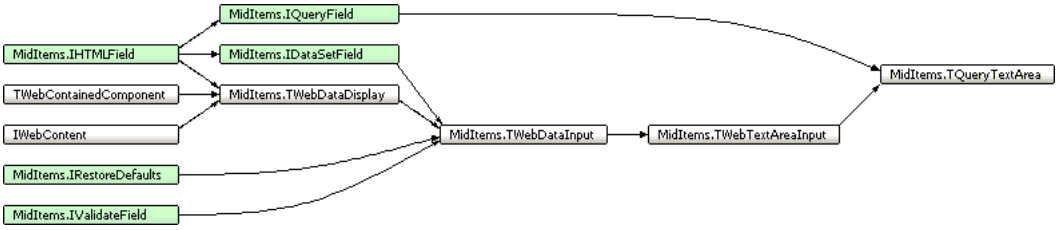
FieldName (see page 5539)

ParamName (see page 5540)

52.1.40 TQueryTextArea Class

TQueryTextArea creates a multi-line HTML input control to represent named parameter.

Class Hierarchy



Pascal

```
TQueryTextArea = class(TWebTextAreaInput, IQueryField);
```

C++

```
class TQueryTextArea : public TWebTextAreaInput, public IQueryField;
```

File

MidItems

Description

Add TQueryTextArea as the child of a TQueryFieldGroup (see page 5415) component to add a multi-line edit control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 5415) represent the parameters of an XML broker. However, TQueryTextArea can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryTextArea to

Specify the field or parameter whose value the control displays.

Assign an initial value to the control.

Specify a caption that describes the value of the control.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 5415)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

TXMLBroker

52.1.40.1 TQueryTextArea Members

The following tables list the members exposed by TQueryTextArea.

Protected Methods

	Name	Description
	AddAttributes (see page 5462)	This is AddAttributes, a member of class TQueryTextArea.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5462)	This is ControlContent, a member of class TQueryTextArea.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.

	EventContent (see page 5587)	This is EventContent, a member of class TWebTextAreaInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetText (see page 5462)	This is GetText, a member of class TQueryTextArea.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 5587)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5587)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetLines (see page 5463)	This is SetLines, a member of class TQueryTextArea.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetText (see page 5463)	This is SetText, a member of class TQueryTextArea.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
	Create (see page 5461)	Creates and initializes an instance of TQueryTextArea (see page 5458).
	Destroy (see page 5461)	Frees an instance of TQueryTextArea (see page 5458).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5463)	Indicates whether the component generates a control that can appear in a query form.

	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Text (see page 5663)	This is Text, a member of class IQueryField.

Published Properties

	Name	Description
	Caption (see page 5464)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5464)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5465)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5465)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5466)	Specifies the height of the generated control.
	DisplayWidth (see page 5466)	Specifies the horizontal size of the generated control.
	FieldName (see page 5466)	Specifies the name of the field whose value the generated control represents.
	Lines (see page 5467)	Specifies the initial value that appears in the multi-line edit control.
	ParamName (see page 5467)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5468)	Indicates whether end users can use the generated control to set values.
	Style (see page 5468)	Adds style attributes to the generated display control.
	StyleRule (see page 5468)	This is StyleRule, a member of class TQueryTextArea.
	TabIndex (see page 5469)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5469)	Specifies how the generated control treats lines that are longer than the display width.

52.1.40.2 TQueryTextArea.Create Constructor

Creates and initializes an instance of TQueryTextArea (see page 5458).

Pascal

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

C++

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

Description

Call Create to instantiate TQueryTextArea (see page 5458) at runtime. TQueryTextArea (see page 5458) components added at design time using the Web Page editor are instantiated automatically.

52.1.40.3 TQueryTextArea.Destroy Destructor

Frees an instance of TQueryTextArea (see page 5458).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TQueryTextArea();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free

52.1.40.4 TQueryTextArea Methods

The methods of the TQueryTextArea class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 5462)	This is AddAttributes, a member of class TQueryTextArea.
	ControlContent (see page 5462)	This is ControlContent, a member of class TQueryTextArea.
	GetText (see page 5462)	This is GetText, a member of class TQueryTextArea.
	SetLines (see page 5463)	This is SetLines, a member of class TQueryTextArea.
	SetText (see page 5463)	This is SetText, a member of class TQueryTextArea.

Public Methods

	Name	Description
	IsQueryField (see page 5463)	Indicates whether the component generates a control that can appear in a query form.

52.1.40.4.1 TQueryTextArea.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TQueryTextArea.

52.1.40.4.2 TQueryTextArea.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TQueryTextArea.

52.1.40.4.3 TQueryTextArea.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall UnicodeString GetText();
```

Description

This is GetText, a member of class TQueryTextArea.

52.1.40.4.4 TQueryTextArea.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 5421)). IsQueryField returns true, because TQueryTextArea (see page 5458) is designed specifically for TQueryFieldGroup (see page 5415) to use.

See Also

IsColumn (see page 5527)

TQueryFieldGroup (see page 5415)

TQueryForm (see page 5421)

52.1.40.4.5 TQueryTextArea.SetLines Method

Pascal

```
procedure SetLines(const Value: TStrings);
```

C++

```
__fastcall SetLines(const TStrings Value);
```

Description

This is SetLines, a member of class TQueryTextArea.

52.1.40.4.6 TQueryTextArea.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const UnicodeString Value);
```

Description

This is SetText, a member of class TQueryTextArea.

52.1.40.5 TQueryTextArea Properties

The properties of the TQueryTextArea class are listed here.

Published Properties

	Name	Description
	Caption (see page 5464)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5464)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5465)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5465)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5466)	Specifies the height of the generated control.
	DisplayWidth (see page 5466)	Specifies the horizontal size of the generated control.
	FieldName (see page 5466)	Specifies the name of the field whose value the generated control represents.
	Lines (see page 5467)	Specifies the initial value that appears in the multi-line edit control.
	ParamName (see page 5467)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 5468)	Indicates whether end users can use the generated control to set values.
	Style (see page 5468)	Adds style attributes to the generated display control.
	StyleRule (see page 5468)	This is StyleRule, a member of class TQueryTextArea.
	TabIndex (see page 5469)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5469)	Specifies how the generated control treats lines that are longer than the display width.

52.1.40.5.1 TQueryTextArea.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.40.5.2 TQueryTextArea.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are

added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.40.5.3 TQueryTextArea.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.40.5.4 TQueryTextArea.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.40.5.5 TQueryTextArea.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 5588)

52.1.40.5.6 TQueryTextArea.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 5588)

52.1.40.5.7 TQueryTextArea.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.40.5.8 TQueryTextArea.Lines Property

Specifies the initial value that appears in the multi-line edit control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to initialize the value that appears in the generated display control.

Each string in the TStrings object designates a line in the text value. This may affect the formatting, depending on the value of Wrap (see page 5469).

See Also

FieldName (see page 5539)

ParamName (see page 5540)

Wrap (see page 5588)

52.1.40.5.9 TQueryTextArea.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;__property UnicodeString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 5539)

IsQueryField (see page 5528)

Params

xmlbroker (see page 5631)

Setting Web Item Properties

52.1.40.5.10 TQueryTextArea.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

52.1.40.5.11 TQueryTextArea.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.40.5.12 TQueryTextArea.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TQueryTextArea.

52.1.40.5.13 TQueryTextArea.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.40.5.14 TQueryTextArea.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 5588)

52.1.41 TResetQueryButton Class

TResetQueryButton generates an HTML control that backs out changes to the values on a query form.

Class Hierarchy

Pascal

```
TResetQueryButton = class(TQueryButton);
```

C++

```
class TResetQueryButton : public TQueryButton;
```

File

MidItems

Description

Add TResetQueryButton as the child of a TQueryButtons (see page 5412) component (or a layout group that is a child of a TQueryButtons (see page 5412) component) to enable the generated HTML form to undo edits made in the query form's display fields. The resulting HTML button's action backs out all edits to the values displayed on the query form.

Use the properties of TResetQueryButton to customize the appearance of the button.

See Also

TXMLBroker

TQueryForm (see page 5421)

TQueryButtons (see page 5412)

TLayoutGroup (see page 5389)

52.1.41.1 TResetQueryButton Members

The following tables list the members exposed by TResetQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5405)	This is DefaultCaption, a member of class TQueryButton.
	InputType (see page 5405)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	EventContent (see page 5471)	This is EventContent, a member of class TResetQueryButton.
V	GetCaption (see page 5406)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
V	GetHTMLControlName (see page 5406)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 5407)	This is GetHTMLForm, a member of class TQueryButton.
V	GetInputType (see page 5407)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetXMLComponent (see page 5407)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 5407)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5408)	Forwards notification messages to all owned components.
	ParentXMLComponent (see page 5408)	This is ParentXMLComponent, a member of class TQueryButton.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetXMLComponent (see page 5408)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 5409)	This is SetXMLUseParent, a member of class TQueryButton.

Public Methods

	Name	Description
	Create (see page 5471)	Creates and initializes an instance of TResetQueryButton (see page 5469).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.

Published Properties

	Name	Description
	Caption (see page 5409)	Specifies the caption that appears on the button.
	Custom (see page 5409)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5410)	Adds style attributes to the generated button control.
	StyleRule (see page 5410)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5411)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 5411)	Specifies whether the button is associated with its parent's query form.

52.1.41.2 TResetQueryButton.Create Constructor

Creates and initializes an instance of TResetQueryButton (see page 5469).

Pascal

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

C++

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

Description

Call Create to instantiate TResetQueryButton (see page 5469) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.41.3 TResetQueryButton Methods

The methods of the TResetQueryButton class are listed here.

Protected Methods

	Name	Description
	EventContent (see page 5471)	This is EventContent, a member of class TResetQueryButton.

52.1.41.3.1 TResetQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

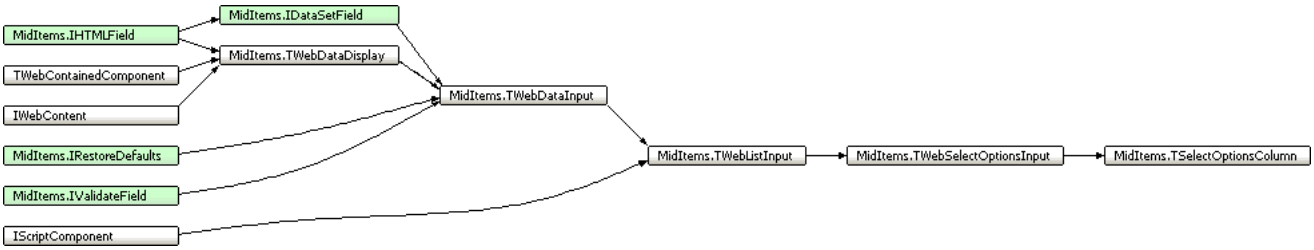
Description

This is EventContent, a member of class TResetQueryButton.

52.1.42 TSelectOptionsColumn Class

TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.

Class Hierarchy



Pascal

```
TSelectOptionsColumn = class(TWebSelectOptionsInput);
```

C++

```
class TSelectOptionsColumn : public TWebSelectOptionsInput;
```

File

MidItems

Description

Add TSelectOptionsColumn as the child of a TDataGrid (see page 5302) component to add a list control as the display control for a column. TSelectOptionsColumn represents the value of a field in the TDataGrid (see page 5302)'s associated dataset.

Use the properties and methods of TSelectOptionsColumn to

Specify the field whose value the list box displays.

Specify the possible values for the field from which the user can choose.

Specify the caption that appears at the top of the column.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TDataGrid (see page 5302)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.42.1 TSelectOptionsColumn Members

The following tables list the members exposed by TSelectOptionsColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 5576)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5577)	This is EventContent, a member of class TWebSelectOptionsInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	FormatInputs (see page 5577)	This is FormatInputs, a member of class TWebSelectOptionsInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSelectIndex (see page 5577)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
V	Create (see page 5576)	Creates and initializes an instance of TWebSelectOptionsInput (see page 5573).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
V	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5475)	Indicates whether the component generates a control that can appear in the column of a data grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5475)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5476)	Customizes the appearance of the generated control's label.
	Custom (see page 5476)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5477)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5477)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5478)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5478)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5478)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Style (see page 5479)	Adds style attributes to the generated display control.
	StyleRule (see page 5479)	This is StyleRule, a member of class TSelectOptionsColumn.
	TabIndex (see page 5480)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5480)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5480)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.42.2 TSelectOptionsColumn Methods

The methods of the TSelectOptionsColumn class are listed here.

Public Methods

	Name	Description
	IsColumn (see page 5475)	Indicates whether the component generates a control that can appear in the column of a data grid.

52.1.42.2.1 TSelectOptionsColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 5302)). IsColumn returns true, because TSelectOptionsColumn (see page 5472) is designed specifically for TDataGrid (see page 5302) to use.

See Also

IsColumn (see page 5527)

TDataGrid (see page 5302)

52.1.42.3 TSelectOptionsColumn Properties

The properties of the TSelectOptionsColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 5475)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5476)	Customizes the appearance of the generated control's label.
	Custom (see page 5476)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5477)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5477)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5478)	Specifies the name of the field whose value the generated control represents.
	Items (see page 5478)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5478)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Style (see page 5479)	Adds style attributes to the generated display control.
	StyleRule (see page 5479)	This is StyleRule, a member of class TSelectOptionsColumn.
	TabIndex (see page 5480)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5480)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5480)	Indicates which field in DataSet supplies the possible values for the generated control.

52.1.42.3.1 TSelectOptionsColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.42.3.2 TSelectOptionsColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.42.3.3 TSelectOptionsColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.42.3.4 TSelectOptionsColumn.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.42.3.5 TSelectOptionsColumn.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 5565)

Items (see page 5565)

52.1.42.3.6 TSelectOptionsColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.42.3.7 TSelectOptionsColumn.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.42.3.8 TSelectOptionsColumn.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.42.3.9 TSelectOptionsColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.42.3.10 TSelectOptionsColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TSelectOptionsColumn.

52.1.42.3.11 TSelectOptionsColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.42.3.12 TSelectOptionsColumn.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.42.3.13 TSelectOptionsColumn.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

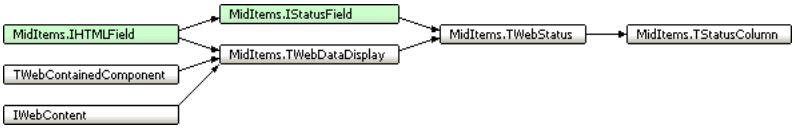
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.43 TStatusColumn Class

TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.

Class Hierarchy



Pascal

```
TStatusColumn = class(TWebStatus);
```

C++

```
class TStatusColumn : public TWebStatus;
```

File

MidItems

Description

Add TStatusColumn as the child of a TDataGrid (see page 5302) component to provide feedback about whether records in the grid have been modified. The generated control displays an M if the user has edited a record, an I if the user has inserted the record, and blank if the user has made no changes to the record.

Use the properties and methods of TStatusColumn to

Specify the caption that appears at the top of the column.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TDataGrid (see page 5302)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.43.1 TStatusColumn Members

The following tables list the members exposed by TStatusColumn.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
A	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 5580)	This is ControlContent, a member of class TWebStatus.
V	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
V	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
V	GetCaption (see page 5581)	Returns the value of the Caption property.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5581)	Returns the name that other components can use to refer to the generated display control.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5580)	Creates and initializes an instance of TWebStatus (see page 5578).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	Identifier (see page 5581)	Returns the name under which record status information is stored in the XML data packet.
V	IsColumn (see page 5483)	Indicates whether the component generates a control that can appear in the column of a data grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Public Properties

	Name	Description
	CaptionPosition (see page 5583)	Specifies where the label appears relative to the generated display control.
	DisplayWidth (see page 5583)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

Published Properties

	Name	Description
	Caption (see page 5484)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5484)	Customizes the appearance of the generated control's label.
	Custom (see page 5484)	Represents a tag option and its value in the tag for the display control.
	Style (see page 5485)	Adds style attributes to the generated display control.
	StyleRule (see page 5485)	This is StyleRule, a member of class TStatusColumn.

52.1.43.2 TStatusColumn Methods

The methods of the TStatusColumn class are listed here.

Public Methods

	Name	Description
	IsColumn (see page 5483)	Indicates whether the component generates a control that can appear in the column of a data grid.

52.1.43.2.1 TStatusColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web item generates a control that represents a column in a data grid (TDataGrid (see page 5302)). IsColumn returns true, because TStatusColumn (see page 5481) is designed specifically for TDataGrid (see page 5302) to use.

See Also

IsColumn (see page 5527)

TDataGrid (see page 5302)

52.1.43.3 TStatusColumn Properties

The properties of the TStatusColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 5484)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5484)	Customizes the appearance of the generated control's label.
	Custom (see page 5484)	Represents a tag option and its value in the tag for the display control.
	Style (see page 5485)	Adds style attributes to the generated display control.
	StyleRule (see page 5485)	This is StyleRule, a member of class TStatusColumn.

52.1.43.3.1 TStatusColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.43.3.2 TStatusColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.43.3.3 TStatusColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator

key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.43.3.4 TStatusColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.43.3.5 TStatusColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TStatusColumn.

52.1.44 TSubmitQueryButton Class

TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.

Class Hierarchy

Pascal

```
TSubmitQueryButton = class(TQueryButton);
```

C++

```
class TSubmitQueryButton : public TQueryButton;
```

File

MidItems

Description

Add TSubmitQueryButton as the child of a TQueryButtons (see page 5412) component (or a layout group that is a child of a TQueryButtons (see page 5412) component) to enable the generated HTML form to send the values entered in the query form back to the Web application. When the user presses the generated button, the values in the query form are sent to the Web application, using the path info specified by the query form's Action property.

Use the properties of TSubmitQueryButton to customize the appearance of the button.

See Also

TXMLBroker

TQueryForm (see page 5421)

TQueryButtons (see page 5412)

TLayoutGroup (see page 5389)

TCustomQueryForm (see page 5293)

52.1.44.1 TSubmitQueryButton Members

The following tables list the members exposed by TSubmitQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5405)	This is DefaultCaption, a member of class TQueryButton.
	InputType (see page 5405)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	EventContent (see page 5488)	This is EventContent, a member of class TSubmitQueryButton.
V	GetCaption (see page 5406)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
V	GetHTMLControlName (see page 5406)	This is GetHTMLControlName, a member of class TQueryButton.

	GetHTMLForm (see page 5407)	This is GetHTMLForm, a member of class TQueryButton.
V	GetInputType (see page 5407)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetXMLComponent (see page 5407)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 5407)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5408)	Forwards notification messages to all owned components.
	ParentXMLComponent (see page 5408)	This is ParentXMLComponent, a member of class TQueryButton.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetXMLComponent (see page 5408)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 5409)	This is SetXMLUseParent, a member of class TQueryButton.

Public Methods

	Name	Description
V	Create (see page 5487)	Creates and initializes an instance of TSubmitQueryButton (see page 5486).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.

Published Properties

	Name	Description
	Caption (see page 5409)	Specifies the caption that appears on the button.
	Custom (see page 5409)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5410)	Adds style attributes to the generated button control.
	StyleRule (see page 5410)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5411)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 5411)	Specifies whether the button is associated with its parent's query form.

52.1.44.2 TSubmitQueryButton.Create Constructor

Creates and initializes an instance of TSubmitQueryButton (see page 5486).

Pascal

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

C++

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

Description

Call Create to instantiate TSubmitQueryButton (see page 5486) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.44.3 TSubmitQueryButton Methods

The methods of the TSubmitQueryButton class are listed here.

Protected Methods

	Name	Description
	EventContent (see page 5488)	This is EventContent, a member of class TSubmitQueryButton.

52.1.44.3.1 TSubmitQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

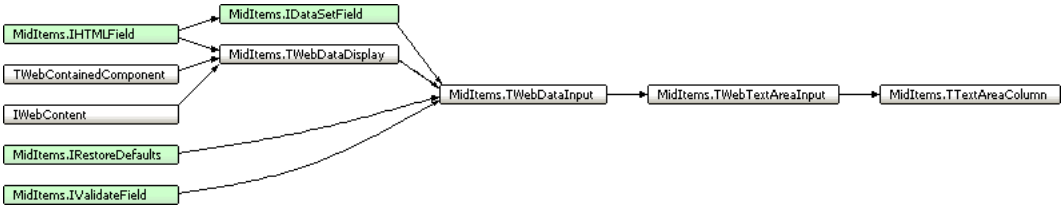
Description

This is EventContent, a member of class TSubmitQueryButton.

52.1.45 TTextAreaColumn Class

TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.

Class Hierarchy



Pascal

```
TTextAreaColumn = class(TWebTextAreaInput);
```

C++

```
class TTextAreaColumn : public TWebTextAreaInput;
```

File

MidItems

Description

Add TTextAreaColumn as the child of a TDataGrid (see page 5302) component to add a multi-line edit control as the display control for a column. TTextAreaColumn represents the value of a field in the TDataGrid (see page 5302)'s associated dataset.

Use the properties and methods of TTextAreaColumn to

Specify the field whose value the control displays.

Specify the caption that appears at the top of the column.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TQueryTextArea (see page 5458)

52.1.45.1 TTextAreaColumn Members

The following tables list the members exposed by TTextAreaColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 5586)	This is AddAttributes, a member of class TWebTextAreaInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5586)	This is ControlContent, a member of class TWebTextAreaInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	EventContent (see page 5587)	This is EventContent, a member of class TWebTextAreaInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 5587)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5587)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5586)	Creates and initializes an instance of TWebTextAreaInput (see page 5583).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5491)	Indicates whether the component generates a control that can appear in the column of a data grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5491)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5492)	Customizes the appearance of the generated control's label.
	Custom (see page 5492)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5493)	Specifies the height of the generated control.
	DisplayWidth (see page 5493)	Specifies the horizontal size of the generated control.
	FieldName (see page 5493)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5494)	Indicates whether end users can use the generated control to set values.
	Style (see page 5494)	Adds style attributes to the generated display control.
	StyleRule (see page 5494)	This is StyleRule, a member of class TTextAreaColumn.
	TabIndex (see page 5495)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5495)	Specifies how the generated control treats lines that are longer than the display width.

52.1.45.2 TTextAreaColumn Methods

The methods of the TTextAreaColumn class are listed here.

Public Methods

	Name	Description
V	IsColumn (see page 5491)	Indicates whether the component generates a control that can appear in the column of a data grid.

52.1.45.2.1 TTextAreaColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 5302)). IsColumn returns true, because TTextAreaColumn (see page 5488) is designed specifically for TDataGrid (see page 5302) to use.

See Also

IsColumn (see page 5527)

TDataGrid (see page 5302)

52.1.45.3 TTextAreaColumn Properties

The properties of the TTextAreaColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 5491)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5492)	Customizes the appearance of the generated control's label.
	Custom (see page 5492)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5493)	Specifies the height of the generated control.
	DisplayWidth (see page 5493)	Specifies the horizontal size of the generated control.
	FieldName (see page 5493)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5494)	Indicates whether end users can use the generated control to set values.
	Style (see page 5494)	Adds style attributes to the generated display control.
	StyleRule (see page 5494)	This is StyleRule, a member of class TTextAreaColumn.
	TabIndex (see page 5495)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5495)	Specifies how the generated control treats lines that are longer than the display width.

52.1.45.3.1 TTextAreaColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.45.3.2 TTextAreaColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.45.3.3 TTextAreaColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.45.3.4 TTextAreaColumn.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 5588)

52.1.45.3.5 TTextAreaColumn.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 5588)

52.1.45.3.6 TTextAreaColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.45.3.7 TTextAreaColumn.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

52.1.45.3.8 TTextAreaColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.45.3.9 TTextAreaColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TTextAreaColumn.

52.1.45.3.10 TTextAreaColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.45.3.11 TTextAreaColumn.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

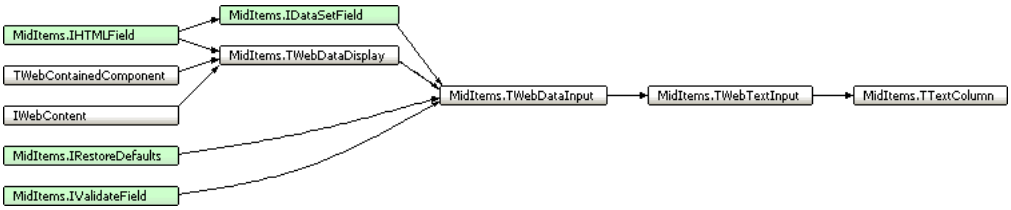
See Also

DisplayWidth (see page 5588)

52.1.46 TTextColumn Class

TTextColumn creates a single-line HTML input control to represent a field value in a data grid.

Class Hierarchy



Pascal

```
TTextColumn = class(TWebTextInput);
```

C++

```
class TTextColumn : public TWebTextInput;
```

File

MidItems

Description

Add TTextColumn as the child of a TDataGrid (see page 5302) component to add an edit control as the display control for a column. TTextColumn represents the value of a field in the TDataGrid (see page 5302)'s associated dataset.

Use the properties and methods of TTextColumn to

Specify the field whose value the control displays.

Specify the caption that appears at the top of the column.

Customize the appearance of the edit control.

Fit the control into the tab order of the generated HTML form.

See Also

TDataGrid (see page 5302)

TXMLDisplayGroup (see page 5620)

TXMLDisplayGroup (see page 5620)

52.1.46.1 TTextColumn Members

The following tables list the members exposed by TTextColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 5592)	This is AddAttributes, a member of class TWebTextInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5592)	This is ControlContent, a member of class TWebTextInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.

	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 5593)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5593)	This is RestoreFieldDefaults, a member of class TWebTextInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Create (see page 5592)	Creates and initializes an instance of TWebTextInput (see page 5589).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5498)	Indicates whether the component generates a control that can appear in the column of a data grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

Published Properties

	Name	Description
	Caption (see page 5499)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5499)	Customizes the appearance of the generated control's label.
	Custom (see page 5500)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5500)	Specifies the horizontal size of the generated control.
	FieldName (see page 5500)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5501)	Indicates whether end users can use the generated control to set values.
	Style (see page 5501)	Adds style attributes to the generated display control.
	StyleRule (see page 5502)	This is StyleRule, a member of class TTextColumn.
	TabIndex (see page 5502)	Determines the position of the generated control in the HTML form's tab order.

52.1.46.2 TTextColumn Methods

The methods of the TTextColumn class are listed here.

Public Methods

	Name	Description
	IsColumn (see page 5498)	Indicates whether the component generates a control that can appear in the column of a data grid.

52.1.46.2.1 TTextColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 5302)). IsColumn returns true, because TTextColumn (see page 5495) is designed specifically for TDataGrid (see page 5302) to use.

See Also

IsColumn (see page 5527)

TDataGrid (see page 5302)

52.1.46.3 TTextColumn Properties

The properties of the TTextColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 5499)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5499)	Customizes the appearance of the generated control's label.
	Custom (see page 5500)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5500)	Specifies the horizontal size of the generated control.
	FieldName (see page 5500)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 5501)	Indicates whether end users can use the generated control to set values.
	Style (see page 5501)	Adds style attributes to the generated display control.
	StyleRule (see page 5502)	This is StyleRule, a member of class TTextColumn.
	TabIndex (see page 5502)	Determines the position of the generated control in the HTML form's tab order.

52.1.46.3.1 TTextColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.46.3.2 TTextColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.46.3.3 TTextColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.46.3.4 TTextColumn.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

52.1.46.3.5 TTextColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.46.3.6 TTextColumn.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

52.1.46.3.7 TTextColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.46.3.8 TTextColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TTextColumn.

52.1.46.3.9 TTextColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

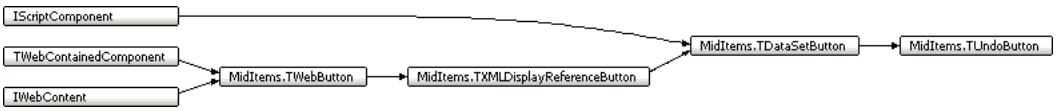
Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.47 TUndoButton Class

TUndoButton generates an HTML control that backs out changes to the current record.

Class Hierarchy



Pascal

```
TUndoButton = class(TDataSetButton);
```

C++

```
class TUndoButton : public TDataSetButton;
```

File

MidItems

Description

Add TUndoButton as the child of a TDataNavigator (see page 5307) component (or a layout group that is a child of a TDataNavigator (see page 5307) component) to enable the generated HTML form to undo edits made in an associated display control or controls. The resulting HTML button's action backs out all edits to the current record.

Use the properties of TUndoButton to customize the appearance of the button and associate it with a Web item that generates

the display controls.

See Also

TXMLBroker

TDataNavigator (see page 5307)

TLayoutGroup (see page 5389)

52.1.47.1 TUndoButton Members

The following tables list the members exposed by TUndoButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 5313)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 5313)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	AddElements (see page 5314)	This is AddElements, a member of class TDataSetButton.
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5314)	Returns the value of the Caption property.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 5314)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 5314)	This is GetSubComponents, a member of class TDataSetButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 5315)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 5315)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5504)	Creates and initializes an instance of TUndoButton (see page 5502).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.

	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 5316)	Specifies the caption that appears on the button.
	Custom (see page 5316)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5316)	Adds style attributes to the generated button control.
	StyleRule (see page 5317)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5317)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5318)	Specifies whether the button is associated with its parent's display object.

52.1.47.2 TUndoButton.Create Constructor

Creates and initializes an instance of TUndoButton (see page 5502).

Pascal

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

C++

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

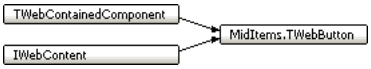
Description

Call Create to instantiate TUndoButton (see page 5502) at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.48 TWebButton Class

TWebButton is the base class for Web items that create HTML button controls.

Class Hierarchy



Pascal

```
TWebButton = class(TWebContainedComponent, IWebContent);
```

C++

```
class TWebButton : public TWebContainedComponent, public IWebContent;
```

File

MidItems

Description

Use TWebButton as a base class for components that generate the HTML statements defining a specialized button control in a WebSnap or InternetExpress application. TWebButton introduces the properties and methods for

Generating the HTML code to define the button control.

Defining a button caption.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

In order to allow other Web items to use the HTML generated by the button control, TWebButton implements the IWebContent interface.

Do not create instances of TWebButton. TWebButton relies on an abstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TWebButton that represent buttons with specific purposes.

See Also

TDeleteButton (see page 5318)

TFirstButton (see page 5363)

TInsertButton (see page 5385)

TLastButton (see page 5387)

TNextButton (see page 5392)

TNextPageButton (see page 5394)

TPostButton (see page 5396)

TPriorButton (see page 5399)

TPriorPageButton (see page 5401)

TUndoButton (see page 5502)

TResetQueryButton (see page 5469)

TSubmitQueryButton (see page 5486)

TApplyUpdatesButton (see page 5250)

IWebContent

52.1.48.1 TWebButton Members

The following tables list the members exposed by TWebButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.

Public Methods

	Name	Description
V	Create (see page 5506)	Creates and initializes an instance of TWebButton (see page 5504).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.

	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.

52.1.48.2 TWebButton.Create Constructor

Creates and initializes an instance of TWebButton (see page 5504).

Pascal

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

C++

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

Description

Call Create to instantiate a Web button at runtime. Web buttons added at design time using the Web Page editor are instantiated automatically.

52.1.48.3 TWebButton Methods

The methods of the TWebButton class are listed here.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.

52.1.48.3.1 TWebButton.Content Method

Returns the HTML that defines the button control.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Content implements the IWebContent method of the same name. This method allows the Web button's parent to extract the HTML defining the button.

Content calls the virtual ImplContent (see page 5507) method to provide the content it returns. Each TWebButton (see page 5504) descendant overrides ImplContent (see page 5507) to generate the definition of its particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

Content returns a string of HTML defining the button control.

See Also

ImplContent (see page 5507)

IWebContent

ILayoutWebContent

52.1.48.3.2 TWebButton.GetCaption Method

Returns the value of the Caption (see page 5508) property.

Pascal

```
function GetCaption: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption (see page 5508) property.

See Also

Caption (see page 5508)

SetCaption (see page 5508)

52.1.48.3.3 TWebButton.GetLayoutAttributes Method

Returns the layout object that specifies how the generated control should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 5506) method) to take into account layout properties of the child component.

As implemented by TWebButton (see page 5504), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout of buttons does not make use of layout attributes.

See Also

Content (see page 5506)

52.1.48.3.4 TWebButton.ImplContent Method

Provides the underlying implementation for the Content (see page 5506) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout Layout) = 0;
```

Description

The Content (see page 5506) method calls ImplContent to generate the HTML that defines the button control. TWebButton (see page 5504) introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton (see page 5504) descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 5506)

ILayoutWebContent

52.1.48.3.5 TWebButton.SetCaption Method

Sets the value of the Caption (see page 5508) property.

Pascal

```
procedure SetCaption(const AValue: string); virtual;
```

C++

```
virtual __fastcall SetCaption(const UnicodeString AValue);
```

Description

SetCaption is the protected write implementation for the Caption (see page 5508) property.

See Also

Caption (see page 5508)

GetCaption (see page 5507)

52.1.48.4 TWebButton Properties

The properties of the TWebButton class are listed here.

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.

52.1.48.4.1 TWebButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 5509)

GetCaption (see page 5507)

SetCaption (see page 5508)

52.1.48.4.2 TWebButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 5508)

Style (see page 5509)

52.1.48.4.3 TWebButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 5509)

StyleRule (see page 5510)

52.1.48.4.4 TWebButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style (see page 5509) property.

See Also

Style (see page 5509)

Styles (see page 5716)

StylesFile (see page 5717)

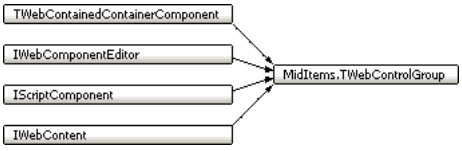
Styles

StylesFile

52.1.49 TWebControlGroup Class

TWebControlGroup is the base class for Web items that arrange HTML controls.

Class Hierarchy



Pascal

```
TWebControlGroup = class(TWebContainedContainerComponent, IWebComponentEditor,
IScriptComponent, IWebContent);
```

C++

```
class TWebControlGroup : public TWebContainedContainerComponent, public IWebComponentEditor,
public IScriptComponent, public IWebContent;
```

File

MidItems

Description

Use TWebControlGroup as a base class for components that generate organized portions of the HTML document containing one or more related HTML controls. TWebControlGroup introduces the properties and methods for managing a collection of Web items that generate the HTML controls that make up the group.

In order to let other Web items to use the HTML that defines the organized collection of controls, TWebControlGroup implements the IWebContent interface.

Do not create instances of TWebControlGroup. TWebControlGroup manages the Web items that generate HTML defining controls, but does not combine the generated content into any content of its own. Instead, use a descendant of TWebControlGroup for a set of specialized controls. These descendants are listed in the See also list.

See Also

TDataNavigator (see page 5307)

TQueryButtons (see page 5412)

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

IWebContent

TAdapterFieldGroup

TAdapterGrid

TAdapterCommandColumn

TAdapterForm

52.1.49.1 TWebControlGroup Members

The following tables list the members exposed by TWebControlGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Methods

	Name	Description
	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).

Public Properties

	Name	Description
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

52.1.49.2 TWebControlGroup.Create Constructor

Creates and initializes an instance of TWebControlGroup (see page 5510).

Pascal

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

C++

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

Description

Call Create to instantiate TWebControlGroup (see page 5510) at runtime. TWebControlGroup (see page 5510) descendants added at design time are instantiated automatically.

52.1.49.3 TWebControlGroup.Destroy Destructor

Frees an instance of TWebControlGroup (see page 5510).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebControlGroup();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

52.1.49.4 TWebControlGroup Methods

The methods of the TWebControlGroup class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.

	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.

52.1.49.4.1 TWebControlGroup.AddElements Method

Used internally to generate any script declarations needed by the group of controls.

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

The page producer calls AddElements when it generates the block of declarations for the script language it uses. This method allows the generated group of controls to use classes and methods in the scripting language of the page producer.

AddElements calls on the virtual ImplAddElements (see page 5516) method, which uses the provided interface to return its declarations. Each TWebControlGroup (see page 5510) descendant overrides ImplAddElements (see page 5516) to add any declarations it needs.

See Also

ImplAddElements (see page 5516)

TAdapterPageProducer

TInetXPageProducer (see page 5730)

52.1.49.4.2 TWebControlGroup.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the TWebControlGroup (see page 5510) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 5516) method to determine whether a class of Web item is valid.

AParent is the TWebControlGroup (see page 5510) descendant to whose WebFieldControls (see page 5519) property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the TWebControlGroup (see page 5510) descendant can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 5516)

WebFieldControls (see page 5519)

52.1.49.4.3 TWebControlGroup.ChangeFieldControls Method

Responds when the group of HTML controls changes.

Pascal

```
procedure ChangeFieldControls(Sender: TObject);
```

C++

```
__fastcall ChangeFieldControls(TObject * Sender);
```

Description

ChangeFieldControls is called automatically when the list of Web items that generate HTML controls in this group changes. (It is assigned as the OnChange event handler to the TWebComponentList object that implements the WebFieldControls (see page 5519) property.)

ChangeFieldControls sets the DefaultWebComponents (see page 5518) property to nil (Delphi) or NULL (C++) when an application explicitly assigns child items to the WebFieldControls (see page 5519) property.

See Also

OnChange

WebFieldControls (see page 5519)

DefaultWebComponents (see page 5518)

DefaultWebComponents (see page 5518)

52.1.49.4.4 TWebControlGroup.Content Method

Returns the HTML that defines the group of HTML controls.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Content implements the IWebContent method of the same name. This method allows the TWebControlGroup (see page 5510) object's parent to extract the HTML that defines and lays out a set of controls.

Content calls the virtual ImplContent (see page 5517) method to provide the HTML it returns. Each TWebControlGroup (see page 5510) descendant overrides ImplContent (see page 5517) to generate its particular group of controls. The generated HTML can make use of any script declarations supplied by the AddElements (see page 5513) method.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML it generates. It implements the ILayoutWebContent interface.

Content returns a string of HTML defining the group of controls.

See Also

ImplContent (see page 5517)

IWebContent

AddElements (see page 5513)

ILayoutWebContent

52.1.49.4.5 TWebControlGroup.GetDefaultWebComponents Method

Returns the value of the DefaultWebComponents (see page 5518) property.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents returns the value of DefaultWebComponents (see page 5518). When no Web items have explicitly been added to the control group, this is a list of controls that appear by default. Once an application explicitly assigns controls to the WebFieldControls (see page 5519) property, GetDefaultWebComponents returns nil (Delphi) or NULL (C++).

See Also

WebFieldControls (see page 5519)

DefaultWebComponents (see page 5518)

SetDefaultWebComponents (see page 5518)

52.1.49.4.6 TWebControlGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 5514) method) to take into account layout properties of the child component.

As implemented by TWebControlGroup (see page 5510), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup (see page 5510) descendants not make use of layout attributes.

See Also

Content (see page 5514)

52.1.49.4.7 TWebControlGroup.GetSubComponents Method

Returns the value of the WebFieldControls (see page 5519) property.

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

GetSubComponents returns the TWebControlGroup (see page 5510)'s list of Web items that it uses to generate HTML content. This method is part of the mechanism that fits TWebControlGroup (see page 5510) into a hierarchy of Web items, each of which generates part of a larger HTML document.

See Also

WebFieldControls (see page 5519)

ImplGetSubComponents (see page 5517)

52.1.49.4.8 TWebControlGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements (see page 5513) method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements (see page 5513) method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup (see page 5510), ImplAddElements does nothing. TWebControlGroup (see page 5510) descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.49.4.9 TWebControlGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5513) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5513) method calls ImpleCanAddClass to determine whether the TWebControlGroup (see page 5510) descendant can add a component of type AClass to its WebComponents property list. AParent is the

TWebControlGroup (see page 5510) instance to which the component will be added.

In TWebControlGroup (see page 5510), ImplCanAddClass always returns true. TWebControlGroup (see page 5510) descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 5513)

WebFieldControls (see page 5519)

52.1.49.4.10 TWebControlGroup.ImplContent Method

Provides the underlying implementation for the Content (see page 5514) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content (see page 5514) method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup (see page 5510), ImplContent returns an empty string. TWebControlGroup (see page 5510) descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup (see page 5510) should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 5514)

AddElements (see page 5513)

ILayoutWebContent

52.1.49.4.11 TWebControlGroup.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents (see page 5516) method.

Pascal

```
function ImplGetSubComponents: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents (see page 5516) method calls ImplGetSubComponents to retrieve the value of WebFieldControls (see page 5519) as a TObject. TWebControlGroup (see page 5510) descendants override this method if the list is not simply the value of WebFieldControls (see page 5519). For example, some TWebControlGroup (see page 5510) descendants supply a default value (DefaultWebComponents (see page 5518)) when the WebFieldControls (see page 5519) property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 5516)

DefaultWebComponents (see page 5518)

WebFieldControls (see page 5519)

52.1.49.4.12 TWebControlGroup.SetDefaultWebComponents Method

Sets the value of the DefaultWebComponents (see page 5518) property.

Pascal

```
procedure SetDefaultWebComponents(Value: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList Value);
```

Description

SetDefaultWebComponents assigns a list of Web items as the value of the DefaultWebComponents (see page 5518) property. This list specifies the controls that appear in the control group if the application does not explicitly assign any controls to the WebFieldControls (see page 5519) property.

See Also

DefaultWebComponents (see page 5518)

GetDefaultWebComponents (see page 5515)

WebFieldControls (see page 5519)

52.1.49.5 TWebControlGroup Properties

The properties of the TWebControlGroup class are listed here.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Public Properties

	Name	Description
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

52.1.49.5.1 TWebControlGroup.DefaultWebComponents Property

Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Most TWebControlGroup (see page 5510) descendants have a default set of values for the WebFieldControls (see page

5519) property.

Note: For example, on Windows, TDataNavigator (see page 5307) has a default set of buttons and TFieldGroup (see page 5320) defaults to a control for each field in the associated data packet.

TWebControlGroup (see page 5510) descendants that supply default sets of Web items store these default sets as the value of the protected DefaultWebComponents property. Then, if the value of WebFieldControls (see page 5519) is not set, the TWebControlGroup (see page 5510) descendant uses DefaultWebComponents instead.

See Also

WebFieldControls (see page 5519)

52.1.49.5.2 TWebControlGroup.WebFieldControls Property

Lists the components that generate HTML controls for the Web control group.

Pascal

```
property WebFieldControls: TWebComponentList;
```

C++

```
__property TWebComponentList WebFieldControls;
```

Description

WebFieldControls lists the components that generate the related HTML controls that the Web control group assembles into a single unit. Each of these components supports the IWebContent interface, and some can contain their own list of Web items that help them generate their content.

The type of items that can be added to the WebFieldControls list are restricted by the CanAddClass (see page 5513) method. TWebControlGroup (see page 5510) does not restrict the items in the list at all, but descendant classes add restrictions to limit the list to related types of controls.

Note: For example, on Windows, TDataNavigator (see page 5307) limits the list to button generators that are descendants of TDataSetButton (see page 5311) or TXMLButton (see page 5595).

At design time, add components to the WebFieldControls property by bringing up the Web page editor for the containing page producer. Select the Web control group in the editor, and use the editor to add child controls.

See Also

TInetXPageProducer (see page 5730)

TAdapterPageProducer

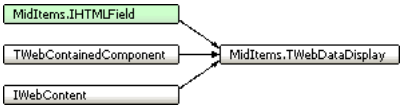
IWebContent

CanAddClass (see page 5513)

52.1.50 TWebDataDisplay Class

TWebDataDisplay is the base class for Web items that create HTML display controls.

Class Hierarchy



Pascal

```
TWebDataDisplay = class(TWebContainedComponent, IWebContent, IHTMLField);
```

C++

```
class TWebDataDisplay : public TWebContainedComponent, public IWebContent, public IHTMLField;
```

File

MidItems

Description

Use TWebDataDisplay as a base class for components that generate HTML controls which display information in a WebSnap or InternetExpress application. TWebDataDisplay descendants generate the individual controls that are organized by TAdapterGrid or TAdapterFieldGroup.

Note: On Windows, additional descendants, TDataGrid (see page 5302), TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415), can be used to generate the individual controls.

TWebDataDisplay introduces the properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebDataDisplay implements the IWebContent interface.

Do not create instances of TWebDataDisplay. TWebDataDisplay relies on an abstract (pure virtual) method for generating the display control. Instead, use a descendant of TWebDataDisplay that generates an HTML control of the desired type.

See Also

IWebContent

TAdapterGrid

TAdapterFieldGroup

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

52.1.50.1 TWebDataDisplay Members

The following tables list the members exposed by TWebDataDisplay.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.

	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

52.1.50.2 TWebDataDisplay.Create Constructor

Creates and initializes an instance of TWebDataDisplay (see page 5519).

Pascal

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

C++

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

Description

Call Create to instantiate TWebDataDisplay (see page 5519) at runtime. TWebDataDisplay (see page 5519) descendants added at design time are instantiated automatically.

52.1.50.3 TWebDataDisplay.Destroy Destructor

Frees an instance of TWebDataDisplay (see page 5519).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebDataDisplay();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component is not nil, and only then calls Destroy.

See Also

System::TObject::Free

52.1.50.4 TWebDataDisplay Methods

The methods of the TWebDataDisplay class are listed here.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

Public Methods

	Name	Description
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

52.1.50.4.1 TWebDataDisplay.Content Method

Returns the HTML that defines the display control.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Content implements the IWebContent method of the same name. This method allows the TWebDataDisplay (see page 5519) component's parent to extract the HTML defining the display control.

Content calls the virtual ImplContent (see page 5526) method to provide the HTML content that it returns. Each TWebDataDisplay (see page 5519) descendant overrides ImplContent (see page 5526) to generate the definition of its particular control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML control it generates. It supports the ILayoutWebContent interface.

Content returns a string of HTML defining the display control.

See Also

ImplContent (see page 5526)

IWebContent

ILayoutWebContent

52.1.50.4.2 TWebDataDisplay.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options) = 0;
```

Description

This is ControlContent, a member of class TWebDataDisplay.

52.1.50.4.3 TWebDataDisplay.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebDataDisplay.

52.1.50.4.4 TWebDataDisplay.FormatCaption Method

Pascal

```
function FormatCaption: string; virtual;
```

C++

```
virtual __fastcall UnicodeString FormatCaption();
```

Description

This is FormatCaption, a member of class TWebDataDisplay.

52.1.50.4.5 TWebDataDisplay.GetCaption Method

Returns the value of the Caption (see page 5529) property.

Pascal

```
function GetCaption: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption (see page 5529) property.

See Also

Caption (see page 5529)

52.1.50.4.6 TWebDataDisplay.GetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function GetHTMLControlName: string;
```

C++

```
__fastcall UnicodeString GetHTMLControlName();
```

Description

GetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

GetHTMLControlName returns the value supplied by the ImplGetHTMLControlName (see page 5527) method. Unlike GetHTMLControlName, ImplGetHTMLControlName (see page 5527) is a virtual method that descendants can override to supply a name in class-specific ways.

See Also

ImplGetHTMLControlName (see page 5527)

52.1.50.4.7 TWebDataDisplay.GetHTMLForm Method

Used internally to retrieve information about the form in which the display control is defined.

Pascal

```
function GetHTMLForm: IHTMLForm;
```

C++

```
__fastcall IHTMLForm GetHTMLForm();
```

Description

TWebDataDisplay (see page 5519) and its descendants use GetHTMLForm to obtain information about the form in which the generated display control appears. This information is used to generate code for calling form methods in response to user events.

GetHTMLForm walks up the hierarchy of Web items (using the GetParentComponent method) until it finds a component that can supply an interface for the HTML form.

See Also

GetParentComponent

52.1.50.4.8 TWebDataDisplay.GetLayoutAttributes Method

Returns the value of the LayoutAttributes (see page 5531) property.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 5523) method) to take into account layout properties of the child component.

GetLayoutAttributes returns the value of the LayoutAttributes (see page 5531) property.

See Also

Content (see page 5523)

LayoutAttributes (see page 5531)

52.1.50.4.9 TWebDataDisplay.GetXmlDisplayName Method

Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.

Pascal

```
function GetXmlDisplayName: string;
```

C++

```
__fastcall UnicodeString GetXmlDisplayName();
```

Description

GetXmlDisplayName is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 5519) for descendant classes that are only accessible on Windows.

On Windows, GetXmlDisplayName returns the name of the javascript object that groups the set of display controls which includes this TWebDataDisplay (see page 5519) descendant's generated control. The javascript object is generated by a TDataGrid (see page 5302) or TFieldGroup (see page 5320) component, and provides the display component's link to its

XML data packet. Typically, the TDataGrid (see page 5302) or TFieldGroup (see page 5320) component is the parent of the TWebDataDisplay (see page 5519) component, but it may be higher in the parent hierarchy if the application uses layout groups.

See Also

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

xmlbroker (see page 5631)

52.1.50.4.10 TWebDataDisplay.GetXmlRowSetName Method

Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.

Pascal

```
function GetXmlRowSetName: string;
```

C++

```
__fastcall UnicodeString GetXmlRowSetName();
```

Description

GetXmlRowSetName is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 5519) for descendant classes that are only accessible on Windows.

On Windows, GetXmlRowSetName is used when generating HTML that accesses data from the XML data packet. It returns the name of the rowset that represents the data in the Web data display.

As implemented in TWebDataDisplay (see page 5519), GetXmlRowSetName walks up the parent hierarchy until it finds a Web item associated with a rowset, and then returns the name of that rowset.

See Also

GetXmlDisplayName (see page 5525)

52.1.50.4.11 TWebDataDisplay.ImplContent Method

Provides the underlying implementation for the Content (see page 5523) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content (see page 5523) method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent (see page 5523) method. ControlContent (see page 5523) is an abstract (pure virtual) method in TWebDataDisplay (see page 5519). TWebDataDisplay (see page 5519) descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay (see page 5519)

should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content (see page 5523)

Caption (see page 5529)

ILayoutWebContent

52.1.50.4.12 TWebDataDisplay.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay (see page 5519) descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name

52.1.50.4.13 TWebDataDisplay.IsColumn Method

Indicates whether the component generates the controls that appear in the column of a grid.

Pascal

```
class function IsColumn: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a TWebDataDisplay (see page 5519) descendant generates the controls that appear in a single column of an adapter grid (TAdapterGrid) or data grid (TDataGrid (see page 5302)). As implemented in TWebDataDisplay (see page 5519), IsColumn returns false. Descendant classes that generate controls for the columns of a grid override this method to return true.

See Also

IsQueryField (see page 5528)

TDataGrid (see page 5302)

TAdapterGrid

52.1.50.4.14 TWebDataDisplay.IsQueryField Method

Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Pascal

```
class function IsQueryField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

IsQueryField is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 5519) for descendant classes that are only used on Windows.

On Windows, Call IsQueryField to determine whether a TWebDataDisplay (see page 5519) descendant generates the controls that represent a named value in a query form (TQueryForm (see page 5421)). As implemented in TWebDataDisplay (see page 5519), IsQueryField returns false. Descendant classes that generate controls for a TQueryFieldGroup (see page 5415) component override this method to return true.

See Also

IsColumn (see page 5527)

TQueryFieldGroup (see page 5415)

TQueryForm (see page 5421)

52.1.50.4.15 TWebDataDisplay.LabelContent Method

Pascal

```
function LabelContent: string; virtual;
```

C++

```
virtual __fastcall UnicodeString LabelContent();
```

Description

This is LabelContent, a member of class TWebDataDisplay.

52.1.50.4.16 TWebDataDisplay.SetCaption Method

Sets the value of the Caption (see page 5529) property.

Pascal

```
procedure SetCaption(Value: string); virtual;
```

C++

```
virtual __fastcall SetCaption(UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption (see page 5529) property.

See Also

Caption (see page 5529)

52.1.50.4.17 TWebDataDisplay.SetCaptionAttributes Method

Sets the value of the CaptionAttributes (see page 5530) property.

Pascal

```
procedure SetCaptionAttributes(const Value: TCaptionAttributes);
```

C++

```
__fastcall SetCaptionAttributes(const TCaptionAttributes Value);
```

Description

SetCaptionAttributes is the protected write implementation for the CaptionAttributes (see page 5530) property.

See Also

CaptionAttributes (see page 5530)

52.1.50.5 TWebDataDisplay Properties

The properties of the TWebDataDisplay class are listed here.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

52.1.50.5.1 TWebDataDisplay.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.50.5.2 TWebDataDisplay.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption (see page 5529) property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.50.5.3 TWebDataDisplay.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.50.5.4 TWebDataDisplay.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay (see page 5519) descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes (see page 5530) property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.50.5.5 TWebDataDisplay.LayoutAttributes Property

Encodes attributes that describe how the generated control should look.

Pascal

```
property LayoutAttributes: TLayoutAttributes;
```

C++

```
__property TLayoutAttributes LayoutAttributes;
```

Description

LayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 5523) method) to take into account layout properties of the child component.

LayoutAttributes encodes the information in the CaptionPosition (see page 5530) and CaptionAttributes (see page 5530) properties. Some descendant classes add information from the Style (see page 5531) and StyleRule (see page 5532) properties as well.

See Also

GetLayoutAttributes (see page 5525)

CaptionPosition (see page 5530)

Content (see page 5523)

Style (see page 5531)

CaptionAttributes (see page 5530)

stylerule (see page 5532)

52.1.50.5.6 TWebDataDisplay.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes (see page 5530) property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.50.5.7 TWebDataDisplay.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TWebDataDisplay.

52.1.50.5.8 TWebDataDisplay.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

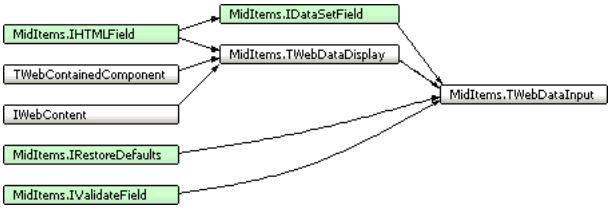
Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

52.1.51 TWebDataInput Class

TWebDataInput is the base class for Web items that create HTML display controls for entering information.

Class Hierarchy



Pascal

```
TWebDataInput = class(TWebDataDisplay, IRestoreDefaults, IDataSetField, IValidateField);
```

C++

```
class TWebDataInput : public TWebDataDisplay, public IRestoreDefaults, public IDataSetField,
public IValidateField;
```

File

MidItems

Description

Use TWebDataInput as a base class for components that generate HTML controls which display information in an InternetExpress application and into which users can enter values. TWebDataInput descendants generate controls that are organized by TDataGrid (see page 5302), TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415).

TWebDataInput defines or inherits properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Specifying a field or parameter whose value the control displays.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebDataInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebDataInput. TWebDataInput relies on an abstract (pure virtual) method for generating the display control. Instead, use a descendant of TWebDataInput that generates an HTML control of the desired type.

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

52.1.51.1 TWebDataInput Members

The following tables list the members exposed by TWebDataInput.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
A	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
V	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5521)	Creates and initializes an instance of TWebDataDisplay (see page 5519).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.

	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

52.1.51.2 TWebDataInput Methods

The methods of the TWebDataInput class are listed here.

Protected Methods

	Name	Description
V	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
V	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

52.1.51.2.1 TWebDataInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebDataInput.

52.1.51.2.2 TWebDataInput.FindAssociatedField Method

Pascal

```
function FindAssociatedField(DataSet: IInetXWebDataSet): TField; virtual;
```

C++

```
virtual __fastcall TField FindAssociatedField(IInetXWebDataSet DataSet);
```

Description

This is FindAssociatedField, a member of class TWebDataInput.

52.1.51.2.3 TWebDataInput.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 5529)

52.1.51.2.4 TWebDataInput.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class TWebDataInput.

52.1.51.2.5 TWebDataInput.GetParamName Method

Pascal

```
function GetParamName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetParamName();
```

Description

This is GetParamName, a member of class TWebDataInput.

52.1.51.2.6 TWebDataInput.GetRowSetFieldAttributes Method

Pascal

```
function GetRowSetFieldAttributes(const FieldVarName: string): string;
```

C++

```
__fastcall UnicodeString GetRowSetFieldAttributes(const UnicodeString FieldVarName);
```

Description

This is GetRowSetFieldAttributes, a member of class TWebDataInput.

52.1.51.2.7 TWebDataInput.ImplGetFieldName Method

Pascal

```
function ImplGetFieldName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetFieldName();
```

Description

This is ImplGetFieldName, a member of class TWebDataInput.

52.1.51.2.8 TWebDataInput.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name

52.1.51.2.9 TWebDataInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetRowSetFieldAttributes(const UnicodeString
FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.

52.1.51.2.10 TWebDataInput.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TWebDataInput.

52.1.51.2.11 TWebDataInput.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TWebDataInput.

52.1.51.2.12 TWebDataInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); virtual;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebDataInput.

52.1.51.2.13 TWebDataInput.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(Value: string); override;
```

C++

```
virtual __fastcall SetCaption(UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption (see page 5529)

52.1.51.2.14 TWebDataInput.SetFieldName Method

Pascal

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

C++

```
__fastcall SetFieldName(UnicodeString Value);
```

Description

This is SetFieldName, a member of class TWebDataInput.

52.1.51.2.15 TWebDataInput.SetParamName Method

Pascal

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

C++

```
__fastcall SetParamName(UnicodeString Value);
```

Description

This is SetParamName, a member of class TWebDataInput.

52.1.51.2.16 TWebDataInput.ValidateField Method

Pascal

```
function ValidateField(DataSet: IInetXWebDataSet; AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateField(IInetXWebDataSet DataSet, IAddScriptElements AddIntf);
```

Description

This is ValidateField, a member of class TWebDataInput.

52.1.51.3 TWebDataInput Properties

The properties of the TWebDataInput class are listed here.

Protected Properties

	Name	Description
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Properties

	Name	Description
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.

52.1.51.3.1 TWebDataInput.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

ParamName (see page 5540)

Setting Web Item Properties

52.1.51.3.2 TWebDataInput.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField (see page 5528) method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 5539)

IsQueryField (see page 5528)

Params

xmlbroker (see page 5631)

Setting Web Item Properties

52.1.52 TWebDisplay Class

Class Hierarchy

Pascal

```
TWebDisplay = class(TObject);
```

C++

```
class TWebDisplay : public TObject;
```

File

MidItems

Description

This is class MidItems.TWebDisplay.

52.1.52.1 TWebDisplay Members

The following tables list the members exposed by TWebDisplay.

Protected Methods

	Name	Description
	Changed (see page 5542)	This is Changed, a member of class TWebDisplay.
V	ChangedDisplayComponent (see page 5542)	This is ChangedDisplayComponent, a member of class TWebDisplay.
V	GetDisplayComponent (see page 5542)	This is GetDisplayComponent, a member of class TWebDisplay.
	SetDisplayComponent (see page 5542)	This is SetDisplayComponent, a member of class TWebDisplay.

Public Events

	Name	Description
	OnChange (see page 5543)	This is OnChange, a member of class TWebDisplay.

Public Methods

	Name	Description
	Create (see page 5541)	This is Create, a member of class TWebDisplay.
V	Notification (see page 5542)	This is Notification, a member of class TWebDisplay.

Public Properties

	Name	Description
	DisplayComponent (see page 5543)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 5543)	This is Parent, a member of class TWebDisplay.

52.1.52.2 TWebDisplay.Create Constructor

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TWebDisplay(TComponent * AParent);
```

Description

This is Create, a member of class TWebDisplay.

52.1.52.3 TWebDisplay Methods

The methods of the TWebDisplay class are listed here.

Protected Methods

	Name	Description
	Changed (see page 5542)	This is Changed, a member of class TWebDisplay.
V	ChangedDisplayComponent (see page 5542)	This is ChangedDisplayComponent, a member of class TWebDisplay.
V	GetDisplayComponent (see page 5542)	This is GetDisplayComponent, a member of class TWebDisplay.
	SetDisplayComponent (see page 5542)	This is SetDisplayComponent, a member of class TWebDisplay.

Public Methods

	Name	Description
V	Notification (see page 5542)	This is Notification, a member of class TWebDisplay.

52.1.52.3.1 TWebDisplay.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TWebDisplay.

52.1.52.3.2 TWebDisplay.ChangedDisplayComponent Method

Pascal

```
procedure ChangedDisplayComponent; virtual;
```

C++

```
virtual __fastcall ChangedDisplayComponent();
```

Description

This is ChangedDisplayComponent, a member of class TWebDisplay.

52.1.52.3.3 TWebDisplay.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TWebDisplay.

52.1.52.3.4 TWebDisplay.Notification Method

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); virtual;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This is Notification, a member of class TWebDisplay.

52.1.52.3.5 TWebDisplay.SetDisplayComponent Method

Pascal

```
procedure SetDisplayComponent(const Value: TComponent);
```

C++

```
__fastcall SetDisplayComponent(const TComponent * Value);
```

Description

This is SetDisplayComponent, a member of class TWebDisplay.

52.1.52.4 TWebDisplay Properties

The properties of the TWebDisplay class are listed here.

Public Properties

	Name	Description
	DisplayComponent (see page 5543)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 5543)	This is Parent, a member of class TWebDisplay.

52.1.52.4.1 TWebDisplay.DisplayComponent Property

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

This is DisplayComponent, a member of class TWebDisplay.

52.1.52.4.2 TWebDisplay.Parent Property

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

This is Parent, a member of class TWebDisplay.

52.1.52.5 TWebDisplay Events

The events of the TWebDisplay class are listed here.

Public Events

	Name	Description
	OnChange (see page 5543)	This is OnChange, a member of class TWebDisplay.

52.1.52.5.1 TWebDisplay.OnChange Event

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

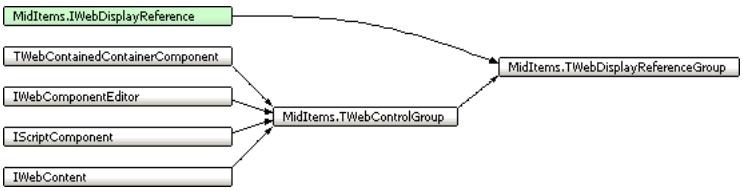
Description

This is OnChange, a member of class TWebDisplay.

52.1.53 TWebDisplayReferenceGroup Class

TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.

Class Hierarchy



Pascal

```
TWebDisplayReferenceGroup = class(TWebControlGroup, IWebDisplayReference);
```

C++

```
class TWebDisplayReferenceGroup : public TWebControlGroup, public IWebDisplayReference;
```

File

MidItems

Description

Use TWebDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of button controls. Each TWebDisplayReferenceGroup object has an associated Web item that generates controls that display data in the final HTML document. The controls arranged by TWebDisplayReferenceGroup manipulate the information the display controls show.

In order to let other Web items use the HTML that the TWebDisplayReferenceGroup object generates, TWebDisplayReferenceGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TWebDisplayReferenceGroup. TWebDisplayReferenceGroup manages the Web items that generate buttons and the association to a display generator, but does not produce any content. Instead, use a descendant of TWebDisplayReferenceGroup such as TDataNavigator (see page 5307) (in InternetExpress applications) or TAdapterCommandGroup (in WebSnap applications).

See Also

TDataNavigator (see page 5307)

IWebContent

TAdapterCommandGroup

52.1.53.1 TWebDisplayReferenceGroup Members

The following tables list the members exposed by TWebDisplayReferenceGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetWebDisplayComponent (see page 5546)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 5516)	Provides the underlying implementation for the AddElements (see page 5513) method.
V	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
V	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
V	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
V	Notification (see page 5546)	Forwards notification messages to all owned components.
V	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
V	SetWebDisplayComponent (see page 5547)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
R	WebDisplay (see page 5547)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 5547)	

Public Methods

	Name	Description
V	Create (see page 5545)	Creates an instance of TWebDisplayReferenceGroup (see page 5544).
V	Destroy (see page 5546)	Frees an instance of TWebDisplayReferenceGroup (see page 5544).

Public Properties

	Name	Description
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.

52.1.53.2 TWebDisplayReferenceGroup.Create Constructor

Creates an instance of TWebDisplayReferenceGroup (see page 5544).

Pascal

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

C++

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

Description

Call TWebDisplayReferenceGroup (see page 5544) to instantiate TWebDisplayReferenceGroup (see page 5544) at runtime. TWebDisplayReferenceGroup (see page 5544) components added at design time are instantiated automatically.

52.1.53.3 TWebDisplayReferenceGroup.Destroy Destructor

Frees an instance of TWebDisplayReferenceGroup (see page 5544).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebDisplayReferenceGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object is not nil before calling Destroy.

See Also

System::TObject::Free

52.1.53.4 TWebDisplayReferenceGroup Methods

The methods of the TWebDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	GetWebDisplayComponent (see page 5546)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	Notification (see page 5546)	Forwards notification messages to all owned components.
V	SetWebDisplayComponent (see page 5547)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

52.1.53.4.1 TWebDisplayReferenceGroup.GetWebDisplayComponent Method

Pascal

```
function GetWebDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetWebDisplayComponent();
```

Description

This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

52.1.53.4.2 TWebDisplayReferenceGroup.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification

to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.53.4.3 TWebDisplayReferenceGroup.SetWebDisplayComponent Method

Pascal

```
procedure SetWebDisplayComponent(const Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetWebDisplayComponent(const TComponent * Value);
```

Description

This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

52.1.53.5 TWebDisplayReferenceGroup Properties

The properties of the TWebDisplayReferenceGroup class are listed here.

Protected Properties

	Name	Description
	WebDisplay (see page 5547)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 5547)	

52.1.53.5.1 TWebDisplayReferenceGroup.WebDisplay Property

Pascal

```
property WebDisplay: TWebDisplay;
```

C++

```
__property TWebDisplay WebDisplay;
```

Description

This is WebDisplay, a member of class TWebDisplayReferenceGroup.

52.1.53.5.2 TWebDisplayReferenceGroup.WebDisplayComponent Property

Pascal

```
property WebDisplayComponent: TComponent;
```

C++

```
__property TComponent * WebDisplayComponent;
```

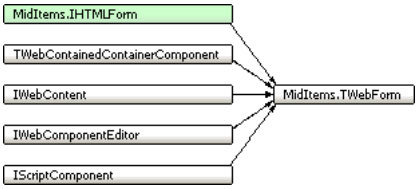
See Also

GetWebDisplayComponent (see page 5672)

52.1.54 TWebForm Class

TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.

Class Hierarchy



Pascal

```
TWebForm = class(TWebContainedContainerComponent, IWebContent, IWebComponentEditor,
IScriptComponent, IHTMLForm);
```

C++

```
class TWebForm : public TWebContainedContainerComponent, public IWebContent, public
IWebComponentEditor, public IScriptComponent, public IHTMLForm;
```

File

MidItems

Description

Use TWebForm as a base class for components that generate the HTML statements defining a form that is contained in an HTML document. The list of Web items that generate content for an InternetExpress page producer can only contain descendants of TWebForm.

In order to allow the InternetExpress page producer to use the HTML generated by the TWebForm descendant, TWebForm implements the IWebContent interface. In order to allow other Web items to generate javascript that refers to form elements, TWebForm implements the IHTMLForm (see page 5655) interface.

Do not create instances of TWebForm. TWebForm relies on an abstract or, in C++ terminology, pure virtual method to generate the HTML form. Instead, create descendants of TWebForm such a TDataForm (see page 5298) or TQueryForm (see page 5421).

See Also

TQueryForm (see page 5421)

TDataForm (see page 5298)

TBasePageItemsProducer (see page 6390)

TInetXPageProducer (see page 5730)

IWebContent

IHTMLForm (see page 5655)

52.1.54.1 TWebForm Members

The following tables list the members exposed by TWebForm.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5554)	Provides the underlying implementation for the AddElements (see page 5550) method.
A	ImplCanAddClass (see page 5554)	Provides the underlying implementation for the CanAddClass (see page 5551) method.
A	ImplContent (see page 5555)	Provides the underlying implementation for the Content (see page 5551) method.
V	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

Protected Properties

	Name	Description
R	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Methods

	Name	Description
V	Create (see page 5549)	Creates and initializes an instance of TWebForm (see page 5548).
V	Destroy (see page 5550)	Frees an instance of TWebForm (see page 5548).

Public Properties

	Name	Description
	Custom (see page 5556)	Represents a tag option and its value in the generated FORM tag.
R	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.
	Style (see page 5556)	Adds style attributes to the generated form.
	StyleRule (see page 5557)	Applies a style in the page producer's style sheet to the button.
R	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

52.1.54.2 TWebForm.Create Constructor

Creates and initializes an instance of TWebForm (see page 5548).

Pascal

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

C++

```
virtual __fastcall TWebForm(TComponent * AComponent);
```

Description

Call Create to instantiate a TWebForm (see page 5548) descendant at runtime. TWebForm (see page 5548) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.54.3 TWebForm.Destroy Destructor

Frees an instance of TWebForm (see page 5548).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebForm();
```

Description

Destroy removes the form generator from its parent's list of components that generate HTML before freeing the TWebForm (see page 5548) instance.

52.1.54.4 TWebForm Methods

The methods of the TWebForm class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 5550)	Used internally to generate any javascript declarations needed by the form.
	CanAddClass (see page 5551)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.
	Content (see page 5551)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 5552)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 5552)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5553)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 5553)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 5553)	Returns the value of the WebComponents (see page 5558) property.
	GetVisibleFields (see page 5554)	Returns the value of the VisibleFields (see page 5557) property.
V	ImplAddElements (see page 5554)	Provides the underlying implementation for the AddElements (see page 5550) method.
A	ImplCanAddClass (see page 5554)	Provides the underlying implementation for the CanAddClass (see page 5551) method.
A	ImplContent (see page 5555)	Provides the underlying implementation for the Content (see page 5551) method.
V	SetDefaultWebComponents (see page 5555)	Prevents an application from trying to assign a default set of Web items on the form.

52.1.54.4.1 TWebForm.AddElements Method

Used internally to generate any javascript declarations needed by the form.

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

The InternetExpress page producer calls AddElements when it generates the block of javascript declarations in its HTML document. This method allows the generated form to use javascript classes and methods.

AddElements calls on the virtual ImplAddElements (see page 5554) method, which uses the provided interface to return javascript declarations. Each TWebForm (see page 5548) descendant overrides ImplAddElements (see page 5554) to add any declarations it needs.

See Also

ImplAddElements (see page 5554)

TInetXPageProducer (see page 5730)

52.1.54.4.2 TWebForm.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebComponents (see page 5558) property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the TWebForm (see page 5548) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 5554) method to determine whether a class of Web item is valid.

AParent is the TWebForm (see page 5548) descendant to whose WebComponents (see page 5558) property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web form can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 5554)

WebComponents (see page 5558)

52.1.54.4.3 TWebForm.Content Method

Returns the HTML that defines the HTML form.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Content implements the IWebContent method of the same name. This method allows the page producer to extract the HTML defining the form.

Content calls the virtual ImplContent (see page 5555) method to provide the HTML form it returns. Each TWebForm (see page 5548) descendant overrides ImplContent (see page 5555) to generate its particular HTML form. The generated HTML

can make use of any javascript declarations supplied by the AddElements (see page 5550) method.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML form it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

Content returns a string of HTML defining the form.

See Also

ImplContent (see page 5555)

IWebContent

AddElements (see page 5550)

52.1.54.4.4 TWebForm.GetDefaultWebComponents Method

Returns nil (Delphi) or NULL (C++).

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents is implemented by all Web items that use sub Web items when generating HTML. (TWebForm (see page 5548) uses the subitems in the WebComponents (see page 5558) property). This method returns a list of Web items that are used by default when an application does not explicitly assign child Web items.

In TWebForm (see page 5548), GetDefaultWebComponents returns nil (Delphi) or NULL (C++) because Web forms do not any set of items that appear by default.

See Also

WebComponents (see page 5558)

GetDefaultWebComponents

52.1.54.4.5 TWebForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormName();
```

Description

TWebForm (see page 5548) uses GetHTMLFormName internally to specify the NAME option of the generated form and to declare a javascript variable that references the form.

GetHTMLFormName returns the value of the Name property.

See Also

GetHTMLFormVarName (see page 5553)

Name

52.1.54.4.6 TWebForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

TWebForm (see page 5548) uses GetHTMLFormVarName internally to declare a javascript variable that references the form. GetHTMLFormVarName returns the value of the Name property.

See Also

GetHTMLFormName (see page 5552)

Name

52.1.54.4.7 TWebForm.GetLayoutAttributes Method

Returns the object that specifies how the generated form should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 5551) method) to take into account layout properties of the child component.

As implemented by TWebForm (see page 5548), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout at the form level does not make use of layout attributes.

See Also

Content (see page 5551)

52.1.54.4.8 TWebForm.GetSubComponents Method

Returns the value of the WebComponents (see page 5558) property.

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

GetSubComponents returns the TWebForm (see page 5548)'s list of Web items that it uses to generate HTML content. This method is part of the mechanism that fits TWebForm (see page 5548) into a hierarchy of Web items, each of which generates

part of a larger HTML document.

See Also

WebComponents (see page 5558)

52.1.54.4.9 TWebForm.GetVisibleFields Method

Returns the value of the VisibleFields (see page 5557) property.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

GetVisibleFields is the protected read implementation of the VisibleFields (see page 5557) property.

See Also

VisibleFields (see page 5557)

52.1.54.4.10 TWebForm.ImplAddElements Method

Provides the underlying implementation for the AddElements (see page 5550) method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements (see page 5550) method calls ImplAddElements to add any javascript declarations by calling the AddIntf interface. These declarations allow the generated display control to call javascript methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebForm (see page 5548), ImplAddElements does nothing. TWebForm (see page 5548) descendants can override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5550)

52.1.54.4.11 TWebForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5551) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass) = 0;
```

Description

The CanAddClass (see page 5551) method calls ImpleCanAddClass to determine whether the TWebForm (see page 5548)

descendant can add a component of type AClass to its WebComponents (see page 5558) property list. AParent is the TWebForm (see page 5548) instance to which the component will be added.

In TWebForm (see page 5548), ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm (see page 5548) descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 5551)

WebComponents (see page 5558)

52.1.54.4.12 TWebForm.ImplContent Method

Provides the underlying implementation for the Content (see page 5551) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout Layout) = 0;
```

Description

The Content (see page 5551) method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm (see page 5548). TWebForm (see page 5548) descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm (see page 5548) should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

Content (see page 5551)

AddElements (see page 5550)

52.1.54.4.13 TWebForm.SetDefaultWebComponents Method

Prevents an application from trying to assign a default set of Web items on the form.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents is implemented by all Web items that use sub Web items when generating HTML. (TWebForm (see page 5548) uses the subitems in the WebComponents (see page 5558) property). This method assigns a list of Web items that are used by default when an application does not explicitly assign child Web items.

In TWebForm (see page 5548), SetDefaultWebComponents does nothing with the list supplied as the AList parameter. This is because web forms do not have default items.

See Also

WebComponents (see page 5558)

SetDefaultWebComponents

52.1.54.5 TWebForm Properties

The properties of the TWebForm class are listed here.

Protected Properties

	Name	Description
	VisibleFields (see page 5557)	Lists the Web items whose generated content appears in the HTML form.

Public Properties

	Name	Description
	Custom (see page 5556)	Represents a tag option and its value in the generated FORM tag.
	Style (see page 5556)	Adds style attributes to the generated form.
	StyleRule (see page 5557)	Applies a style in the page producer's style sheet to the button.
	WebComponents (see page 5558)	Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

52.1.54.5.1 TWebForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm (see page 5548) descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 5556)

52.1.54.5.2 TWebForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm (see page 5548) descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 5556)

StyleRule (see page 5557)

52.1.54.5.3 TWebForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 5556) property.

See Also

Style (see page 5556)

Styles (see page 5716)

StylesFile (see page 5717)

52.1.54.5.4 TWebForm.VisibleFields Property

Lists the Web items whose generated content appears in the HTML form.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

VisibleFields is used internally when assembling the HTML form from the HTML generated by the form's Web items. For TWebForm (see page 5548), the value of VisibleFields is always the same as WebComponents (see page 5558).

Override GetVisibleFields (see page 5554) in a descendant class to provide a default set of that appear when the WebComponents (see page 5558) property is not set.

See Also

WebComponents (see page 5558)

52.1.54.5.5 TWebForm.WebComponents Property

Maintains a list of components that generate HTML content for the TWebForm (see page 5548) descendant.

Pascal

```
property WebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList WebComponents;
```

Description

WebComponents lists the components that generate content for the Web form. Each of these components supports the IWebContent interface, and some can contain their own list of Web items that help them generate their content.

The type of items that can be added to the WebComponents list are restricted by the protected CanAddClass (see page 5551) method. Descendant classes override this method to indicate what types of items can be added to the list. For example, TQueryForm (see page 5421) limits the list to components that display or manipulate named values (TQueryButtons (see page 5412), TQueryFields) or layout groups that manage the layout of those items (TLayoutGroup (see page 5389)).

At design time, add components to the WebComponents property by bringing up the Web page editor for the containing TInetXPageProducer (see page 5730) component. Select the TWebForm (see page 5548) descendant in the editor, and use the editor to add child controls.

See Also

IWebContent

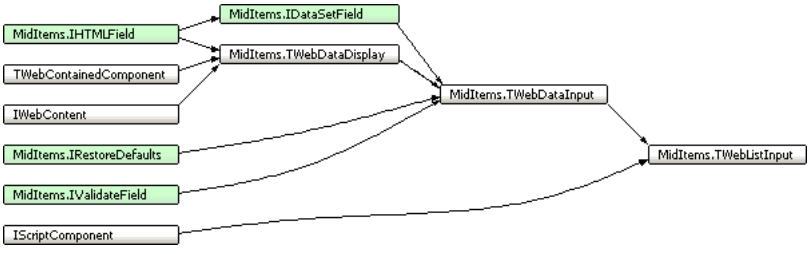
TInetXPageProducer (see page 5730)

CanAddClass (see page 5551)

52.1.55 TWebListInput Class

TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.

Class Hierarchy



Pascal

```
TWebListInput = class(TWebDataInput, IScriptComponent);
```

C++

```
class TWebListInput : public TWebDataInput, public IScriptComponent;
```

File

MidItems

Description

Use TWebListInput as a base class for components that generate HTML controls for displaying a fixed set of possible values from which a user can choose. TWebListInput descendants generate controls that are organized by TDataGrid (see page 5302), TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415).

TWebListInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebListInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebListInput. Instead, use a descendant of TWebListInput such as TFieldRadioGroup (see page 5326), TQueryRadioGroup (see page 5425), TFieldSelectOptions (see page 5335), TQuerySelectOptions (see page 5437), or TSelectOptionsColumn (see page 5472).

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TFieldRadioGroup (see page 5326)

TQUeryRadioGroup

TFieldSelectOptions (see page 5335)

TQuerySelectOptions (see page 5437)

TSelectOptionsColumn (see page 5472)

52.1.55.1 TWebListInput Members

The following tables list the members exposed by TWebListInput.

Protected Methods

	Name	Description
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
V	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
V	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.

	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
	Create (see page 5561)	Creates and initializes an instance of TWebListInput (see page 5558).
	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.

	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5564)	Specifies a dataset that supplies the possible values listed in the generated control.
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Items (see page 5565)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5565)	Indicates which field in DataSet (see page 5564) supplies the possible selections for the generated control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5566)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5566)	Indicates which field in DataSet (see page 5564) supplies the possible values for the generated control.

52.1.55.2 TWebListInput.Create Constructor

Creates and initializes an instance of TWebListInput (see page 5558).

Pascal

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

C++

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

Description

Call Create to instantiate TWebListInput (see page 5558) at runtime. TWebListInput (see page 5558) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.55.3 TWebListInput.Destroy Destructor

Frees an instance of TWebListInput (see page 5558).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebListInput();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component is not nil, and only

then calls Destroy.

See Also

System::TObject::Free

52.1.55.4 TWebListInput Methods

The methods of the TWebListInput class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
V	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
A	FormatInputs (see page 5562)	This is FormatInputs, a member of class TWebListInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
V	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
V	Notification (see page 5563)	Forwards notification messages to all owned components.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.

52.1.55.4.1 TWebListInput.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TWebListInput.

52.1.55.4.2 TWebListInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebListInput.

52.1.55.4.3 TWebListInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options) = 0;
```

Description

This is FormatInputs, a member of class TWebListInput.

52.1.55.4.4 TWebListInput.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TWebListInput.

52.1.55.4.5 TWebListInput.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TWebListInput.

52.1.55.4.6 TWebListInput.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.55.4.7 TWebListInput.SetDataSet Method

Pascal

```
procedure SetDataSet(const Value: TDataSet);
```

C++

```
__fastcall SetDataSet(const TDataSet Value);
```

Description

This is SetDataSet, a member of class TWebListInput.

52.1.55.4.8 TWebListInput.SetItems Method

Pascal

```
procedure SetItems(const Value: TStrings);
```

C++

```
__fastcall SetItems(const TStrings Value);
```

Description

This is SetItems, a member of class TWebListInput.

52.1.55.4.9 TWebListInput.SetValues Method

Pascal

```
procedure SetValues(const Value: TStrings);
```

C++

```
__fastcall SetValues(const TStrings Value);
```

Description

This is SetValues, a member of class TWebListInput.

52.1.55.5 TWebListInput Properties

The properties of the TWebListInput class are listed here.

Public Properties

	Name	Description
	DataSet (see page 5564)	Specifies a dataset that supplies the possible values listed in the generated control.
	Items (see page 5565)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5565)	Indicates which field in DataSet (see page 5564) supplies the possible selections for the generated control.
	Values (see page 5566)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5566)	Indicates which field in DataSet (see page 5564) supplies the possible values for the generated control.

52.1.55.5.1 TWebListInput.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField (see page 5565) property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField (see page 5566) property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items (see page 5565) and Values (see page 5566) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 5565)

ValuesField (see page 5566)

Items (see page 5565)

Values (see page 5566)

52.1.55.5.2 TWebListInput.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values (see page 5566) property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values (see page 5566) properties, use the DataSet (see page 5564), ItemsField (see page 5565), and ValuesField (see page 5566) properties.

See Also

ItemsField (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.55.5.3 TWebListInput.ItemsField Property

Indicates which field in DataSet (see page 5564) supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property UnicodeString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet (see page 5564) property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField (see page 5566) property.

Note: If the list of possible values is not available in a dataset, use the Items (see page 5565) and Values (see page 5566) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.55.5.4 TWebListInput.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items (see page 5565) property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items (see page 5565) and Values properties, use the DataSet (see page 5564), ItemsField (see page 5565), and ValuesField (see page 5566) properties.

See Also

Items (see page 5565)

ItemsField (see page 5565)

DataSet (see page 5564)

ValuesField (see page 5566)

52.1.55.5.5 TWebListInput.ValuesField Property

Indicates which field in DataSet (see page 5564) supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property UnicodeString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet (see page 5564) property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField (see page 5565) field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items (see page 5565) and Values (see page 5566) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 5565)

Values (see page 5566)

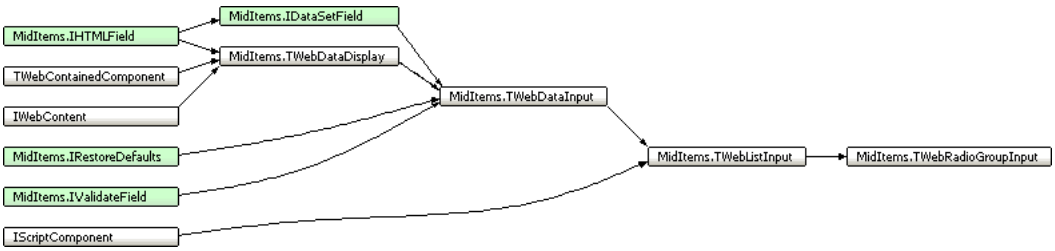
DataSet (see page 5564)

ItemsField (see page 5565)

52.1.56 TWebRadioGroupInput Class

TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.

Class Hierarchy



Pascal

```
TWebRadioGroupInput = class(TWebListInput);
```

C++

```
class TWebRadioGroupInput : public TWebListInput;
```

File

MidItems

Description

Use TWebRadioGroupInput as a base class for components that generate HTML radio groups that represent field or parameter values. TWebRadioGroupInput descendants generate controls that are organized by TFieldGroup (see page 5320) or TQueryFieldGroup (see page 5415).

TWebRadioGroupInput defines or inherits properties and methods for

Specifying a field or parameter whose value the radio group displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the radio group.

Fitting the radio group into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebRadioGroupInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebRadioGroupInput. Instead, use a descendant of TWebRadioGroupInput such as TFieldRadioGroup (see page 5326) or TQueryRadioGroup (see page 5425).

See Also

IWebContent

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TFieldRadioGroup (see page 5326)

TQUeryRadioGroup

52.1.56.1 TWebRadioGroupInput Members

The following tables list the members exposed by TWebRadioGroupInput.

Protected Methods

	Name	Description
	AddAttributes (see page 5570)	This is AddAttributes, a member of class TWebRadioGroupInput.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5571)	This is EventContent, a member of class TWebRadioGroupInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5571)	This is FormatInputs, a member of class TWebRadioGroupInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetCheckIndex (see page 5571)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.

	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 5571)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 5571)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
	Create (see page 5570)	Creates and initializes an instance of TWebRadioGroupInput (see page 5567).
	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 5564)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 5572)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5572)	Specifies the width of the radio group, in characters.
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.

	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Items (see page 5565)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5565)	Indicates which field in DataSet (see page 5564) supplies the possible selections for the generated control.
	ReadOnly (see page 5573)	Indicates whether end users can use the generated control to set values.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5566)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5566)	Indicates which field in DataSet (see page 5564) supplies the possible values for the generated control.

52.1.56.2 TWebRadioGroupInput.Create Constructor

Creates and initializes an instance of TWebRadioGroupInput (see page 5567).

Pascal

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

C++

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

Description

Call Create to instantiate TWebRadioGroupInput (see page 5567) at runtime. TWebRadioGroupInput (see page 5567) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.56.3 TWebRadioGroupInput Methods

The methods of the TWebRadioGroupInput class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 5570)	This is AddAttributes, a member of class TWebRadioGroupInput.
V	EventContent (see page 5571)	This is EventContent, a member of class TWebRadioGroupInput.
V	FormatInputs (see page 5571)	This is FormatInputs, a member of class TWebRadioGroupInput.
V	GetCheckIndex (see page 5571)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
V	ImplGetRowSetFieldAttributes (see page 5571)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
V	RestoreFieldDefaults (see page 5571)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.

52.1.56.3.1 TWebRadioGroupInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TWebRadioGroupInput.

52.1.56.3.2 TWebRadioGroupInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebRadioGroupInput.

52.1.56.3.3 TWebRadioGroupInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options);
```

Description

This is FormatInputs, a member of class TWebRadioGroupInput.

52.1.56.3.4 TWebRadioGroupInput.GetCheckIndex Method

Pascal

```
function GetCheckIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; virtual;
```

C++

```
virtual __fastcall int GetCheckIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetCheckIndex, a member of class TWebRadioGroupInput.

52.1.56.3.5 TWebRadioGroupInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetRowSetFieldAttributes(const UnicodeString
FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.

52.1.56.3.6 TWebRadioGroupInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.

52.1.56.4 TWebRadioGroupInput Properties

The properties of the TWebRadioGroupInput class are listed here.

Public Properties

	Name	Description
	DisplayColumns (see page 5572)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 5572)	Specifies the width of the radio group, in characters.
	ReadOnly (see page 5573)	Indicates whether end users can use the generated control to set values.

52.1.56.4.1 TWebRadioGroupInput.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 5565)

Items (see page 5565)

DisplayWidth (see page 5572)

52.1.56.4.2 TWebRadioGroupInput.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 5572)

Custom (see page 5530)

52.1.56.4.3 TWebRadioGroupInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

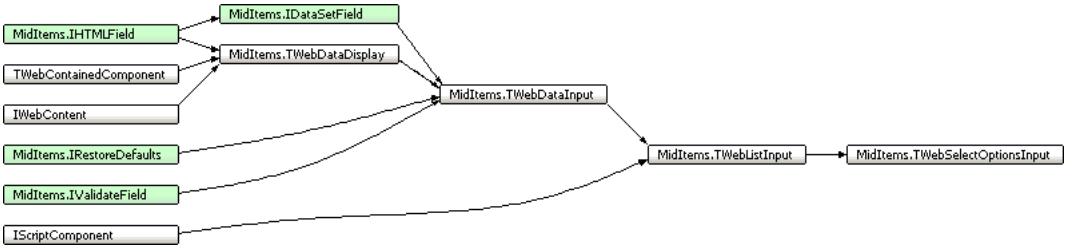
DisplayWidth (see page 5572)

Custom (see page 5530)

52.1.57 TWebSelectOptionsInput Class

TWebSelectOptionsInput is the base class for Web items that create HTML list controls.

Class Hierarchy



Pascal

```
TWebSelectOptionsInput = class(TWebListInput);
```

C++

```
class TWebSelectOptionsInput : public TWebListInput;
```

File

MidItems

Description

Use TWebSelectOptionsInput as a base class for components that generate HTML list controls that represent field or parameter values. TWebSelectOptionsInput descendants generate controls that are organized by TDataGrid (see page 5302), TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415).

TWebSelectOptionsInput defines or inherits properties and methods for

Specifying a field or parameter whose value the list box displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the list box.

Fitting the list box into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebSelectOptionsInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebSelectOptionsInput. Instead, use a descendant of TWebSelectOptionsInput such as TFieldSelectOptions (see page 5335), TQuerySelectOptions (see page 5437), or TSelectOptionsColumn (see page 5472).

See Also

IWebContent

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TFieldSelectOptions (see page 5335)

TQUerySelectOptions

TDataGrid (see page 5302)

TSelectOptionsColumn (see page 5472)

52.1.57.1 TWebSelectOptionsInput Members

The following tables list the members exposed by TWebSelectOptionsInput.

Protected Methods

	Name	Description
	AddAttributes (see page 5576)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 5562)	This is AddElements, a member of class TWebListInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5523)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 5562)	This is ControlContent, a member of class TWebListInput.
	EventContent (see page 5577)	This is EventContent, a member of class TWebSelectOptionsInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 5577)	This is FormatInputs, a member of class TWebSelectOptionsInput.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSelectIndex (see page 5577)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 5563)	This is GetSubComponents, a member of class TWebListInput.

	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 5563)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5537)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	Notification (see page 5563)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5538)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetDataSet (see page 5564)	This is SetDataSet, a member of class TWebListInput.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetItems (see page 5564)	This is SetItems, a member of class TWebListInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	SetValues (see page 5564)	This is SetValues, a member of class TWebListInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5576)	Creates and initializes an instance of TWebSelectOptionsInput (see page 5573).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
V	Destroy (see page 5561)	Frees an instance of TWebListInput (see page 5558).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.

	DataSet (see page 5564)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 5577)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Items (see page 5565)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 5565)	Indicates which field in DataSet (see page 5564) supplies the possible selections for the generated control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 5566)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 5566)	Indicates which field in DataSet (see page 5564) supplies the possible values for the generated control.

52.1.57.2 TWebSelectOptionsInput.Create Constructor

Creates and initializes an instance of TWebSelectOptionsInput (see page 5573).

Pascal

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

C++

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

Description

Call Create to instantiate TWebSelectOptionsInput (see page 5573) at runtime. TWebSelectOptionsInput (see page 5573) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.57.3 TWebSelectOptionsInput Methods

The methods of the TWebSelectOptionsInput class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 5576)	This is AddAttributes, a member of class TWebSelectOptionsInput.
V	EventContent (see page 5577)	This is EventContent, a member of class TWebSelectOptionsInput.
V	FormatInputs (see page 5577)	This is FormatInputs, a member of class TWebSelectOptionsInput.
V	GetSelectIndex (see page 5577)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.

52.1.57.3.1 TWebSelectOptionsInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TWebSelectOptionsInput.

52.1.57.3.2 TWebSelectOptionsInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebSelectOptionsInput.

52.1.57.3.3 TWebSelectOptionsInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options);
```

Description

This is FormatInputs, a member of class TWebSelectOptionsInput.

52.1.57.3.4 TWebSelectOptionsInput.GetSelectIndex Method

Pascal

```
function GetSelectIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; virtual;
```

C++

```
virtual __fastcall int GetSelectIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetSelectIndex, a member of class TWebSelectOptionsInput.

52.1.57.4 TWebSelectOptionsInput Properties

The properties of the TWebSelectOptionsInput class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 5577)	Specifies the number of rows that appear in the generated list box.

52.1.57.4.1 TWebSelectOptionsInput.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

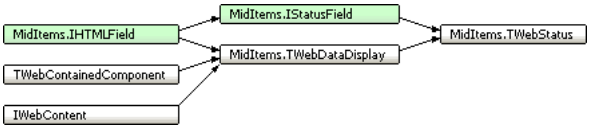
ItemsField (see page 5565)

Items (see page 5565)

52.1.58 TWebStatus Class

TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.

Class Hierarchy



Pascal

```
TWebStatus = class(TWebDataDisplay, IStatusField);
```

C++

```
class TWebStatus : public TWebDataDisplay, public IStatusField;
```

File

MidItems

Description

Use TWebStatus as a base class for components that generate HTML controls which indicate whether records have been inserted or modified by the user. TWebStatus descendants generate controls that are added to TDataGrid (see page 5302) or TFieldGroup (see page 5320) to provide feedback about modifications to the underlying XML data packet.

TWebStatus defines or inherits properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebStatus inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebStatus. Instead, use a descendant of TWebStatus such as TFieldStatus (see page 5344) or TStatusColumn (see page 5481).

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TFieldStatus (see page 5344)

TStatusColumn (see page 5481)

52.1.58.1 TWebStatus Members

The following tables list the members exposed by TWebStatus.

Protected Methods

	Name	Description
	Content (see page 5523)	Returns the HTML that defines the display control.
V	ControlContent (see page 5580)	This is ControlContent, a member of class TWebStatus.
V	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
V	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 5581)	Returns the value of the Caption property.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetHTMLControlName (see page 5581)	Returns the name that other components can use to refer to the generated display control.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.

Public Methods

	Name	Description
V	Create (see page 5580)	Creates and initializes an instance of TWebStatus (see page 5578).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	Identifier (see page 5581)	Returns the name under which record status information is stored in the XML data packet.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Public Properties

	Name	Description
	Caption (see page 5582)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5582)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5583)	Specifies where the label appears relative to the generated display control.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5583)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

52.1.58.2 TWebStatus.Create Constructor

Creates and initializes an instance of TWebStatus (see page 5578).

Pascal

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

C++

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

Description

Call Create to instantiate TWebStatus (see page 5578) at runtime. TWebStatus (see page 5578) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.58.3 TWebStatus Methods

The methods of the TWebStatus class are listed here.

Protected Methods

	Name	Description
V	ControlContent (see page 5580)	This is ControlContent, a member of class TWebStatus.
V	GetCaption (see page 5581)	Returns the value of the Caption property.
V	ImplGetHTMLControlName (see page 5581)	Returns the name that other components can use to refer to the generated display control.

Public Methods

	Name	Description
	Identifier (see page 5581)	Returns the name under which record status information is stored in the XML data packet.

52.1.58.3.1 TWebStatus.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebStatus.

52.1.58.3.2 TWebStatus.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 5529)

52.1.58.3.3 TWebStatus.Identifier Method

Returns the name under which record status information is stored in the XML data packet.

Pascal

```
class function Identifier: string;
```

C++

```
__fastcall UnicodeString Identifier();
```

Description

Call Identifier to determine the name of the pseudo-field in the XML data packet that stores the status information displayed by the generated control.

52.1.58.3.4 TWebStatus.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name

52.1.58.4 TWebStatus Properties

The properties of the TWebStatus class are listed here.

Public Properties

	Name	Description
	Caption (see page 5582)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5582)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 5583)	Specifies where the label appears relative to the generated display control.
	DisplayWidth (see page 5583)	Specifies the width, in characters, of the generated HTML control.

52.1.58.4.1 TWebStatus.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 5530)

CaptionPosition (see page 5530)

52.1.58.4.2 TWebStatus.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 5529)

CaptionPosition (see page 5530)

HeadingAttributes (see page 5269)

52.1.58.4.3 TWebStatus.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 5530)

Caption (see page 5529)

52.1.58.4.4 TWebStatus.DisplayWidth Property

Specifies the width, in characters, of the generated HTML control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

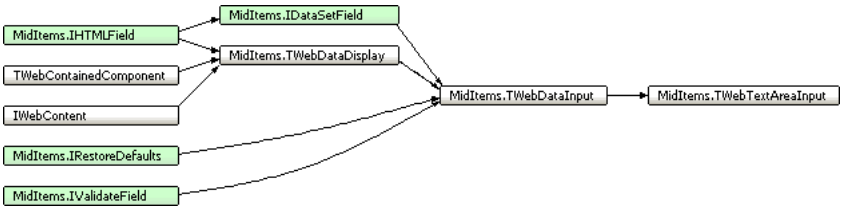
Description

Use DisplayWidth to control the width of the HTML control that displays record status information. By default, DisplayWidth is 1 because the TWebStatus (see page 5578) control values are one character (M for modified or I for inserted).

52.1.59 TWebTextAreaInput Class

TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.

Class Hierarchy



Pascal

```
TWebTextAreaInput = class(TWebDataInput);
```

C++

```
class TWebTextAreaInput : public TWebDataInput;
```

File

MidItems

Description

Use TWebTextAreaInput as a base class for components that generate HTML controls for displaying a field or parameter value in a multi-line text control. TWebTextAreaInput descendants generate controls that are organized by TDataGrid (see page 5302), TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415).

TWebTextAreaInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebTextAreaInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebTextAreaInput. Instead, use a descendant of TWebTextAreaInput such as TFieldTextArea (see page 5355), TQueryTextArea (see page 5458), or TTextAreaColumn (see page 5488).

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TFieldTextArea (see page 5355)

TQueryTextArea (see page 5458)

TTextAreaColumn (see page 5488)

52.1.59.1 TWebTextAreaInput Members

The following tables list the members exposed by TWebTextAreaInput.

Protected Methods

	Name	Description
	AddAttributes (see page 5586)	This is AddAttributes, a member of class TWebTextAreaInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5586)	This is ControlContent, a member of class TWebTextAreaInput.
	EventContent (see page 5587)	This is EventContent, a member of class TWebTextAreaInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.

	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
V	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5587)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5587)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5586)	Creates and initializes an instance of TWebTextAreaInput (see page 5583).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5530)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 5588)	Specifies the height of the generated control.
	DisplayWidth (see page 5588)	Specifies the horizontal size of the generated control.

	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	ReadOnly (see page 5588)	Indicates whether end users can use the generated control to set values.
	Style (see page 5531)	Adds style attributes to the generated display control.
	StyleRule (see page 5532)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 5588)	Specifies how the generated control treats lines that are longer than the display width.

52.1.59.2 TWebTextAreaInput.Create Constructor

Creates and initializes an instance of TWebTextAreaInput (see page 5583).

Pascal

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

C++

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

Description

Call Create to instantiate TWebTextAreaInput (see page 5583) at runtime. TWebTextAreaInput (see page 5583) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.59.3 TWebTextAreaInput Methods

The methods of the TWebTextAreaInput class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 5586)	This is AddAttributes, a member of class TWebTextAreaInput.
V	ControlContent (see page 5586)	This is ControlContent, a member of class TWebTextAreaInput.
V	EventContent (see page 5587)	This is EventContent, a member of class TWebTextAreaInput.
V	ImplGetRowSetFieldAttributes (see page 5587)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	RestoreFieldDefaults (see page 5587)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.

52.1.59.3.1 TWebTextAreaInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TWebTextAreaInput.

52.1.59.3.2 TWebTextAreaInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebTextAreaInput.

52.1.59.3.3 TWebTextAreaInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebTextAreaInput.

52.1.59.3.4 TWebTextAreaInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetRowSetFieldAttributes(const UnicodeString
FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.

52.1.59.3.5 TWebTextAreaInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebTextAreaInput.

52.1.59.4 TWebTextAreaInput Properties

The properties of the TWebTextAreaInput class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 5588)	Specifies the height of the generated control.
	DisplayWidth (see page 5588)	Specifies the horizontal size of the generated control.
	ReadOnly (see page 5588)	Indicates whether end users can use the generated control to set values.
	Wrap (see page 5588)	Specifies how the generated control treats lines that are longer than the display width.

52.1.59.4.1 TWebTextAreaInput.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 5588)

52.1.59.4.2 TWebTextAreaInput.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 5588)

52.1.59.4.3 TWebTextAreaInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

52.1.59.4.4 TWebTextAreaInput.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth (see page 5588). TWebTextAreaInput (see page 5583) uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

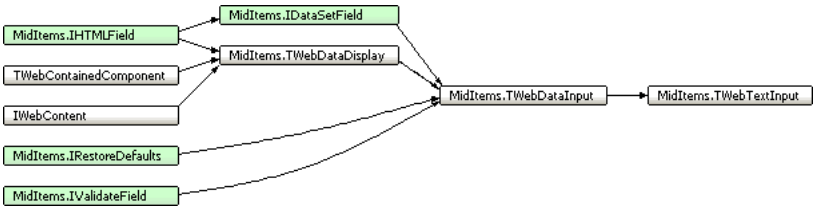
See Also

DisplayWidth (see page 5588)

52.1.60 TWebTextInput Class

TWebTextInput is the base class for Web items that create HTML single-line text controls.

Class Hierarchy



Pascal

```
TWebTextInput = class(TWebDataInput);
```

C++

```
class TWebTextInput : public TWebDataInput;
```

File

MidItems

Description

Use TWebTextInput as a base class for components that generate HTML controls for displaying a field or parameter value as a single-line text string. TWebTextInput descendants generate controls that are organized by TDataGrid (see page 5302),

TFieldGroup (see page 5320), or TQueryFieldGroup (see page 5415).

TWebTextInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebTextInput inherits the IWebContent interface from TWebDataDisplay (see page 5519).

Do not create instances of TWebTextInput. Instead, use a descendant of TWebTextInput such as TFieldText (see page 5349), TQueryText (see page 5449), or TTextColumn (see page 5495).

See Also

IWebContent

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

TFieldText (see page 5349)

TQueryText (see page 5449)

TTextColumn (see page 5495)

52.1.60.1 TWebTextInput Members

The following tables list the members exposed by TWebTextInput.

Protected Methods

	Name	Description
	AddAttributes (see page 5592)	This is AddAttributes, a member of class TWebTextInput.
	Content (see page 5523)	Returns the HTML that defines the display control.
	ControlContent (see page 5592)	This is ControlContent, a member of class TWebTextInput.
	EventContent (see page 5523)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 5535)	This is EventContent, a member of class TWebDataInput.
	FindAssociatedField (see page 5535)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 5524)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 5524)	Returns the value of the Caption (see page 5529) property.
	GetCaption (see page 5536)	Returns the value of the Caption property.
	GetFieldName (see page 5536)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 5524)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 5524)	Used internally to retrieve information about the form in which the display control is defined.
	GetLayoutAttributes (see page 5525)	Returns the value of the LayoutAttributes (see page 5531) property.
	GetParamName (see page 5536)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 5536)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetXmlDisplayName (see page 5525)	Returns the name of the javascript object that uses TWebDataDisplay (see page 5519) to generate one of its display controls. Not used on Linux.

	GetXmlRowSetName (see page 5526)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 5526)	Provides the underlying implementation for the Content (see page 5523) method.
V	ImplGetFieldName (see page 5537)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 5527)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 5537)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 5593)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 5537)	This is ImplRestoreDefaults, a member of class TWebDataInput.
V	LabelContent (see page 5528)	This is LabelContent, a member of class TWebDataDisplay.
	RestoreDefaults (see page 5538)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 5593)	This is RestoreFieldDefaults, a member of class TWebTextInput.
V	SetCaption (see page 5528)	Sets the value of the Caption (see page 5529) property.
V	SetCaption (see page 5538)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 5529)	Sets the value of the CaptionAttributes (see page 5530) property.
	SetFieldName (see page 5538)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 5539)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 5539)	This is ValidateField, a member of class TWebDataInput.

Protected Properties

	Name	Description
	CaptionPosition (see page 5530)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 5531)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 5540)	Specifies the name of the parameter whose value the generated control represents.

Public Methods

	Name	Description
V	Create (see page 5592)	Creates and initializes an instance of TWebTextInput (see page 5589).
V	Destroy (see page 5522)	Frees an instance of TWebDataDisplay (see page 5519).
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
V	IsColumn (see page 5527)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 5528)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 5529)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 5530)	Customizes the appearance of the generated control's label.
	Custom (see page 5593)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5594)	Specifies the horizontal size of the generated control.
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 5539)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	ReadOnly (see page 5594)	Indicates whether end users can use the generated control to set values.
	Style (see page 5594)	Adds style attributes to the generated display control.
	StyleRule (see page 5595)	This is StyleRule, a member of class TWebTextInput.
	TabIndex (see page 5532)	Determines the position of the generated control in the HTML form's tab order.

52.1.60.2 TWebTextInput.Create Constructor

Creates and initializes an instance of TWebTextInput (see page 5589).

Pascal

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

C++

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

Description

Call Create to instantiate a TWebTextInput (see page 5589) descendant at runtime. TWebTextInput (see page 5589) descendants added at design time using the Web Page editor are instantiated automatically.

52.1.60.3 TWebTextInput Methods

The methods of the TWebTextInput class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 5592)	This is AddAttributes, a member of class TWebTextInput.
	ControlContent (see page 5592)	This is ControlContent, a member of class TWebTextInput.
	ImplGetRowSetFieldAttributes (see page 5593)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
	RestoreFieldDefaults (see page 5593)	This is RestoreFieldDefaults, a member of class TWebTextInput.

52.1.60.3.1 TWebTextInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TWebTextInput.

52.1.60.3.2 TWebTextInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebTextInput.

52.1.60.3.3 TWebTextInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetRowSetFieldAttributes(const UnicodeString
FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.

52.1.60.3.4 TWebTextInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebTextInput.

52.1.60.4 TWebTextInput Properties

The properties of the TWebTextInput class are listed here.

Public Properties

	Name	Description
	Custom (see page 5593)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 5594)	Specifies the horizontal size of the generated control.
	ReadOnly (see page 5594)	Indicates whether end users can use the generated control to set values.
	Style (see page 5594)	Adds style attributes to the generated display control.
	StyleRule (see page 5595)	This is StyleRule, a member of class TWebTextInput.

52.1.60.4.1 TWebTextInput.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator

key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 5531)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

52.1.60.4.2 TWebTextInput.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

52.1.60.4.3 TWebTextInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

52.1.60.4.4 TWebTextInput.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 5530)

stylerule (see page 5532)

CaptionAttributes (see page 5530)

Setting Web Item Properties

52.1.60.4.5 TWebTextInput.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

This is StyleRule, a member of class TWebTextInput.

52.1.61 TXMLButton Class

TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.

Class Hierarchy

Pascal

```
TXMLButton = class(TWebButton);
```

C++

```
class TXMLButton : public TWebButton;
```

File

MidItems

Description

Use TXMLButton as a base class for components that generate the HTML statements defining a specialized button control for manipulating an XML data packet. TXMLButton defines properties and methods for

Generating the HTML code to define the button control.

Associating the button with an XML broker that fetches the XML data packet and responds to update messages.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TXMLButton inherits support for the IWebContent interface from TWebButton (see page 5504).

Do not create instances of TXMLButton. TXMLButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TXMLButton, such as TApplyUpdatesButton (see page 5250).

See Also

TApplyUpdatesButton (see page 5250)

IWebContent

52.1.61.1 TXMLButton Members

The following tables list the members exposed by TXMLButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5597)	Returns the value of the Caption property.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetXMLBroker (see page 5598)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 5598)	This is GetXMLUseParent, a member of class TXMLButton.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5598)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetXMLBroker (see page 5599)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 5599)	This is SetXMLUseParent, a member of class TXMLButton.

Protected Properties

	Name	Description
	DefaultCaption (see page 5599)	This is DefaultCaption, a member of class TXMLButton.
R	XMLData (see page 5600)	This is XMLData, a member of class TXMLButton.

Public Methods

	Name	Description
V	Create (see page 5596)	Creates and initializes an instance of TXMLButton (see page 5595).
V	Destroy (see page 5597)	Frees an instance of TXMLButton (see page 5595).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLBroker (see page 5599)	Associates the XML button with an XML broker.
	XMLUseParent (see page 5600)	Specifies whether the XML button is associated with its parent's XML broker.

52.1.61.2 TXMLButton.Create Constructor

Creates and initializes an instance of TXMLButton (see page 5595).

Pascal

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

C++

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

Description

Call Create to instantiate an XML button at runtime. Web buttons added at design time using the Web Page editor are instantiated automatically.

52.1.61.3 TXMLButton.Destroy Destructor

Frees an instance of TXMLButton (see page 5595).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLButton();
```

Description

Destroy removes the XML button from its parent's list of components that generate HTML before freeing the button instance.

52.1.61.4 TXMLButton Methods

The methods of the TXMLButton class are listed here.

Protected Methods

	Name	Description
	GetCaption (see page 5597)	Returns the value of the Caption property.
	GetXMLBroker (see page 5598)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 5598)	This is GetXMLUseParent, a member of class TXMLButton.
	Notification (see page 5598)	Forwards notification messages to all owned components.
	SetXMLBroker (see page 5599)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 5599)	This is SetXMLUseParent, a member of class TXMLButton.

52.1.61.4.1 TXMLButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 5508)

SetCaption (see page 5508)

52.1.61.4.2 TXMLButton.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLButton.

52.1.61.4.3 TXMLButton.GetXMLUseParent Method

Pascal

```
function GetXMLUseParent: Boolean;
```

C++

```
__fastcall Boolean GetXMLUseParent();
```

Description

This is GetXMLUseParent, a member of class TXMLButton.

52.1.61.4.4 TXMLButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.61.4.5 TXMLButton.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker);
```

C++

```
__fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLButton.

52.1.61.4.6 TXMLButton.SetXMLUseParent Method

Pascal

```
procedure SetXMLUseParent(const Value: Boolean);
```

C++

```
__fastcall SetXMLUseParent(const Boolean Value);
```

Description

This is SetXMLUseParent, a member of class TXMLButton.

52.1.61.5 TXMLButton Properties

The properties of the TXMLButton class are listed here.

Protected Properties

	Name	Description
	DefaultCaption (see page 5599)	This is DefaultCaption, a member of class TXMLButton.
R	XMLData (see page 5600)	This is XMLData, a member of class TXMLButton.

Public Properties

	Name	Description
	XMLBroker (see page 5599)	Associates the XML button with an XML broker.
	XMLUseParent (see page 5600)	Specifies whether the XML button is associated with its parent's XML broker.

52.1.61.5.1 TXMLButton.DefaultCaption Property

Pascal

```
property DefaultCaption: string;
```

C++

```
__property UnicodeString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TXMLButton.

52.1.61.5.2 TXMLButton.XMLBroker Property

Associates the XML button with an XML broker.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Set XMLBroker to specify the XML broker whose data packets the XML button manipulates.

When XML buttons are used by TDataNavigator (see page 5307), the data navigator already establishes a connection (indirectly) to an XML broker. If the XML button uses this same XML broker, you do not need to set the XMLBroker property. Instead, setting the XMLUseParent (see page 5600) property to true (the default) automatically associates the XML button with the same broker as the one associated with the navigator.

See Also

XMLUseParent (see page 5600)

52.1.61.5.3 TXMLButton.XMLData Property

Pascal

```
property XMLData: TXMLDataParent;
```

C++

```
__property TXMLDataParent XMLData;
```

Description

This is XMLData, a member of class TXMLButton.

52.1.61.5.4 TXMLButton.XMLUseParent Property

Specifies whether the XML button is associated with its parent's XML broker.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the XML button determines the XML broker whose data packets it manipulates. When XMLUseParent is false, the XMLBroker (see page 5599) property must be set to indicate the XML broker. When XMLUseParent is true, the XML broker associated with the XML button's parent determines the associated XML broker.

The XML button's parent is the Web item (typically TDataNavigator (see page 5307)) that uses the XML button to assemble its own HTML content. The parent's associated broker may be defined through an indirect association. For example, the data navigator acts on another component, such as a field group, which has an associated broker.

If the parent does not have an associated XML broker, setting XMLUseParent to true sets the XMLBroker (see page 5599) property to nil (Delphi) or NULL (C++).

See Also

XMLBroker (see page 5599)

52.1.62 TXMLData Class

TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.

Class Hierarchy

Pascal

```
TXMLData = class(TObject);
```

C++

```
class TXMLData : public TObject;
```

File

MidItems

Description

TXMLData represents the link to an XML broker for a Web item that is bound to a dataset in the XML broker's data packet. It is the base class for TXMLDataSet (see page 5608), which allows the Web item to bind to a dataset field nested in the XML broker's top-level dataset.

TXMLData does not support the ability to bind to a nested dataset field. Rather, it introduces the underlying support needed to keep track of the XML broker for components that supply this ability.

See Also

TXMLBroker

IDataSetComponent (see page 5644)

TXMLDataSet (see page 5608)

52.1.62.1 TXMLData Members

The following tables list the members exposed by TXMLData.

Protected Methods

	Name	Description
	Changed (see page 5602)	This is Changed, a member of class TXMLData.
	ChangedXMLBroker (see page 5603)	This is ChangedXMLBroker, a member of class TXMLData.
	GetXMLBroker (see page 5603)	This is GetXMLBroker, a member of class TXMLData.
	SetXMLBroker (see page 5603)	This is SetXMLBroker, a member of class TXMLData.

Public Events

	Name	Description
	OnChange (see page 5605)	Occurs when the XMLBroker (see page 5604) property changes.

Public Methods

	Name	Description
	Create (see page 5602)	Creates a new TXMLData (see page 5601) object.
	Notification (see page 5603)	Responds to notifications when objects are created or destroyed.

Public Properties

	Name	Description
	Parent (see page 5604)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 5604)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

52.1.62.2 TXMLData.Create Constructor

Creates a new TXMLData (see page 5601) object.

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLData(TComponent * AParent);
```

Description

Most applications do not need to call the TXMLData (see page 5601) constructor. The Web item that uses the TXMLData (see page 5601) object to represent its link to a dataset instantiates this object.

The AParent parameter specifies the Web item that uses the TXMLData (see page 5601) object to link to an XML broker. This becomes the value of the Parent (see page 5604) property.

See Also

Parent (see page 5604)

52.1.62.3 TXMLData Methods

The methods of the TXMLData class are listed here.

Protected Methods

	Name	Description
	Changed (see page 5602)	This is Changed, a member of class TXMLData.
	ChangedXMLBroker (see page 5603)	This is ChangedXMLBroker, a member of class TXMLData.
	GetXMLBroker (see page 5603)	This is GetXMLBroker, a member of class TXMLData.
	SetXMLBroker (see page 5603)	This is SetXMLBroker, a member of class TXMLData.

Public Methods

	Name	Description
	Notification (see page 5603)	Responds to notifications when objects are created or destroyed.

52.1.62.3.1 TXMLData.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TXMLData.

52.1.62.3.2 TXMLData.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; virtual;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLData.

52.1.62.3.3 TXMLData.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; virtual;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLData.

52.1.62.3.4 TXMLData.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); virtual;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method. It is called automatically when objects are created or destroyed. TXMLData (see page 5601) responds to notifications by setting the XMLBroker (see page 5604) property to nil (Delphi) or NULL (C++) if it is notified that this component is about to be destroyed.

See Also

XMLBroker (see page 5604)

52.1.62.3.5 TXMLData.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker); virtual;
```

C++

```
virtual __fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLData.

52.1.62.4 TXMLData Properties

The properties of the TXMLData class are listed here.

Public Properties

	Name	Description
	Parent (see page 5604)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 5604)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

52.1.62.4.1 TXMLData.Parent Property

Indicates the Web item that uses this object to represent its link to an XML broker.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to determine the Web item that uses this object to bind to a dataset from an XML broker. The value of Parent is set by the constructor.

See Also

XMLBroker (see page 5604)

Create (see page 5602)

52.1.62.4.2 TXMLData.XMLBroker Property

Specifies the XML broker that generates the dataset to which the associated Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

The Web item specified by the Parent (see page 5604) property sets XMLBroker to bind to an XML broker.

When the value of XMLBroker changes, the TXMLData (see page 5601) object generates an OnChange (see page 5605) event.

See Also

Parent (see page 5604)

OnChange (see page 5605)

52.1.62.5 TXMLData Events

The events of the TXMLData class are listed here.

Public Events

	Name	Description
	OnChange (see page 5605)	Occurs when the XMLBroker (see page 5604) property changes.

52.1.62.5.1 TXMLData.OnChange Event

Occurs when the XMLBroker (see page 5604) property changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond immediately after the Web item specified by Parent (see page 5604) becomes bound to a different XML broker. This occurs when Parent (see page 5604) changes the XMLBroker (see page 5604) property explicitly in response to a change in its own properties, or when the XML broker object to which it is bound is destroyed.

See Also

XMLBroker (see page 5604)

Parent (see page 5604)

52.1.63 TXMLDataParent Class

Class Hierarchy

Pascal

```
TXMLDataParent = class(TXMLData);
```

C++

```
class TXMLDataParent : public TXMLData;
```

File

MidItems

Description

This is class MidItems.TXMLDataParent.

52.1.63.1 TXMLDataParent Members

The following tables list the members exposed by TXMLDataParent.

Protected Methods

	Name	Description
	Changed (see page 5602)	This is Changed, a member of class TXMLData.
	ChangedUseParent (see page 5606)	This is ChangedUseParent, a member of class TXMLDataParent.

	ChangedXMLBroker (see page 5606)	This is ChangedXMLBroker, a member of class TXMLDataParent.
	GetXMLBroker (see page 5607)	This is GetXMLBroker, a member of class TXMLDataParent.
	ParentXMLBroker (see page 5607)	This is ParentXMLBroker, a member of class TXMLDataParent.
	SetUseParent (see page 5607)	This is SetUseParent, a member of class TXMLDataParent.
	SetXMLBroker (see page 5603)	This is SetXMLBroker, a member of class TXMLData.

Public Events

	Name	Description
	OnChange (see page 5605)	Occurs when the XMLBroker (see page 5604) property changes.

Public Methods

	Name	Description
	Create (see page 5602)	Creates a new TXMLData (see page 5601) object.
	Notification (see page 5603)	Responds to notifications when objects are created or destroyed.

Public Properties

	Name	Description
	Parent (see page 5604)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 5604)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

Published Properties

	Name	Description
	UseParent (see page 5608)	This is UseParent, a member of class TXMLDataParent.

52.1.63.2 TXMLDataParent Methods

The methods of the TXMLDataParent class are listed here.

Protected Methods

	Name	Description
	ChangedUseParent (see page 5606)	This is ChangedUseParent, a member of class TXMLDataParent.
	ChangedXMLBroker (see page 5606)	This is ChangedXMLBroker, a member of class TXMLDataParent.
	GetXMLBroker (see page 5607)	This is GetXMLBroker, a member of class TXMLDataParent.
	ParentXMLBroker (see page 5607)	This is ParentXMLBroker, a member of class TXMLDataParent.
	SetUseParent (see page 5607)	This is SetUseParent, a member of class TXMLDataParent.

52.1.63.2.1 TXMLDataParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDataParent.

52.1.63.2.2 TXMLDataParent.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataParent.

52.1.63.2.3 TXMLDataParent.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; override;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLDataParent.

52.1.63.2.4 TXMLDataParent.ParentXMLBroker Method

Pascal

```
function ParentXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker ParentXMLBroker();
```

Description

This is ParentXMLBroker, a member of class TXMLDataParent.

52.1.63.2.5 TXMLDataParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDataParent.

52.1.63.3 TXMLDataParent Properties

The properties of the TXMLDataParent class are listed here.

Published Properties

	Name	Description
	UseParent (see page 5608)	This is UseParent, a member of class TXMLDataParent.

52.1.63.3.1 TXMLDataParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

Description

This is UseParent, a member of class TXMLDataParent.

52.1.64 TXMLDataSet Class

TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.

Class Hierarchy

Pascal

```
TXMLDataSet = class(TXMLData, INotifyConnectionChange);
```

C++

```
class TXMLDataSet : public TXMLData, public INotifyConnectionChange;
```

File

MidItems

Description

TXMLDataSet represents a Web item's link to an XML dataset. The Web item can be bound to an XML broker's data packet, or to a dataset field nested in an XML broker's data packet.

See Also

TXMLBroker

IDataSetComponent (see page 5644)

TXMLDisplayGroup (see page 5620)

52.1.64.1 TXMLDataSet Members

The following tables list the members exposed by TXMLDataSet.

Protected Methods

	Name	Description
	_AddRef (see page 5610)	This is _AddRef, a member of class TXMLDataSet.
	_Release (see page 5611)	This is _Release, a member of class TXMLDataSet.
	Changed (see page 5602)	This is Changed, a member of class TXMLData.
V	ChangedDataSetField (see page 5611)	This is ChangedDataSetField, a member of class TXMLDataSet.
V	ChangedXMLBroker (see page 5611)	This is ChangedXMLBroker, a member of class TXMLDataSet.

	CompareXMLData (see page 5611)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 5611)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 5612)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 5612)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 5612)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 5612)	This is GetDataSetField, a member of class TXMLDataSet.
	GetDataSetPath (see page 5613)	This is GetDataSetPath, a member of class TXMLDataSet.
V	GetXMLBroker (see page 5603)	This is GetXMLBroker, a member of class TXMLData.
V	QueryInterface (see page 5613)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 5613)	This is SetDataSetField, a member of class TXMLDataSet.
V	SetXMLBroker (see page 5613)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 5613)	This is WrapperDataChange, a member of class TXMLDataSet.

Protected Properties

	Name	Description
R	Parent (see page 5615)	Indicates the Web item that uses this object to represent its link to an XML broker.

Public Events

	Name	Description
	OnChange (see page 5605)	Occurs when the XMLBroker (see page 5604) property changes.

Public Methods

	Name	Description
	Create (see page 5609)	Creates a new TXMLDataSet (see page 5608) object.
V	Destroy (see page 5610)	TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 5608) object.
V	Notification (see page 5603)	Responds to notifications when objects are created or destroyed.

Public Properties

	Name	Description
R	DataSet (see page 5614)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 5614)	Identifies a nested dataset in the associated XML broker's data packet.
R	DataSetPath (see page 5615)	Represents the parsed value of the DataSetField (see page 5614) property.
	XMLBroker (see page 5604)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

52.1.64.2 TXMLDataSet.Create Constructor

Creates a new TXMLDataSet (see page 5608) object.

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLDataSet(TComponent * AParent);
```

Description

Most applications do not need to call the TXMLDataSet (see page 5608) constructor. The Web item that uses the TXMLDataSet (see page 5608) object to represent its link to a dataset instantiates this object.

The AParent parameter specifies the Web item that uses the TXMLDataSet (see page 5608) object to link to an XML dataset. This becomes the value of the Parent (see page 5615) property.

See Also

Parent (see page 5604)

52.1.64.3 TXMLDataSet.Destroy Destructor

TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 5608) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDataSet();
```

Description

Typically, applications do not free instances of TXMLDataSet (see page 5608). Instead, this object is freed by the Web item that uses it.

52.1.64.4 TXMLDataSet Methods

The methods of the TXMLDataSet class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5610)	This is _AddRef, a member of class TXMLDataSet.
	_Release (see page 5611)	This is _Release, a member of class TXMLDataSet.
V	ChangedDataSetField (see page 5611)	This is ChangedDataSetField, a member of class TXMLDataSet.
V	ChangedXMLBroker (see page 5611)	This is ChangedXMLBroker, a member of class TXMLDataSet.
	CompareXMLData (see page 5611)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 5611)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 5612)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 5612)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 5612)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 5612)	This is GetDataSetField, a member of class TXMLDataSet.
	GetDataSetPath (see page 5613)	This is GetDataSetPath, a member of class TXMLDataSet.
V	QueryInterface (see page 5613)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 5613)	This is SetDataSetField, a member of class TXMLDataSet.
V	SetXMLBroker (see page 5613)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 5613)	This is WrapperDataChange, a member of class TXMLDataSet.

52.1.64.4.1 TXMLDataSet._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TXMLDataSet.

52.1.64.4.2 TXMLDataSet._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TXMLDataSet.

52.1.64.4.3 TXMLDataSet.ChangedDataSetField Method

Pascal

```
procedure ChangedDataSetField; virtual;
```

C++

```
virtual __fastcall ChangedDataSetField();
```

Description

This is ChangedDataSetField, a member of class TXMLDataSet.

52.1.64.4.4 TXMLDataSet.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataSet.

52.1.64.4.5 TXMLDataSet.CompareXMLData Method

Pascal

```
function CompareXMLData(XMLDataSet: TXMLDataSet): Boolean;
```

C++

```
__fastcall Boolean CompareXMLData(TXMLDataSet XMLDataSet);
```

Description

This is CompareXMLData, a member of class TXMLDataSet.

52.1.64.4.6 TXMLDataSet.ConnectionChange Method

Pascal

```
procedure ConnectionChange(Sender: TComponent; Connecting: Boolean);
```

C++

```
__fastcall ConnectionChange(TComponent * Sender, Boolean Connecting);
```

Description

This is ConnectionChange, a member of class TXMLDataSet.

52.1.64.4.7 TXMLDataSet.CreateDataSet Method

Pascal

```
function CreateDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet CreateDataSet();
```

Description

This is CreateDataSet, a member of class TXMLDataSet.

52.1.64.4.8 TXMLDataSet.FindDataSet Method

Pascal

```
function FindDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet FindDataSet();
```

Description

This is FindDataSet, a member of class TXMLDataSet.

52.1.64.4.9 TXMLDataSet.GetDataSet Method

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TXMLDataSet.

52.1.64.4.10 TXMLDataSet.GetDataSetField Method

Pascal

```
function GetDataSetField: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDataSet.

52.1.64.4.11 TXMLDataSet.GetDataSetPath Method

Pascal

```
function GetDataSetPath: TStrings;
```

C++

```
__fastcall TStrings GetDataSetPath();
```

Description

This is GetDataSetPath, a member of class TXMLDataSet.

52.1.64.4.12 TXMLDataSet.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TXMLDataSet.

52.1.64.4.13 TXMLDataSet.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: string);
```

C++

```
__fastcall SetDataSetField(const UnicodeString Value);
```

Description

This is SetDataSetField, a member of class TXMLDataSet.

52.1.64.4.14 TXMLDataSet.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker); override;
```

C++

```
virtual __fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLDataSet.

52.1.64.4.15 TXMLDataSet.WrapperDataChange Method

Pascal

```
procedure WrapperDataChange(Sender: TObject);
```

C++

```
__fastcall WrapperDataChange(TObject * Sender);
```

Description

This is WrapperDataChange, a member of class TXMLDataSet.

52.1.64.5 TXMLDataSet Properties

The properties of the TXMLDataSet class are listed here.

Protected Properties

	Name	Description
	Parent (see page 5615)	Indicates the Web item that uses this object to represent its link to an XML broker.

Public Properties

	Name	Description
	DataSet (see page 5614)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 5614)	Identifies a nested dataset in the associated XML broker's data packet.
	DataSetPath (see page 5615)	Represents the parsed value of the DataSetField (see page 5614) property.

52.1.64.5.1 TXMLDataSet.DataSet Property

Provides access to an interface for working with the associated dataset.

Pascal

```
property DataSet: IInetXWebDataSet;
```

C++

```
__property IInetXWebDataSet DataSet;
```

Description

Use DataSet to manipulate the associated dataset. DataSet is an interface that provides information about the available fields and parameters of the dataset and lets you respond when this information changes.

An advantage of using the DataSet property is that it handles the task of identifying the correct dataset within the associated XML broker's data packet. This lets you treat nested datasets in the same way that you interact with the XML broker's main dataset.

See Also

XMLBroker

DataSetField (see page 5614)

52.1.64.5.2 TXMLDataSet.DataSetField Property

Identifies a nested dataset in the associated XML broker's data packet.

Pascal

```
property DataSetField: string;
```

C++

```
__property UnicodeString DataSetField;
```

Description

Use DataSetField, in conjunction with the XMLBroker (see page 5604) property, to identify the XML dataset that the TXMLDataSet (see page 5608) object represents. If the dataset is the dataset represented directly by the XML broker's data

packet, DataSetField is an empty string and the XMLBroker (see page 5604) property fully describes the dataset. If the dataset is a nested detail set, DataSetField indicates what detail dataset to use from the data packet associated with XMLBroker (see page 5604).

Specify the name of the dataset field as the value of DataSetField. If the dataset is the detail set of a detail set, you can use periods to qualify dataset field names. Thus, for example, if XMLBroker (see page 5604) represents a customer dataset with a nested Orders dataset field that has its own nested LineItems field, you can specify the LineItems dataset by setting DataSetField to

```
Orders.LineItems
```

See Also

XMLBroker

DataSet (see page 5614)

DataSetPath (see page 5615)

52.1.64.5.3 TXMLDataSet.DataSetPath Property

Represents the parsed value of the DataSetField (see page 5614) property.

Pascal

```
property DataSetPath: TStrings;
```

C++

```
__property TStrings DataSetPath;
```

Description

DataSetPath is a parsed version of the DataSetField (see page 5614) property, where each dataset field is a separate string in the TStrings object. The order of the strings indicates the level of nesting, where the first string names a dataset field in the XML broker's data packet, the second string names a dataset field in that nested dataset, and so on.

See Also

XMLBroker

DataSetField (see page 5614)

52.1.64.5.4 TXMLDataSet.Parent Property

Indicates the Web item that uses this object to represent its link to an XML broker.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to determine the Web item that uses this object to bind to a dataset from an XML broker. The value of Parent is set by the constructor.

See Also

XMLBroker (see page 5604)

Create (see page 5602)

52.1.65 TXMLDataSetParent Class

Class Hierarchy

Pascal

```
TXMLDataSetParent = class(TXMLDataSet);
```

C++

```
class TXMLDataSetParent : public TXMLDataSet;
```

File

MidItems

Description

This is class MidItems.TXMLDataSetParent.

52.1.65.1 TXMLDataSetParent Members

The following tables list the members exposed by TXMLDataSetParent.

Protected Methods

	Name	Description
	_AddRef (see page 5610)	This is _AddRef, a member of class TXMLDataSet.
	_Release (see page 5611)	This is _Release, a member of class TXMLDataSet.
	Changed (see page 5602)	This is Changed, a member of class TXMLData.
V	ChangedDataSetField (see page 5617)	This is ChangedDataSetField, a member of class TXMLDataSetParent.
V	ChangedUseParent (see page 5618)	This is ChangedUseParent, a member of class TXMLDataSetParent.
V	ChangedXMLBroker (see page 5618)	This is ChangedXMLBroker, a member of class TXMLDataSetParent.
	CompareXMLData (see page 5611)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 5611)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 5612)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 5612)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 5612)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 5618)	This is GetDataSetField, a member of class TXMLDataSetParent.
	GetDataSetPath (see page 5613)	This is GetDataSetPath, a member of class TXMLDataSet.
V	GetXMLBroker (see page 5618)	This is GetXMLBroker, a member of class TXMLDataSetParent.
	ParentDataSetField (see page 5618)	This is ParentDataSetField, a member of class TXMLDataSetParent.
	ParentXMLBroker (see page 5619)	This is ParentXMLBroker, a member of class TXMLDataSetParent.
	ParentXMLDataSet (see page 5619)	This is ParentXMLDataSet, a member of class TXMLDataSetParent.
V	QueryInterface (see page 5613)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 5613)	This is SetDataSetField, a member of class TXMLDataSet.
	SetUseParent (see page 5619)	This is SetUseParent, a member of class TXMLDataSetParent.
V	SetXMLBroker (see page 5603)	This is SetXMLBroker, a member of class TXMLData.
V	SetXMLBroker (see page 5613)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 5613)	This is WrapperDataChange, a member of class TXMLDataSet.

Protected Properties

	Name	Description
	Parent (see page 5615)	Indicates the Web item that uses this object to represent its link to an XML broker.
	UseParent (see page 5619)	This is UseParent, a member of class TXMLDataSetParent.

Public Events

	Name	Description
	OnChange (see page 5605)	Occurs when the XMLBroker (see page 5604) property changes.

Public Methods

	Name	Description
	Create (see page 5602)	Creates a new TXMLData (see page 5601) object.
	Create (see page 5609)	Creates a new TXMLDataSet (see page 5608) object.
	Destroy (see page 5610)	TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 5608) object.
	Notification (see page 5603)	Responds to notifications when objects are created or destroyed.

Public Properties

	Name	Description
	DataSet (see page 5614)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 5614)	Identifies a nested dataset in the associated XML broker's data packet.
	DataSetPath (see page 5615)	Represents the parsed value of the DataSetField (see page 5614) property.
	Parent (see page 5604)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 5604)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

52.1.65.2 TXMLDataSetParent Methods

The methods of the TXMLDataSetParent class are listed here.

Protected Methods

	Name	Description
	ChangedDataSetField (see page 5617)	This is ChangedDataSetField, a member of class TXMLDataSetParent.
	ChangedUseParent (see page 5618)	This is ChangedUseParent, a member of class TXMLDataSetParent.
	ChangedXMLBroker (see page 5618)	This is ChangedXMLBroker, a member of class TXMLDataSetParent.
	GetDataSetField (see page 5618)	This is GetDataSetField, a member of class TXMLDataSetParent.
	GetXMLBroker (see page 5618)	This is GetXMLBroker, a member of class TXMLDataSetParent.
	ParentDataSetField (see page 5618)	This is ParentDataSetField, a member of class TXMLDataSetParent.
	ParentXMLBroker (see page 5619)	This is ParentXMLBroker, a member of class TXMLDataSetParent.
	ParentXMLDataSet (see page 5619)	This is ParentXMLDataSet, a member of class TXMLDataSetParent.
	SetUseParent (see page 5619)	This is SetUseParent, a member of class TXMLDataSetParent.

52.1.65.2.1 TXMLDataSetParent.ChangedDataSetField Method

Pascal

```
procedure ChangedDataSetField; override;
```

C++

```
virtual __fastcall ChangedDataSetField();
```

Description

This is ChangedDataSetField, a member of class TXMLDataSetParent.

52.1.65.2.2 TXMLDataSetParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDataSetParent.

52.1.65.2.3 TXMLDataSetParent.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataSetParent.

52.1.65.2.4 TXMLDataSetParent.GetDataSetField Method

Pascal

```
function GetDataSetField: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDataSetParent.

52.1.65.2.5 TXMLDataSetParent.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; override;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLDataSetParent.

52.1.65.2.6 TXMLDataSetParent.ParentDataSetField Method

Pascal

```
function ParentDataSetField: string;
```

C++

```
__fastcall UnicodeString ParentDataSetField();
```

Description

This is ParentDataSetField, a member of class TXMLDataSetParent.

52.1.65.2.7 TXMLDataSetParent.ParentXMLBroker Method

Pascal

```
function ParentXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker ParentXMLBroker();
```

Description

This is ParentXMLBroker, a member of class TXMLDataSetParent.

52.1.65.2.8 TXMLDataSetParent.ParentXMLDataSet Method

Pascal

```
function ParentXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet ParentXMLDataSet();
```

Description

This is ParentXMLDataSet, a member of class TXMLDataSetParent.

52.1.65.2.9 TXMLDataSetParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDataSetParent.

52.1.65.3 TXMLDataSetParent Properties

The properties of the TXMLDataSetParent class are listed here.

Protected Properties

	Name	Description
	UseParent (see page 5619)	This is UseParent, a member of class TXMLDataSetParent.

52.1.65.3.1 TXMLDataSetParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

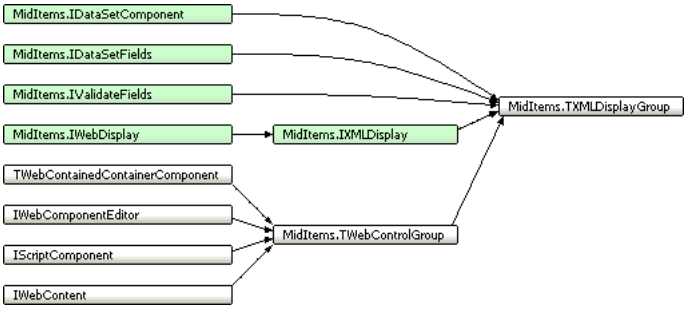
Description

This is UseParent, a member of class TXMLDataSetParent.

52.1.66 TXMLDisplayGroup Class

TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.

Class Hierarchy



Pascal

```
TXMLDisplayGroup = class(TWebControlGroup, IDataSetComponent, IDataSetFields, IXMLDisplay,
IValidateFields);
```

C++

```
class TXMLDisplayGroup : public TWebControlGroup, public IDataSetComponent, public
IDataSetFields, public IXMLDisplay, public IValidateFields;
```

File

MidItems

Description

Use TXMLDisplayGroup as a base class for Web items that generate the HTML to arrange a collection of HTML controls. Each TXMLDisplayGroup object maintains a list of Web items that display information in the final HTML document. The XML display group arranges the controls that these items generate.

If the controls generated for TXMLDisplayGroup by its Web items represent database information, you can bind the XML display group to an XML data packet or dataset field within an XML data packet. The Web items can use that binding to determine the source of the data they display.

In order to let other Web items use the HTML that the TXMLDisplayGroup object generates, TXMLDisplayGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TXMLDisplayGroup. TXMLDisplayGroup manages the Web items that generate display controls, but does not produce any content. Instead, use a descendant of TXMLDisplayGroup that generates the HTML for its set of specialized controls such as TDataGrid (see page 5302), TFieldGroup (see page 5320) or TQueryFieldGroup (see page 5415).

See Also

TDataGrid (see page 5302)

TFieldGroup (see page 5320)

TQueryFieldGroup (see page 5415)

IWebContent

52.1.66.1 TXMLDisplayGroup Members

The following tables list the members exposed by TXMLDisplayGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
	ImplAddElements (see page 5626)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.

	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	Create (see page 5622)	Creates and initializes an instance of TXMLDisplayGroup (see page 5620).
	Destroy (see page 5622)	Frees an instance of TXMLDisplayGroup (see page 5620).

Public Properties

	Name	Description
	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
	IsMultipleRecordView (see page 5671)	
	IsMultipleRecordView (see page 5674)	
	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.
	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.66.2 TXMLDisplayGroup.Create Constructor

Creates and initializes an instance of TXMLDisplayGroup (see page 5620).

Pascal

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

C++

```
virtual __fastcall TXMLDisplayGroup(TComponent * AComponent);
```

Description

Call Create to instantiate TXMLDisplayGroup (see page 5620) at runtime. TXMLDisplayGroup (see page 5620) components added at design time using the Web Page editor are instantiated automatically.

52.1.66.3 TXMLDisplayGroup.Destroy Destructor

Frees an instance of TXMLDisplayGroup (see page 5620).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDisplayGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the object is not nil before calling Destroy.

See Also

System::TObject::Free

52.1.66.4 TXMLDisplayGroup Methods

The methods of the TXMLDisplayGroup class are listed here.

Protected Methods

	Name	Description
	CreateDefaultFields (see page 5623)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 5623)	This is FindField, a member of class TXMLDisplayGroup.
	GetDataSet (see page 5624)	Returns the value of the DataSet property.
	GetDataSetField (see page 5624)	This is GetDataSetField, a member of class TXMLDisplayGroup.
	GetEnableValidateFields (see page 5624)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
	GetFieldsList (see page 5624)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 5625)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
	GetVisibleFields (see page 5625)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 5625)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 5625)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 5626)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 5626)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 5626)	This is HasStatusField, a member of class TXMLDisplayGroup.
	ImplAddElements (see page 5626)	Provides the underlying implementation for the AddElements method.
	ImplGetSubComponents (see page 5627)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 5627)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 5627)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	Notification (see page 5628)	Forwards notification messages to all owned components.
	SetDataSetField (see page 5628)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetEnableValidateFields (see page 5628)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 5629)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 5629)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 5629)	This is ValidateFields, a member of class TXMLDisplayGroup.
	XMLDataChange (see page 5629)	This is XMLDataChange, a member of class TXMLDisplayGroup.

52.1.66.4.1 TXMLDisplayGroup.CreateDefaultFields Method

Pascal

```
procedure CreateDefaultFields; virtual;
```

C++

```
virtual __fastcall CreateDefaultFields();
```

Description

This is CreateDefaultFields, a member of class TXMLDisplayGroup.

52.1.66.4.2 TXMLDisplayGroup.FindField Method

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

This is FindField, a member of class TXMLDisplayGroup.

52.1.66.4.3 TXMLDisplayGroup.GetDataSet Method

Returns the value of the DataSet property.

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

GetDataSet is the read access method for the DataSet property.

See Also

DataSet (see page 5646)

52.1.66.4.4 TXMLDisplayGroup.GetDataSetField Method

Pascal

```
function GetDataSetField: string;
```

C++

```
__fastcall UnicodeString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDisplayGroup.

52.1.66.4.5 TXMLDisplayGroup.GetEnableValidateFields Method

Pascal

```
function GetEnableValidateFields: Boolean;
```

C++

```
__fastcall Boolean GetEnableValidateFields();
```

Description

This is GetEnableValidateFields, a member of class TXMLDisplayGroup.

52.1.66.4.6 TXMLDisplayGroup.GetFieldsList Method

Pascal

```
procedure GetFieldsList(List: TStrings); virtual;
```

C++

```
virtual __fastcall GetFieldsList(TStrings List);
```

Description

This is GetFieldsList, a member of class TXMLDisplayGroup.

52.1.66.4.7 TXMLDisplayGroup.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.

52.1.66.4.8 TXMLDisplayGroup.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class TXMLDisplayGroup.

52.1.66.4.9 TXMLDisplayGroup.GetXMLBroker Method

Returns the value of the XMLBroker property.

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

GetXMLBroker is the read access method for the XMLBroker property.

See Also

XMLBroker (see page 5647)

52.1.66.4.10 TXMLDisplayGroup.GetXMLDataSet Method

Returns the value of the XMLDataSet property.

Pascal

```
function GetXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet GetXMLDataSet();
```

Description

GetXMLDataSet is the read access method for the XMLDataSet property.

See Also

XMLDataSet (see page 5647)

52.1.66.4.11 TXMLDisplayGroup.GetXMLDisplayName Method

Pascal

```
function GetXMLDisplayName: string;
```

C++

```
__fastcall UnicodeString GetXMLDisplayName();
```

Description

This is GetXMLDisplayName, a member of class TXMLDisplayGroup.

52.1.66.4.12 TXMLDisplayGroup.GetXMLRowSetName Method

Pascal

```
function GetXMLRowSetName: string;
```

C++

```
__fastcall UnicodeString GetXMLRowSetName();
```

Description

This is GetXMLRowSetName, a member of class TXMLDisplayGroup.

52.1.66.4.13 TXMLDisplayGroup.HasStatusField Method

Pascal

```
function HasStatusField: Boolean;
```

C++

```
__fastcall Boolean HasStatusField();
```

Description

This is HasStatusField, a member of class TXMLDisplayGroup.

52.1.66.4.14 TXMLDisplayGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override

ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.66.4.15 TXMLDisplayGroup.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents method.

Pascal

```
function ImplGetSubComponents: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents method calls ImplGetSubComponents to retrieve the value of WebFieldControls as a TObject. TWebControlGroup descendants override this method if the list is not simply the value of WebFieldControls. For example, some TWebControlGroup descendants supply a default value (DefaultWebComponents) when the WebFieldControls property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 5516)

DefaultWebComponents (see page 5518)

WebFieldControls (see page 5519)

52.1.66.4.16 TXMLDisplayGroup.ImplIsMultipleRecordView Method

Pascal

```
function ImplIsMultipleRecordView: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsMultipleRecordView();
```

Description

This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.

52.1.66.4.17 TXMLDisplayGroup.IsFieldInUse Method

Pascal

```
function IsFieldInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(UnicodeString AName);
```

Description

This is IsFieldInUse, a member of class TXMLDisplayGroup.

52.1.66.4.18 TXMLDisplayGroup.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.66.4.19 TXMLDisplayGroup.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: string);
```

C++

```
__fastcall SetDataSetField(const UnicodeString Value);
```

Description

This is SetDataSetField, a member of class TXMLDisplayGroup.

52.1.66.4.20 TXMLDisplayGroup.SetEnableValidateFields Method

Pascal

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

C++

```
__fastcall SetEnableValidateFields(Boolean Value);
```

Description

This is SetEnableValidateFields, a member of class TXMLDisplayGroup.

52.1.66.4.21 TXMLDisplayGroup.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker);
```

C++

```
__fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLDisplayGroup.

52.1.66.4.22 TXMLDisplayGroup.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class TXMLDisplayGroup.

52.1.66.4.23 TXMLDisplayGroup.ValidateFields Method

Pascal

```
function ValidateFields(AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateFields(IAddScriptElements AddIntf);
```

Description

This is ValidateFields, a member of class TXMLDisplayGroup.

52.1.66.4.24 TXMLDisplayGroup.XMLDataChange Method

Pascal

```
procedure XMLDataChange(Sender: TObject); virtual;
```

C++

```
virtual __fastcall XMLDataChange(TObject * Sender);
```

Description

This is XMLDataChange, a member of class TXMLDisplayGroup.

52.1.66.5 TXMLDisplayGroup Properties

The properties of the TXMLDisplayGroup class are listed here.

Protected Properties

	Name	Description
	EnableValidateFields (see page 5630)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	XMLDataSet (see page 5631)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Properties

	Name	Description
R	VisibleFields (see page 5630)	Indicates the controls displayed by the HTML that the XML display group generates.

Published Properties

	Name	Description
	XMLBroker (see page 5631)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 5631)	Binds the XML display group to a nested dataset from the XML data packet.

52.1.66.5.1 TXMLDisplayGroup.EnableValidateFields Property

Pascal

```
property EnableValidateFields: Boolean;
```

C++

```
__property Boolean EnableValidateFields;
```

Description

This is EnableValidateFields, a member of class TXMLDisplayGroup.

52.1.66.5.2 TXMLDisplayGroup.VisibleFields Property

Indicates the controls displayed by the HTML that the XML display group generates.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Use VisibleFields to access the components that generate HTML controls for the XML display group. If a set of Web items is explicitly assigned to the WebFieldControls (see page 5519) property, VisibleFields has the same value as WebFieldControls (see page 5519). Otherwise, VisibleFields represents a set of default Web items that generate display controls for all the fields or parameters of the associated dataset or dataset field.

When no Web items are assigned to WebFieldControls (see page 5519), the default Web items in the VisibleFields list are specific to the TXMLDisplayGroup (see page 5620) descendant.

See Also

WebFieldControls (see page 5519)

xmlbroker (see page 5631)

XMLDataSetField (see page 5631)

52.1.66.5.3 TXMLDisplayGroup.XMLBroker Property

Indicates the XML broker that fetches the data packets to which the Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Use XMLBroker to access the TXMLBroker object that fetches XML data packets from the application server. The XML broker can be used to obtain information about the source of the dataset, the form that contains the XML data packet, the javascript variables that represent this data packet, and so on.

Note: The dataset represented by the Web item may not correspond directly to the XML broker's dataset. It may be a nested dataset contained by that dataset. The XMLDataSet property indicates whether the dataset is nested, and if so, describes the field that represents it.

See Also

XMLDataSet (see page 5647)

DataSet (see page 5646)

52.1.66.5.4 TXMLDisplayGroup.XMLDataSet Property

Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Pascal

```
property XMLDataSet: TXMLDataSet;
```

C++

```
__property TXMLDataSet XMLDataSet;
```

Description

XMLDataSet describes how to locate the appropriate dataset, given the XML broker that provides its XML data packet. This dataset may be the main dataset of the XML broker, or it may be a nested dataset that is represented by a dataset field within that dataset (or a dataset field nested in a dataset field, and so on).

See Also

DataSet (see page 5646)

XMLBroker (see page 5647)

52.1.66.5.5 TXMLDisplayGroup.XMLDataSetField Property

Binds the XML display group to a nested dataset from the XML data packet.

Pascal

```
property XMLDataSetField: string;
```

C++

```
__property UnicodeString XMLDataSetField;
```

Description

Most XML display groups contain controls that represent database information. TDataGrid (see page 5302) and TFieldGroup (see page 5320) represent the fields from a dataset. By default, TQueryFieldGroup (see page 5415) represents the parameter values for a query or stored procedure or field values that filter a table. If the database information comes from the dataset represented by an XML broker's data packet, XMLDataSetField is an empty string and the XMLBroker (see page 5631) property fully describes the dataset. However, if the database information comes from a nested detail set, XMLDataSetField indicates what detail dataset to use from the data packet associated with XMLBroker (see page 5631).

Specify the name of the dataset field as the value of XMLDataSetField. If the XML display group is bound to the detail set of a detail set, you can use periods to qualify dataset field names. Thus, for example, if XMLBroker (see page 5631) represents a customer dataset with a nested Orders dataset field that has its own nested LineItems field, you can bind the XML display group to the LineItems dataset by setting XMLDataSetField to

```
Orders.LineItems
```

See Also

xmlbroker (see page 5631)

VisibleFields (see page 5630)

Setting Web Item Properties

52.1.67 TXMLDisplayParent Class

Class Hierarchy

Pascal

```
TXMLDisplayParent = class(TWebDisplay);
```

C++

```
class TXMLDisplayParent : public TWebDisplay;
```

File

MidItems

Description

This is class MidItems.TXMLDisplayParent.

52.1.67.1 TXMLDisplayParent Members

The following tables list the members exposed by TXMLDisplayParent.

Protected Methods

	Name	Description
	Changed (see page 5542)	This is Changed, a member of class TWebDisplay.
V	ChangedDisplayComponent (see page 5633)	This is ChangedDisplayComponent, a member of class TXMLDisplayParent.
V	ChangedUseParent (see page 5634)	This is ChangedUseParent, a member of class TXMLDisplayParent.
V	GetDisplayComponent (see page 5634)	This is GetDisplayComponent, a member of class TXMLDisplayParent.
	SetDisplayComponent (see page 5542)	This is SetDisplayComponent, a member of class TWebDisplay.

	SetUseParent (see page 5634)	This is SetUseParent, a member of class TXMLDisplayParent.

Public Events

	Name	Description
	OnChange (see page 5543)	This is OnChange, a member of class TWebDisplay.

Public Methods

	Name	Description
	Create (see page 5633)	This is Create, a member of class TXMLDisplayParent.
V	Notification (see page 5542)	This is Notification, a member of class TWebDisplay.

Public Properties

	Name	Description
	DisplayComponent (see page 5543)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 5543)	This is Parent, a member of class TWebDisplay.

Published Properties

	Name	Description
	UseParent (see page 5634)	This is UseParent, a member of class TXMLDisplayParent.

52.1.67.2 TXMLDisplayParent.Create Constructor

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLDisplayParent(TComponent * AParent);
```

Description

This is Create, a member of class TXMLDisplayParent.

52.1.67.3 TXMLDisplayParent Methods

The methods of the TXMLDisplayParent class are listed here.

Protected Methods

	Name	Description
V	ChangedDisplayComponent (see page 5633)	This is ChangedDisplayComponent, a member of class TXMLDisplayParent.
V	ChangedUseParent (see page 5634)	This is ChangedUseParent, a member of class TXMLDisplayParent.
V	GetDisplayComponent (see page 5634)	This is GetDisplayComponent, a member of class TXMLDisplayParent.
	SetUseParent (see page 5634)	This is SetUseParent, a member of class TXMLDisplayParent.

52.1.67.3.1 TXMLDisplayParent.ChangedDisplayComponent Method

Pascal

```
procedure ChangedDisplayComponent; override;
```

C++

```
virtual __fastcall ChangedDisplayComponent();
```

Description

This is ChangedDisplayComponent, a member of class TXMLDisplayParent.

52.1.67.3.2 TXMLDisplayParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDisplayParent.

52.1.67.3.3 TXMLDisplayParent.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TXMLDisplayParent.

52.1.67.3.4 TXMLDisplayParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDisplayParent.

52.1.67.4 TXMLDisplayParent Properties

The properties of the TXMLDisplayParent class are listed here.

Published Properties

	Name	Description
	UseParent (see page 5634)	This is UseParent, a member of class TXMLDisplayParent.

52.1.67.4.1 TXMLDisplayParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

Description

This is UseParent, a member of class TXMLDisplayParent.

52.1.68 TXMLDisplayReferenceButton Class

TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.

Class Hierarchy

Pascal

```
TXMLDisplayReferenceButton = class(TWebButton);
```

C++

```
class TXMLDisplayReferenceButton : public TWebButton;
```

File

MidItems

Description

Use TXMLDisplayReferenceButton as a base class for components that generate the HTML statements defining a specialized button control for working with a data display control in the Web page produced by TInetXPageProducer (see page 5730).

TXMLDisplayReferenceButton defines properties and methods for

Generating the HTML code to define the button.

Associating the button with another Web item that generates a data display control.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TXMLDisplayReferenceButton inherits support for the IWebContent interface from TWebButton (see page 5504).

Do not create instances of TXMLDisplayReferenceButton. TXMLDisplayReferenceButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TXMLDisplayReferenceButton, such as TPostButton (see page 5396), TInsertButton (see page 5385), and so on.

See Also

TFirstButton (see page 5363)

TNextButton (see page 5392)

TPriorButton (see page 5399)

TLastButton (see page 5387)

TNextPageButton (see page 5394)

TPriorPageButton (see page 5401)

TInsertButton (see page 5385)

TDeleteButton (see page 5318)

TUndoButton (see page 5502)

TPostButton (see page 5396)

IWebContent

TDataNavigator (see page 5307)

TResetQueryButton (see page 5469)

TSubmitQueryButton (see page 5486)

52.1.68.1 TXMLDisplayReferenceButton Members

The following tables list the members exposed by TXMLDisplayReferenceButton.

Protected Methods

	Name	Description
	Content (see page 5506)	Returns the HTML that defines the button control.
V	GetCaption (see page 5507)	Returns the value of the Caption (see page 5508) property.
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
V	GetLayoutAttributes (see page 5507)	Returns the layout object that specifies how the generated control should look.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
A	ImplContent (see page 5507)	Provides the underlying implementation for the Content (see page 5506) method.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
V	SetCaption (see page 5508)	Sets the value of the Caption (see page 5508) property.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

Protected Properties

	Name	Description
R	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
V	Create (see page 5637)	Creates and initializes an instance of TXMLDisplayReferenceButton (see page 5635).
V	Destroy (see page 5637)	Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Public Properties

	Name	Description
	Caption (see page 5508)	Specifies the caption that appears on the button.
	Custom (see page 5509)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 5509)	Adds style attributes to the generated button control.
	StyleRule (see page 5510)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

52.1.68.2 TXMLDisplayReferenceButton.Create Constructor

Creates and initializes an instance of TXMLDisplayReferenceButton (see page 5635).

Pascal

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

C++

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

Description

Call Create to instantiate a dataset button at runtime. Buttons added at design time using the Web Page editor are instantiated automatically.

52.1.68.3 TXMLDisplayReferenceButton.Destroy Destructor

Frees an instance of TXMLDisplayReferenceButton (see page 5635).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDisplayReferenceButton();
```

Description

Destroy removes the button from its parent's list of components that generate HTML.

52.1.68.4 TXMLDisplayReferenceButton Methods

The methods of the TXMLDisplayReferenceButton class are listed here.

Protected Methods

	Name	Description
	GetDisplayComponent (see page 5637)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 5638)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetXmlDisplayName (see page 5638)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	Notification (see page 5638)	Forwards notification messages to all owned components.
	SetDisplayComponent (see page 5639)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 5639)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

52.1.68.4.1 TXMLDisplayReferenceButton.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.

52.1.68.4.2 TXMLDisplayReferenceButton.GetDisplayComponentParent Method

Pascal

```
function GetDisplayComponentParent: Boolean;
```

C++

```
__fastcall Boolean GetDisplayComponentParent();
```

Description

This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

52.1.68.4.3 TXMLDisplayReferenceButton.GetXmlDisplayName Method

Pascal

```
function GetXmlDisplayName: string;
```

C++

```
__fastcall UnicodeString GetXmlDisplayName();
```

Description

This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.

52.1.68.4.4 TXMLDisplayReferenceButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

52.1.68.4.5 TXMLDisplayReferenceButton.SetDisplayComponent Method

Pascal

```
procedure SetDisplayComponent(const Value: TComponent);
```

C++

```
__fastcall SetDisplayComponent(const TComponent * Value);
```

Description

This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.

52.1.68.4.6 TXMLDisplayReferenceButton.SetDisplayComponentParent Method

Pascal

```
procedure SetDisplayComponentParent(const Value: Boolean);
```

C++

```
__fastcall SetDisplayComponentParent(const Boolean Value);
```

Description

This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

52.1.68.5 TXMLDisplayReferenceButton Properties

The properties of the TXMLDisplayReferenceButton class are listed here.

Protected Properties

	Name	Description
	XMLDisplay (see page 5640)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Properties

	Name	Description
	XMLComponent (see page 5639)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 5640)	Specifies whether the button is associated with its parent's display object.

52.1.68.5.1 TXMLDisplayReferenceButton.XMLComponent Property

Associates the button with the component that generates the display control it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates a display control that the button influences.

Note: Typically, descendants of TXMLDisplayReferenceButton (see page 5635) are used by a descendant of TXMLDisplayReferenceGroup (see page 5640), which has its own XMLComponent property. If the button influences this same component, you need not set the XMLComponent property. Instead, setting the XMLUseParent (see page 5640) property to true (the default) sets XMLComponent to the value of the parent's XMLComponent property.

See Also

XMLUseParent (see page 5640)

52.1.68.5.2 TXMLDisplayReferenceButton.XMLDisplay Property

Pascal

```
property XMLDisplay: TXMLDisplayParent;
```

C++

```
__property TXMLDisplayParent XMLDisplay;
```

Description

This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

52.1.68.5.3 TXMLDisplayReferenceButton.XMLUseParent Property

Specifies whether the button is associated with its parent's display object.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the dataset button determines the component that it influences. When XMLUseParent is false, the XMLComponent (see page 5639) property must be set to indicate the display component. When XMLUseParent is true, the button influences the component specified by its parent's XMLComponent (see page 5639) property.

The button's parent is the Web item that uses the button to assemble its own HTML content (typically a TDataNavigator (see page 5307) or TQueryButtons (see page 5412) object). If the parent does not have an associated display component, setting XMLUseParent to true sets the XMLComponent (see page 5639) property to nil (Delphi) or NULL (C++).

See Also

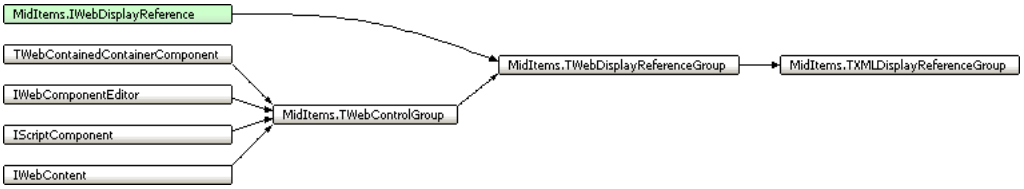
XMLComponent (see page 5639)

XMLComponent (see page 5643)

52.1.69 TXMLDisplayReferenceGroup Class

TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

Class Hierarchy



Pascal

```
TXMLDisplayReferenceGroup = class(TWebDisplayReferenceGroup);
```

C++

```
class TXMLDisplayReferenceGroup : public TWebDisplayReferenceGroup;
```

File

MidItems

Description

Use TXMLDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of button controls. Each TXMLDisplayGroup (see page 5620) object has an associated Web item that generates controls that display data in the final HTML document. The buttons arranged by TXMLDisplayReferenceGroup manipulate the information the display controls show.

In order to let other Web items use the HTML that the TXMLDisplayReferenceGroup object generates, TXMLDisplayReferenceGroup inherits the IWebContent interface from TWebControlGroup (see page 5510).

Do not create instances of TXMLDisplayReferenceGroup. TXMLDisplayReferenceGroup manages the Web items that generate buttons and the association to a display generator, but does not produce any content. Instead, use a descendant of TXMLDisplayReferenceGroup that generates the HTML for its set of specialized buttons such as TDataNavigator (see page 5307) or TQueryButtons (see page 5412).

See Also

TDataNavigator (see page 5307)

IWebContent

52.1.69.1 TXMLDisplayReferenceGroup Members

The following tables list the members exposed by TXMLDisplayReferenceGroup.

Protected Methods

	Name	Description
	AddElements (see page 5513)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 5513)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 5519) property list.
	ChangeFieldControls (see page 5514)	Responds when the group of HTML controls changes.
	Content (see page 5514)	Returns the HTML that defines the group of HTML controls.
V	GetDefaultWebComponents (see page 5515)	Returns the value of the DefaultWebComponents (see page 5518) property.
V	GetLayoutAttributes (see page 5515)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 5516)	Returns the value of the WebFieldControls (see page 5519) property.
	GetWebDisplayComponent (see page 5546)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

	ImplAddElements (see page 5642)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 5516)	Provides the underlying implementation for the CanAddClass (see page 5513) method.
	ImplContent (see page 5517)	Provides the underlying implementation for the Content (see page 5514) method.
	ImplGetSubComponents (see page 5517)	Provides the underlying implementation for the GetSubComponents (see page 5516) method.
	Notification (see page 5546)	Forwards notification messages to all owned components.
	SetDefaultWebComponents (see page 5518)	Sets the value of the DefaultWebComponents (see page 5518) property.
	SetWebDisplayComponent (see page 5547)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 5518)	Specifies the default value of the WebFieldControls (see page 5519) property when that property is not explicitly set.
R	WebDisplay (see page 5547)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 5547)	

Public Methods

	Name	Description
	Create (see page 5512)	Creates and initializes an instance of TWebControlGroup (see page 5510).
	Create (see page 5545)	Creates an instance of TWebDisplayReferenceGroup (see page 5544).
	Destroy (see page 5512)	Frees an instance of TWebControlGroup (see page 5510).
	Destroy (see page 5546)	Frees an instance of TWebDisplayReferenceGroup (see page 5544).
	GetWebDisplayComponent (see page 5672)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

Public Properties

	Name	Description
R	WebDisplayComponent (see page 5672)	
R	WebFieldControls (see page 5519)	Lists the components that generate HTML controls for the Web control group.
	XMLComponent (see page 5643)	Associates the group with a component that generates HTML display controls.

52.1.69.2 TXMLDisplayReferenceGroup Methods

The methods of the TXMLDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	ImplAddElements (see page 5642)	Provides the underlying implementation for the AddElements method.

52.1.69.2.1 TXMLDisplayReferenceGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 5513)

52.1.69.3 TXMLDisplayReferenceGroup Properties

The properties of the TXMLDisplayReferenceGroup class are listed here.

Public Properties

	Name	Description
	XMLComponent (see page 5643)	Associates the group with a component that generates HTML display controls.

52.1.69.3.1 TXMLDisplayReferenceGroup.XMLComponent Property

Associates the group with a component that generates HTML display controls.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Use XMLComponent to associate the group of buttons with a Web item that generates the display controls those buttons influence. Typical values of XMLComponent are TFieldGroup (see page 5320) or TDataGrid (see page 5302).

If all the buttons arranged by the TXMLDisplayReferenceGroup (see page 5640) object operate on the same set of display controls, it is most efficient to specify the Web item generating these controls as the value of XMLComponent. Then, each button uses this value to determine its associated display controls when its XMLUseParent property is true. Any buttons that do not operate on the display controls generated by XMLComponent can set their own XMLComponent property to another Web item.

See Also

XMLComponent (see page 5639)

XMLUseParent (see page 5640)

TFieldGroup (see page 5320)

TDataGrid (see page 5302)

TQueryFieldGroup (see page 5415)

52.2 Interfaces

The following table lists interfaces in this documentation.

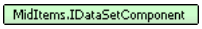
Interfaces

	Name	Description
	IDataSetComponent (see page 5644)	IDataSetComponent defines the methods that bind a Web item to an XML broker.
	IDataSetField (see page 5648)	This is class MidItems.IDataSetField.
	IDataSetFields (see page 5650)	This is class MidItems.IDataSetFields.
	IFormatColumn (see page 5652)	This is class MidItems.IFormatColumn.
	IHTMLField (see page 5653)	IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.
	IHTMLForm (see page 5655)	IHTMLForm is an interface that must be implemented by Web items that generate forms.
	IInetXWebDataSet (see page 5657)	IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.
	IQueryField (see page 5661)	This is class MidItems.IQueryField.
	IQueryFields (see page 5664)	This is class MidItems.IQueryFields.
	IQueryForm (see page 5665)	This is class MidItems.IQueryForm.
	IRestoreDefaults (see page 5665)	This is class MidItems.IRestoreDefaults.
	IStatusField (see page 5666)	This is class MidItems.IStatusField.
	IValidateField (see page 5667)	This is class MidItems.IValidateField.
	IValidateFields (see page 5668)	This is class MidItems.IValidateFields.
	IWebDisplay (see page 5670)	
	IWebDisplayReference (see page 5671)	
	IXMLDisplay (see page 5673)	This is class MidItems.IXMLDisplay.

52.2.1 IDataSetComponent Interface

IDataSetComponent defines the methods that bind a Web item to an XML broker.

Class Hierarchy



Pascal

```
IDataSetComponent = interface;
```

C++

```
__interface IDataSetComponent;
```

File

MidItems

Description

The TXMLDisplayGroup (see page 5620) class implements IDataSetComponent to provide information about the dataset from which its child controls obtain their data. The child Web items use this information to determine the default data binding.

Most Web items that generate a group of data-aware controls inherit the support for IDataSetComponent from TXMLDisplayGroup (see page 5620).

See Also

TXMLDisplayGroup (see page 5620)

TXMLBroker

52.2.1.1 IDataSetComponent Members

The following tables list the members exposed by IDataSetComponent.

Public Methods

	Name	Description
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.

Public Properties

	Name	Description
R	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
R	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.

52.2.1.2 IDataSetComponent Methods

The methods of the IDataSetComponent class are listed here.

Public Methods

	Name	Description
	GetDataSet (see page 5645)	Returns the value of the DataSet (see page 5646) property.
	GetXMLBroker (see page 5645)	Returns the value of the XMLBroker (see page 5647) property.
	GetXMLDataSet (see page 5646)	Returns the value of the XMLDataSet (see page 5647) property.

52.2.1.2.1 IDataSetComponent.GetDataSet Method

Returns the value of the DataSet (see page 5646) property.

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

GetDataSet is the read access method for the DataSet (see page 5646) property.

See Also

DataSet (see page 5646)

52.2.1.2.2 IDataSetComponent.GetXMLBroker Method

Returns the value of the XMLBroker (see page 5647) property.

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

GetXMLBroker is the read access method for the XMLBroker (see page 5647) property.

See Also

XMLBroker (see page 5647)

52.2.1.2.3 IDataSetComponent.GetXMLDataSet Method

Returns the value of the XMLDataSet (see page 5647) property.

Pascal

```
function GetXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet GetXMLDataSet();
```

Description

GetXMLDataSet is the read access method for the XMLDataSet (see page 5647) property.

See Also

XMLDataSet (see page 5647)

52.2.1.3 IDataSetComponent Properties

The properties of the IDataSetComponent class are listed here.

Public Properties

	Name	Description
R	DataSet (see page 5646)	Provides access to an interface for working with the associated dataset.
R	XMLBroker (see page 5647)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 5647)	Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.

52.2.1.3.1 IDataSetComponent.DataSet Property

Provides access to an interface for working with the associated dataset.

Pascal

```
property DataSet: IInetXWebDataSet;
```

C++

```
__property IInetXWebDataSet DataSet;
```

Description

The child Web items of the item that implements IDataSetComponent (see page 5644) use the DataSet property to obtain information about the dataset to which they are bound and to respond when this information changes. This information includes

details about the available fields and parameters in the dataset to which they can be bound.

See Also

XMLDataSet (see page 5647)

52.2.1.3.2 IDataSetComponent.XMLBroker Property

Indicates the XML broker that fetches the data packets to which the Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Use XMLBroker to access the TXMLBroker object that fetches XML data packets from the application server. The XML broker can be used to obtain information about the source of the dataset, the form that contains the XML data packet, the javascript variables that represent this data packet, and so on.

Note: The dataset represented by the Web item may not correspond directly to the XML broker's dataset. It may be a nested dataset contained by that dataset. The XMLDataSet (see page 5647) property indicates whether the dataset is nested, and if so, describes the field that represents it.

See Also

XMLDataSet (see page 5647)

DataSet (see page 5646)

52.2.1.3.3 IDataSetComponent.XMLDataSet Property

Indicates the TXMLDataSet (see page 5608) object that represents the dataset to which the Web item is bound.

Pascal

```
property XMLDataSet: TXMLDataSet;
```

C++

```
__property TXMLDataSet XMLDataSet;
```

Description

XMLDataSet describes how to locate the appropriate dataset, given the XML broker that provides its XML data packet. This dataset may be the main dataset of the XML broker, or it may be a nested dataset that is represented by a dataset field within that dataset (or a dataset field nested in a dataset field, and so on).

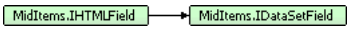
See Also

DataSet (see page 5646)

XMLBroker (see page 5647)

52.2.2 IDataSetField Interface

Class Hierarchy



Pascal

```
IDataSetField = interface(IHTMLField);
```

C++

```
__interface IDataSetField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IDataSetField.

52.2.2.1 IDataSetField Members

The following tables list the members exposed by IDataSetField.

Public Methods

	Name	Description
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.

Public Properties

	Name	Description
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

52.2.2.2 IDataSetField Methods

The methods of the IDataSetField class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 5648)	This is GetFieldName, a member of class IDataSetField.
	GetRowSetFieldAttributes (see page 5649)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	SetFieldName (see page 5649)	This is SetFieldName, a member of class IDataSetField.

52.2.2.2.1 IDataSetField.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class IDataSetField.

52.2.2.2.2 IDataSetField.GetRowSetFieldAttributes Method

Pascal

```
function GetRowSetFieldAttributes(const FieldVarName: string): string;
```

C++

```
__fastcall UnicodeString GetRowSetFieldAttributes(const UnicodeString FieldVarName);
```

Description

This is GetRowSetFieldAttributes, a member of class IDataSetField.

52.2.2.2.3 IDataSetField.SetFieldName Method

Pascal

```
procedure SetFieldName(AFieldName: string);
```

C++

```
__fastcall SetFieldName(UnicodeString AFieldName);
```

Description

This is SetFieldName, a member of class IDataSetField.

52.2.2.3 IDataSetField Properties

The properties of the IDataSetField class are listed here.

Public Properties

	Name	Description
	FieldName (see page 5649)	This is FieldName, a member of class IDataSetField.

52.2.2.3.1 IDataSetField.FieldName Property

Pascal

```
property FieldName: string;
```

C++

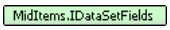
```
__property UnicodeString FieldName;
```

Description

This is FieldName, a member of class IDataSetField.

52.2.3 IDataSetFields Interface

Class Hierarchy



Pascal

```
IDataSetFields = interface;
```

C++

```
__interface IDataSetFields;
```

File

MidItems

Description

This is class MidItems.IDataSetFields.

52.2.3.1 IDataSetFields Members

The following tables list the members exposed by IDataSetFields.

Public Methods

	Name	Description
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.

Public Properties

	Name	Description
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.

52.2.3.2 IDataSetFields Methods

The methods of the IDataSetFields class are listed here.

Public Methods

	Name	Description
	GetFieldsList (see page 5651)	This is GetFieldsList, a member of class IDataSetFields.
	GetVisibleFields (see page 5651)	This is GetVisibleFields, a member of class IDataSetFields.
	HasStatusField (see page 5651)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 5651)	This is IsFieldInUse, a member of class IDataSetFields.
	ValidateDataSet (see page 5651)	This is ValidateDataSet, a member of class IDataSetFields.

52.2.3.2.1 IDataSetFields.GetFieldsList Method

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

This is GetFieldsList, a member of class IDataSetFields.

52.2.3.2.2 IDataSetFields.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class IDataSetFields.

52.2.3.2.3 IDataSetFields.HasStatusField Method

Pascal

```
function HasStatusField: Boolean;
```

C++

```
__fastcall Boolean HasStatusField();
```

Description

This is HasStatusField, a member of class IDataSetFields.

52.2.3.2.4 IDataSetFields.IsFieldInUse Method

Pascal

```
function IsFieldInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(UnicodeString AName);
```

Description

This is IsFieldInUse, a member of class IDataSetFields.

52.2.3.2.5 IDataSetFields.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class IDataSetFields.

52.2.3.3 IDataSetFields Properties

The properties of the IDataSetFields class are listed here.

Public Properties

	Name	Description
	VisibleFields (see page 5652)	This is VisibleFields, a member of class IDataSetFields.

52.2.3.3.1 IDataSetFields.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class IDataSetFields.

52.2.4 IFormatColumn Interface

Class Hierarchy

MidItems.IFormatColumn

Pascal

```
IFormatColumn = interface;
```

C++

```
__interface IFormatColumn;
```

File

MidItems

Description

This is class MidItems.IFormatColumn.

52.2.4.1 IFormatColumn Members

The following tables list the members exposed by IFormatColumn.

Public Methods

	Name	Description
	FormatColumnData (see page 5653)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 5653)	This is FormatColumnHeading, a member of class IFormatColumn.

52.2.4.2 IFormatColumn Methods

The methods of the IFormatColumn class are listed here.

Public Methods

	Name	Description
	FormatColumnData (see page 5653)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 5653)	This is FormatColumnHeading, a member of class IFormatColumn.

52.2.4.2.1 IFormatColumn.FormatColumnData Method

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnData(const UnicodeString Content, TWebContentOptions
Options);
```

Description

This is FormatColumnData, a member of class IFormatColumn.

52.2.4.2.2 IFormatColumn.FormatColumnHeading Method

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnHeading(TWebContentOptions Options);
```

Description

This is FormatColumnHeading, a member of class IFormatColumn.

52.2.5 IHTMLField Interface

IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.

Class Hierarchy

MidItems.IHTMLField

Pascal

```
IHTMLField = interface;
```

C++

```
__interface IHTMLField;
```

File

MidItems

Description

The TWebDataDisplay (see page 5519) class implements IHTMLField to provide information about information about the HTML control it represents. This interface is used when generating javascript that refers to the control.

Most Web items that generate HTML <INPUT> elements inherit the support for IHTMLField from TWebDataDisplay (see page 5519). However, when writing custom Web items that generate <INPUT> tags but that do not descend from TWebDataDisplay (see page 5519), component writers must ensure that the Web item supports the IHTMLField interface.

See Also

TWebDataDisplay (see page 5519)

52.2.5.1 IHTMLField Members

The following tables list the members exposed by IHTMLField.

Public Methods

	Name	Description
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

52.2.5.2 IHTMLField Methods

The methods of the IHTMLField class are listed here.

Public Methods

	Name	Description
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.

52.2.5.2.1 IHTMLField.GetHTMLControlName Method

Returns the name of the generated HTML <INPUT> control.

Pascal

```
function GetHTMLControlName: string;
```

C++

```
__fastcall UnicodeString GetHTMLControlName();
```

Description

GetHTMLFormName is the read access method for the HTMLControlName (see page 5655) property.

See Also

HTMLControlName (see page 5655)

52.2.5.3 IHTMLField Properties

The properties of the IHTMLField class are listed here.

Public Properties

	Name	Description
	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.

52.2.5.3.1 IHTMLField.HTMLControlName Property

Indicates the name of the generated HTML control.

Pascal

```
property HTMLControlName: string;
```

C++

```
__property UnicodeString HTMLControlName;
```

Description

HTMLControlName is the value of the NAME option on the generated <INPUT> tag. This value is used when creating javascript references to the control.

52.2.6 IHTMLForm Interface

IHTMLForm is an interface that must be implemented by Web items that generate forms.

Class Hierarchy

MidItems.IHTMLForm

Pascal

```
IHTMLForm = interface;
```

C++

```
__interface IHTMLForm;
```

File

MidItems

Description

Web items that produce forms (such as TQueryForm (see page 5421) or TDataForm (see page 5298)) must implement the IHTML form interface. This interface allows other Web items to access information about form elements when generating javascript declarations and method calls.

See Also

TQueryForm (see page 5421)

TDataForm (see page 5298)

52.2.6.1 IHTMLForm Members

The following tables list the members exposed by IHTMLForm.

Public Methods

	Name	Description
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
R	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.

52.2.6.2 IHTMLForm Methods

The methods of the IHTMLForm class are listed here.

Public Methods

	Name	Description
	GetHTMLFormName (see page 5656)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 5656)	Returns the name of the javascript variable that refers to the generated form.

52.2.6.2.1 IHTMLForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormName();
```

Description

GetHTMLFormName is the read access method for the HTMLFormName (see page 5657) property.

See Also

HTMLFormName (see page 5657)

52.2.6.2.2 IHTMLForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

GetHTMLFormVarName is the read access method for the HTMLFormVarName (see page 5657) property.

See Also

HTMLFormVarName (see page 5657)

52.2.6.3 IHTMLForm Properties

The properties of the IHTMLForm class are listed here.

Public Properties

	Name	Description
	HTMLFormName (see page 5657)	Indicates the name of the generated HTML form.
	HTMLFormVarName (see page 5657)	Indicates the name of the javascript variable that represents the HTML form.

52.2.6.3.1 IHTMLForm.HTMLFormName Property

Indicates the name of the generated HTML form.

Pascal

```
property HTMLFormName: string;
```

C++

```
__property UnicodeString HTMLFormName;
```

Description

HTMLFormName is the value of the NAME option on the generated form. Other Web items can use this property when they need to refer to the form name. For example, this property is used to declare the javascript variable that represents the form.

See Also

HTMLFormVarName (see page 5657)

52.2.6.3.2 IHTMLForm.HTMLFormVarName Property

Indicates the name of the javascript variable that represents the HTML form.

Pascal

```
property HTMLFormVarName: string;
```

C++

```
__property UnicodeString HTMLFormVarName;
```

Description

HTMLFormVarName is the value of the javascript variable for the form. Other Web items can use this property when they need to refer to the form in generated javascript code.

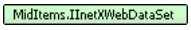
See Also

HTMLFormName (see page 5657)

52.2.7 IInetXWebDataSet Interface

IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.

Class Hierarchy



Pascal

```
IInetXWebDataSet = interface;
```

C++

```
__interface IInetXWebDataSet;
```

File

MidItems

Description

Web items use the IInetXWebDataSet interface (formerly known as IMidasWebDataSet (see page 5684)) to obtain information about a dataset supplied by an XML broker. By using this interface, the Web item does not need to worry about whether the dataset is the top-level dataset in the XML data packet or a nested dataset in that data packet.

IInetXWebDataSet defines methods for obtaining information about the fields and parameters of the associated dataset. It also defines methods for assigning an event handler that receives notifications when this information changes.

Note: VCL Method declarations that use IInetXWebDataSet use the _di_IInetXWebDataSet type instead. This type is a DelphiInterface wrapper around the IInetXWebDataSet interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInetXWebDataSet >
_di_IInetXWebDataSet;
```

See Also

TXMLDataSet (see page 5608)

TXMLBroker

52.2.7.1 IInetXWebDataSet Members

The following tables list the members exposed by IInetXWebDataSet.

Public Methods

	Name	Description
	AddOnDataChange (see page 5659)	Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.
	FetchParams (see page 5659)	Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.
	FieldCount (see page 5659)	Returns the number of fields in the associated dataset.
	Fields (see page 5660)	Returns the set of fields in the associated dataset.
	ParamCount (see page 5660)	Returns the number of parameters for the associated dataset.
	Params (see page 5660)	Returns the set of parameters in the associated dataset.
	RemoveOnDataChange (see page 5661)	Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

52.2.7.2 IInetXWebDataSet Methods

The methods of the IInetXWebDataSet class are listed here.

Public Methods

	Name	Description
	AddOnDataChange (see page 5659)	Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.

	FetchParams (see page 5659)	Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.
	FieldCount (see page 5659)	Returns the number of fields in the associated dataset.
	Fields (see page 5660)	Returns the set of fields in the associated dataset.
	ParamCount (see page 5660)	Returns the number of parameters for the associated dataset.
	Params (see page 5660)	Returns the set of parameters in the associated dataset.
	RemoveOnDataChange (see page 5661)	Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

52.2.7.2.1 IInetXWebDataSet.AddOnDataChange Method

Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.

Pascal

```
procedure AddOnDataChange(const Value: TNotifyEvent);
```

C++

```
__fastcall AddOnDataChange(const TNotifyEvent Value);
```

Description

Call AddOnDataChange to specify an event handler that is called when the fields or parameters of the dataset change. This allows Web items to update their data bindings to reflect changes in the associated data packet.

Value specifies the event handler that is added to the list.

See Also

RemoveOnDataChange (see page 5661)

52.2.7.2.2 IInetXWebDataSet.FetchParams Method

Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.

Pascal

```
procedure FetchParams;
```

C++

```
__fastcall FetchParams();
```

Description

Call FetchParams to tell the XML broker to fetch parameter values from the application server. This ensures that the Params (see page 5660) method returns the current values of any parameters on the server.

See Also

FetchParams

Params (see page 5660)

52.2.7.2.3 IInetXWebDataSet.FieldCount Method

Returns the number of fields in the associated dataset.

Pascal

```
function FieldCount: Integer;
```

C++

```
__fastcall int FieldCount();
```

Description

Call FieldCount to determine the number of fields returned by the Fields (see page 5660) method. This is the number of fields in the XML data packet belonging to the associated dataset.

See Also

Fields (see page 5660)

52.2.7.2.4 IInetXWebDataSet.Fields Method

Returns the set of fields in the associated dataset.

Pascal

```
function Fields: TFields;
```

C++

```
__fastcall TFields Fields();
```

Description

Call Fields to obtain access to the field objects (TField) that represent the fields in the XML data packet. Each field object can be used to read the value of the field, determine its type, and so on.

See Also

FieldCount (see page 5659)

52.2.7.2.5 IInetXWebDataSet.ParamCount Method

Returns the number of parameters for the associated dataset.

Pascal

```
function ParamCount: Integer;
```

C++

```
__fastcall int ParamCount();
```

Description

Call ParamCount to determine the number of parameters returned by the Params (see page 5660) method. This is the number of parameters in the XML data packet belonging to the associated dataset.

See Also

Params (see page 5660)

52.2.7.2.6 IInetXWebDataSet.Params Method

Returns the set of parameters in the associated dataset.

Pascal

```
function Params: TParams;
```

C++

```
__fastcall TParams Params();
```

Description

Call Params to obtain access to the parameter objects (TParam) that represent the parameters in the XML data packet. Each parameter object can be used to read the value of the parameter, determine its type, and so on.

See Also

ParamCount (see page 5660)

FetchParams (see page 5659)

52.2.7.2.7 IInetXWebDataSet.RemoveOnDataChange Method

Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

Pascal

```
procedure RemoveOnDataChange(const Value: TNotifyEvent);
```

C++

```
__fastcall RemoveOnDataChange(const TNotifyEvent Value);
```

Description

Call RemoveOnDataChange to remove an event handler that was added using the AddOnDataChange (see page 5659) method.

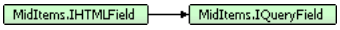
Value specifies the event handler to remove from the list.

See Also

AddOnDataChange (see page 5659)

52.2.8 IQueryField Interface

Class Hierarchy



Pascal

```
IQueryField = interface(IHTMLField);
```

C++

```
__interface IQueryField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IQueryField.

52.2.8.1 IQueryField Members

The following tables list the members exposed by IQueryField.

Public Methods

	Name	Description
	GetHTMLControlName (see page 5654)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetText (see page 5662)	This is GetText, a member of class IQueryField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	SetText (see page 5663)	This is SetText, a member of class IQueryField.

Public Properties

	Name	Description
R	HTMLControlName (see page 5655)	Indicates the name of the generated HTML control.
	ParamName (see page 5663)	This is ParamName, a member of class IQueryField.
	Text (see page 5663)	This is Text, a member of class IQueryField.

52.2.8.2 IQueryField Methods

The methods of the IQueryField class are listed here.

Public Methods

	Name	Description
	GetParamName (see page 5662)	This is GetParamName, a member of class IQueryField.
	GetText (see page 5662)	This is GetText, a member of class IQueryField.
	SetParamName (see page 5662)	This is SetParamName, a member of class IQueryField.
	SetText (see page 5663)	This is SetText, a member of class IQueryField.

52.2.8.2.1 IQueryField.GetParamName Method

Pascal

```
function GetParamName: string;
```

C++

```
__fastcall UnicodeString GetParamName();
```

Description

This is GetParamName, a member of class IQueryField.

52.2.8.2.2 IQueryField.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall UnicodeString GetText();
```

Description

This is GetText, a member of class IQueryField.

52.2.8.2.3 IQueryField.SetParamName Method

Pascal

```
procedure SetParamName(AParamName: string);
```

C++

```
__fastcall SetParamName(UnicodeString AParamName);
```

Description

This is SetParamName, a member of class IQueryField.

52.2.8.2.4 IQueryField.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const UnicodeString Value);
```

Description

This is SetText, a member of class IQueryField.

52.2.8.3 IQueryField Properties

The properties of the IQueryField class are listed here.

Public Properties

	Name	Description
	ParamName (see page 5663)	This is ParamName, a member of class IQueryField.
	Text (see page 5663)	This is Text, a member of class IQueryField.

52.2.8.3.1 IQueryField.ParamName Property

Pascal

```
property ParamName: string;
```

C++

```
__property UnicodeString ParamName;
```

Description

This is ParamName, a member of class IQueryField.

52.2.8.3.2 IQueryField.Text Property

Pascal

```
property Text: string;
```

C++

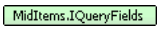
```
__property UnicodeString Text;
```

Description

This is Text, a member of class IQueryField.

52.2.9 IQueryFields Interface

Class Hierarchy



Pascal

```
IQueryFields = interface;
```

C++

```
__interface IQueryFields;
```

File

MidItems

Description

This is class MidItems.IQueryFields.

52.2.9.1 IQueryFields Members

The following tables list the members exposed by IQueryFields.

Public Methods

	Name	Description
	GetParamsList (see page 5664)	This is GetParamsList, a member of class IQueryFields.
	IsParamInUse (see page 5665)	This is IsParamInUse, a member of class IQueryFields.
	ValidateDataSet (see page 5665)	This is ValidateDataSet, a member of class IQueryFields.

52.2.9.2 IQueryFields Methods

The methods of the IQueryFields class are listed here.

Public Methods

	Name	Description
	GetParamsList (see page 5664)	This is GetParamsList, a member of class IQueryFields.
	IsParamInUse (see page 5665)	This is IsParamInUse, a member of class IQueryFields.
	ValidateDataSet (see page 5665)	This is ValidateDataSet, a member of class IQueryFields.

52.2.9.2.1 IQueryFields.GetParamsList Method

Pascal

```
procedure GetParamsList(AList: TStrings);
```

C++

```
__fastcall GetParamsList(TStrings AList);
```

Description

This is GetParamsList, a member of class IQueryFields.

52.2.9.2.2 IQueryFields.IsParamInUse Method

Pascal

```
function IsParamInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsParamInUse(UnicodeString AName);
```

Description

This is IsParamInUse, a member of class IQueryFields.

52.2.9.2.3 IQueryFields.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class IQueryFields.

52.2.10 IQueryForm Interface

Class Hierarchy

MidItems.IQueryForm

Pascal

```
IQueryForm = interface;
```

C++

```
__interface IQueryForm;
```

File

MidItems

Description

This is class MidItems.IQueryForm.

52.2.11 IRestoreDefaults Interface

Class Hierarchy

MidItems.IRestoreDefaults

Pascal

```
IRestoreDefaults = interface;
```

C++

```
__interface IRestoreDefaults;
```

File

MidItems

Description

This is class MidItems.IRestoreDefaults.

52.2.11.1 IRestoreDefaults Members

The following tables list the members exposed by IRestoreDefaults.

Public Methods

	Name	Description
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.

52.2.11.2 IRestoreDefaults Methods

The methods of the IRestoreDefaults class are listed here.

Public Methods

	Name	Description
	RestoreDefaults (see page 5666)	This is RestoreDefaults, a member of class IRestoreDefaults.

52.2.11.2.1 IRestoreDefaults.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

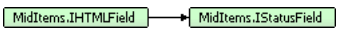
```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class IRestoreDefaults.

52.2.12 IStatusField Interface

Class Hierarchy



Pascal

```
IStatusField = interface(IHTMLField);
```

C++

```
__interface IStatusField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IStatusField.

52.2.13 IValidateField Interface

Class Hierarchy

MidItems.IValidateField

Pascal

```
IValidateField = interface;
```

C++

```
__interface IValidateField;
```

File

MidItems

Description

This is class MidItems.IValidateField.

52.2.13.1 IValidateField Members

The following tables list the members exposed by IValidateField.

Public Methods

	Name	Description
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

52.2.13.2 IValidateField Methods

The methods of the IValidateField class are listed here.

Public Methods

	Name	Description
	ValidateField (see page 5667)	This is ValidateField, a member of class IValidateField.

52.2.13.2.1 IValidateField.ValidateField Method

Pascal

```
function ValidateField(DataSet: IInetXWebDataSet; AddIntf: IAddScriptElements): Boolean;
```

C++

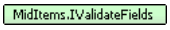
```
__fastcall Boolean ValidateField(IInetXWebDataSet DataSet, IAddScriptElements AddIntf);
```

Description

This is ValidateField, a member of class IValidateField.

52.2.14 IValidateFields Interface

Class Hierarchy



Pascal

```
IValidateFields = interface;
```

C++

```
__interface IValidateFields;
```

File

MidItems

Description

This is class MidItems.IValidateFields.

52.2.14.1 IValidateFields Members

The following tables list the members exposed by IValidateFields.

Public Methods

	Name	Description
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.

52.2.14.2 IValidateFields Methods

The methods of the IValidateFields class are listed here.

Public Methods

	Name	Description
	GetEnableValidateFields (see page 5669)	This is GetEnableValidateFields, a member of class IValidateFields.
	SetEnableValidateFields (see page 5669)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateFields (see page 5669)	This is ValidateFields, a member of class IValidateFields.

52.2.14.2.1 IValidateFields.GetEnableValidateFields Method

Pascal

```
function GetEnableValidateFields: Boolean;
```

C++

```
__fastcall Boolean GetEnableValidateFields();
```

Description

This is GetEnableValidateFields, a member of class IValidateFields.

52.2.14.2.2 IValidateFields.SetEnableValidateFields Method

Pascal

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

C++

```
__fastcall SetEnableValidateFields(Boolean Value);
```

Description

This is SetEnableValidateFields, a member of class IValidateFields.

52.2.14.2.3 IValidateFields.ValidateFields Method

Pascal

```
function ValidateFields(AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateFields(IAddScriptElements AddIntf);
```

Description

This is ValidateFields, a member of class IValidateFields.

52.2.14.3 IValidateFields Properties

The properties of the IValidateFields class are listed here.

Public Properties

	Name	Description
	EnableValidateFields (see page 5669)	This is EnableValidateFields, a member of class IValidateFields.

52.2.14.3.1 IValidateFields.EnableValidateFields Property

Pascal

```
property EnableValidateFields: Boolean;
```

C++

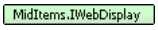
```
__property Boolean EnableValidateFields;
```

Description

This is EnableValidateFields, a member of class IValidateFields.

52.2.15 IWebDisplay Interface

Class Hierarchy



Pascal

```
IWebDisplay = interface;
```

C++

```
__interface IWebDisplay;
```

File

MidItems

See Also

TWebDisplay (see page 5540)

52.2.15.1 IWebDisplay Members

The following tables list the members exposed by IWebDisplay.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.

Public Properties

	Name	Description
	IsMultipleRecordView (see page 5671)	

52.2.15.2 IWebDisplay Methods

The methods of the IWebDisplay class are listed here.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 5670)	This is GetIsMultipleRecordView, a member of class IWebDisplay.

52.2.15.2.1 IWebDisplay.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class IWebDisplay.

52.2.15.3 IWebDisplay Properties

The properties of the IWebDisplay class are listed here.

Public Properties

	Name	Description
	IsMultipleRecordView (see page 5671)	

52.2.15.3.1 IWebDisplay.IsMultipleRecordView Property

Pascal

```
property IsMultipleRecordView: Boolean;
```

C++

```
__property Boolean IsMultipleRecordView;
```

Description

.

See Also

GetIsMultipleRecordView (see page 5670)

52.2.16 IWebDisplayReference Interface

Class Hierarchy

MidItems.IWebDisplayReference

Pascal

```
IWebDisplayReference = interface;
```

C++

```
__interface IWebDisplayReference;
```

File

MidItems

See Also

TWebDisplay (see page 5540)

52.2.16.1 IWebDisplayReference Members

The following tables list the members exposed by IWebDisplayReference.

Public Methods

	Name	Description
	GetWebDisplayComponent (see page 5672)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

Public Properties

	Name	Description
	WebDisplayComponent (see page 5672)	

52.2.16.2 IWebDisplayReference Methods

The methods of the IWebDisplayReference class are listed here.

Public Methods

	Name	Description
	GetWebDisplayComponent (see page 5672)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

52.2.16.2.1 IWebDisplayReference.GetWebDisplayComponent Method

Pascal

```
function GetWebDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetWebDisplayComponent();
```

Description

This is GetWebDisplayComponent, a member of class IWebDisplayReference.

52.2.16.3 IWebDisplayReference Properties

The properties of the IWebDisplayReference class are listed here.

Public Properties

	Name	Description
	WebDisplayComponent (see page 5672)	

52.2.16.3.1 IWebDisplayReference.WebDisplayComponent Property

Pascal

```
property WebDisplayComponent: TComponent;
```

C++

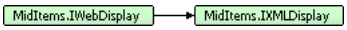
```
__property TComponent * WebDisplayComponent;
```

See Also

GetWebDisplayComponent (see page 5672)

52.2.17 IXMLDisplay Interface

Class Hierarchy



Pascal

```
IXMLDisplay = interface(IWebDisplay);
```

C++

```
__interface IXMLDisplay : public IWebDisplay;
```

File

MidItems

Description

This is class MidItems.IXMLDisplay.

52.2.17.1 IXMLDisplay Members

The following tables list the members exposed by IXMLDisplay.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 5674)	
R	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

52.2.17.2 IXMLDisplay Methods

The methods of the IXMLDisplay class are listed here.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 5673)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetXMLDisplayName (see page 5674)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 5674)	This is GetXMLRowSetName, a member of class IXMLDisplay.

52.2.17.2.1 IXMLDisplay.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class IXMLDisplay.

52.2.17.2.2 IXMLDisplay.GetXMLDisplayName Method

Pascal

```
function GetXMLDisplayName: string;
```

C++

```
__fastcall UnicodeString GetXMLDisplayName();
```

Description

This is GetXMLDisplayName, a member of class IXMLDisplay.

52.2.17.2.3 IXMLDisplay.GetXMLRowSetName Method

Pascal

```
function GetXMLRowSetName: string;
```

C++

```
__fastcall UnicodeString GetXMLRowSetName();
```

Description

This is GetXMLRowSetName, a member of class IXMLDisplay.

52.2.17.3 IXMLDisplay Properties

The properties of the IXMLDisplay class are listed here.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 5674)	
R	XMLDisplayName (see page 5675)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 5675)	This is XMLRowSetName, a member of class IXMLDisplay.

52.2.17.3.1 IXMLDisplay.IsMultipleRecordView Property

Pascal

```
property IsMultipleRecordView: Boolean;
```

C++

```
__property Boolean IsMultipleRecordView;
```

Description

.

See Also

GetIsMultipleRecordView (see page 5670)

52.2.17.3.2 IXMLDisplay.XMLDisplayName Property

Pascal

```
property XMLDisplayName: string;
```

C++

```
__property UnicodeString XMLDisplayName;
```

Description

This is XMLDisplayName, a member of class IXMLDisplay.

52.2.17.3.3 IXMLDisplay.XMLRowSetName Property

Pascal

```
property XMLRowSetName: string;
```

C++

```
__property UnicodeString XMLRowSetName;
```

Description

This is XMLRowSetName, a member of class IXMLDisplay.

52.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddBoolAttrib (see page 5676)	This is function MidItems.AddBoolAttrib.
	AddCustomAttrib (see page 5676)	This is function MidItems.AddCustomAttrib.
	AddIntAttrib (see page 5677)	This is function MidItems.AddIntAttrib.
	AddQuotedAttrib (see page 5677)	This is function MidItems.AddQuotedAttrib.
	AddStringAttrib (see page 5677)	This is function MidItems.AddStringAttrib.
	CanAddClassHelper (see page 5678)	This is function MidItems.CanAddClassHelper.
	CreateDefaultButtonClasses (see page 5678)	This is function MidItems.CreateDefaultButtonClasses.
	DeclareSubmitForm (see page 5678)	This is function MidItems.DeclareSubmitForm.
	FindDispatcher (see page 5679)	This is function MidItems.FindDispatcher.
	FindProducer (see page 5679)	This is function MidItems.FindProducer.
	FormatColumnData (see page 5679)	This is function MidItems.FormatColumnData.
	FormatColumnHeading (see page 5680)	This is function MidItems.FormatColumnHeading.
	GetItemValuesFromDataSet (see page 5680)	This is function MidItems.GetItemValuesFromDataSet.

52.3.1 MidItems.AddBoolAttrib Function

Pascal

```
procedure AddBoolAttrib(var Attribs: string; const Attrib: string; Value: Boolean);
```

C++

```
AddBoolAttrib(UnicodeString Attribs, const UnicodeString Attrib, Boolean Value);
```

File

MidItems

Description

This is function MidItems.AddBoolAttrib.

52.3.2 MidItems.AddCustomAttrib Function

Pascal

```
procedure AddCustomAttrib(var Attribs: string; Value: string);
```

C++

```
AddCustomAttrib(UnicodeString Attribs, UnicodeString Value);
```

File

MidItems

Description

This is function MidItems.AddCustomAttrib.

52.3.3 MidItems.AddIntAttrib Function

Pascal

```
procedure AddIntAttrib(var Attribs: string; const Attrib: string; Value: Integer);
```

C++

```
AddIntAttrib(UnicodeString Attribs, const UnicodeString Attrib, int Value);
```

File

MidItems

Description

This is function MidItems.AddIntAttrib.

52.3.4 MidItems.AddQuotedAttrib Function

Pascal

```
procedure AddQuotedAttrib(var Attribs: string; const Attrib: string; const Value: string);
```

C++

```
AddQuotedAttrib(UnicodeString Attribs, const UnicodeString Attrib, const UnicodeString Value);
```

File

MidItems

Description

This is function MidItems.AddQuotedAttrib.

52.3.5 MidItems.AddStringAttrib Function

Pascal

```
procedure AddStringAttrib(var Attribs: string; const Attrib: string; const Value: string);
```

C++

```
AddStringAttrib(UnicodeString Attribs, const UnicodeString Attrib, const UnicodeString Value);
```

File

MidItems

Description

This is function MidItems.AddStringAttrib.

52.3.6 MidItems.CanAddClassHelper Function

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass AClass);
```

File

MidItems

Description

This is function MidItems.CanAddClassHelper.

52.3.7 MidItems.CreateDefaultButtonClasses Function

Pascal

```
procedure CreateDefaultButtonClasses(const Classes: array of TWebButtonClass; Container:
TWebComponentList);
```

C++

```
CreateDefaultButtonClasses(const array of TWebButtonClass Classes, TWebComponentList
Container);
```

File

MidItems

Description

This is function MidItems.CreateDefaultButtonClasses.

52.3.8 MidItems.DeclareSubmitForm Function

Pascal

```
procedure DeclareSubmitForm(Component: TComponent; XMLBroker: TXMLBroker; AddIntf:
IAddScriptElements);
```

C++

```
DeclareSubmitForm(TComponent * Component, TXMLBroker XMLBroker, IAddScriptElements AddIntf);
```

File

MidItems

Description

This is function MidItems.DeclareSubmitForm.

52.3.9 MidItems.FindDispatcher Function

Pascal

```
function FindDispatcher(Component: TComponent): IWebDispatcherAccess;
```

C++

```
IWebDispatcherAccess FindDispatcher(TComponent * Component);
```

File

MidItems

Description

This is function MidItems.FindDispatcher.

52.3.10 MidItems.FindProducer Function

Pascal

```
function FindProducer(Component: TComponent): TCustomContentProducer;
```

C++

```
TCustomContentProducer FindProducer(TComponent * Component);
```

File

MidItems

Description

This is function MidItems.FindProducer.

52.3.11 MidItems.FormatColumnData Function

Pascal

```
function FormatColumnData(Field: TWebDataDisplay; Content: string): string;
```

C++

```
UnicodeString FormatColumnData(TWebDataDisplay Field, UnicodeString Content);
```

File

MidItems

Description

This is function MidItems.FormatColumnData.

52.3.12 MidItems.FormatColumnHeading Function

Pascal

```
function FormatColumnHeading(Field: TWebDataDisplay): string;
```

C++

```
UnicodeString FormatColumnHeading(TWebDataDisplay Field);
```

File

MidItems

Description

This is function MidItems.FormatColumnHeading.

52.3.13 MidItems.GetItemValuesFromDataSet Function

Pascal

```
function GetItemValuesFromDataSet(DataSet: TDataSet; const ItemsField: string; const
ValuesField: string; var ItemsStrings: TStrings; var ValuesStrings: TStrings): Boolean;
```

C++

```
Boolean GetItemValuesFromDataSet(TDataSet DataSet, const UnicodeString ItemsField, const
UnicodeString ValuesField, TStrings ItemsStrings, TStrings ValuesStrings);
```

File

MidItems

Description

This is function MidItems.GetItemValuesFromDataSet.

52.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCaptionPosition (see page 5681)	TCaptionPosition indicates the relative position of a label to the HTML control it labels.
	TFormMethod (see page 5682)	TFormMethod represents the method attribute on the <form> tag of an HTML form.
	TLayoutState (see page 5682)	This is record MidItems.TLayoutState.
	TTextAreaWrap (see page 5683)	TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

52.4.1 MidItems.TCaptionPosition Enumeration

TCaptionPosition indicates the relative position of a label to the HTML control it labels.

Pascal

```
TCaptionPosition = (
  capLeft,
  capRight,
  capAbove,
  capBelow
);
```

C++

```
enum TCaptionPosition {
  capLeft,
  capRight,
  capAbove,
  capBelow
};
```

File

MidItems

Description

TCaptionPosition can have one of the following values.

Value	Meaning
capLeft	The caption appears to the left of the HTML control.
capRight	The caption appears to the right of the HTML control.
capAbove	The caption appears above the HTML control.
capBelow	The caption appears below the HTML control.

52.4.2 MidItems.TFormMethod Enumeration

TFormMethod represents the method attribute on the <form> tag of an HTML form.

Pascal

```
TFormMethod = (
  fmPost,
  fmGet
);
```

C++

```
enum TFormMethod {
  fmPost,
  fmGet
};
```

File

MidItems

Description

HTML forms include a <form> tag that distinguishes them from other types of HTML pages. This<form> tag includes a method attribute, which determines the method header of the HTTP request message that the form sends to its server. The method type corresponds to the MethodType property of the HTTP request message.

Components that generate HTML forms use TFormMethod to specify the method attribute of the <form> tag they generate, and hence the method header of the messages that they send to the server.

The following table describes the possible values:

Value	Corresponding MethodType	Meaning
fmPost	mtPost	The value of the method attribute is 'post'. The form submits messages that post displayed values to the server application. Named value pairs are passed as content fields of the HTTP message.
fmGet	mtGet	The value of the method attribute is 'get'. The form submits messages that request values it can display. Named value pairs are supplied as query fields in the URL.

See Also

TWebRequest

52.4.3 MidItems.TLayoutState Enumeration

Pascal

```
TLayoutState = (
  lsUnknown,
  lsFields,
  lsButtons
);
```

C++

```
enum TLayoutState {
```

```
    lsUnknown,
    lsFields,
    lsButtons
  };
```

File

MidItems

Description

This is record MidItems.TLayoutState.

52.4.4 MidItems.TTextAreaWrap Enumeration

TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

Pascal

```
  TTextAreaWrap = (
    wrOff,
    wrPhysical,
    wrVirtual
  );
```

C++

```
  enum TTextAreaWrap {
    wrOff,
    wrPhysical,
    wrVirtual
  };
```

File

MidItems

Description

TTextAreaWrap can have one of the following values.

Value	Meaning
wrOff	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

52.5 Types

The following table lists types in this documentation.

Types

Name	Description
IMidasWebDataSet (see page 5684)	This is type MidItems.IMidasWebDataSet.
TWebButtonClass (see page 5684)	This is type MidItems.TWebButtonClass.
TWebDataDisplayClass (see page 5685)	This is type MidItems.TWebDataDisplayClass.
TWebDataInputClass (see page 5685)	This is type MidItems.TWebDataInputClass.
TWebStatusClass (see page 5685)	This is type MidItems.TWebStatusClass.

52.5.1 MidItems.IMidasWebDataSet Type

Pascal

```
IMidasWebDataSet = IInetXWebDataSet;
```

C++

```
IInetXWebDataSet IMidasWebDataSet;
```

File

MidItems

Description

This is type MidItems.IMidasWebDataSet.

52.5.2 MidItems.TWebButtonClass Type

Pascal

```
TWebButtonClass = class of TWebButton;
```

C++

```
TWebButton TWebButtonClass;
```

File

MidItems

Description

This is type MidItems.TWebButtonClass.

52.5.3 MidItems.TWebDataDisplayClass Type

Pascal

```
TWebDataDisplayClass = class of TWebDataDisplay;
```

C++

```
TWebDataDisplay TWebDataDisplayClass;
```

File

MidItems

Description

This is type MidItems.TWebDataDisplayClass.

52.5.4 MidItems.TWebDataInputClass Type

Pascal

```
TWebDataInputClass = class of TWebDataInput;
```

C++

```
TWebDataInput TWebDataInputClass;
```

File

MidItems

Description

This is type MidItems.TWebDataInputClass.

52.5.5 MidItems.TWebStatusClass Type

Pascal

```
TWebStatusClass = class of TWebStatus;
```

C++

```
TWebStatus TWebStatusClass;
```

File

MidItems

Description

This is type MidItems.TWebStatusClass.

52.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
FormMethodNames (see page 5686)	This is constant MidItems.FormMethodNames.
HTMLAlignValues (see page 5687)	This is constant MidItems.HTMLAlignValues.
HTMLVAlignValues (see page 5687)	This is constant MidItems.HTMLVAlignValues.
sActionAttr (see page 5687)	This is constant MidItems.sActionAttr.
sAlignAttr (see page 5688)	This is constant MidItems.sAlignAttr.
sBgColorAttr (see page 5688)	This is constant MidItems.sBgColorAttr.
sBorderAttr (see page 5688)	This is constant MidItems.sBorderAttr.
sCellPaddingAttr (see page 5689)	This is constant MidItems.sCellPaddingAttr.
sCellSpacingAttr (see page 5689)	This is constant MidItems.sCellSpacingAttr.
sCheckedAttr (see page 5689)	This is constant MidItems.sCheckedAttr.
sClassAttr (see page 5690)	This is constant MidItems.sClassAttr.
sColsAttr (see page 5690)	This is constant MidItems.sColsAttr.
sColSpanAttr (see page 5690)	This is constant MidItems.sColSpanAttr.
sEncTypeAttr (see page 5691)	This is constant MidItems.sEncTypeAttr.
sImageHeightAttr (see page 5691)	This is constant MidItems.sImageHeightAttr.
sImageWidthAttr (see page 5691)	This is constant MidItems.sImageWidthAttr.
sMaxLengthAttr (see page 5692)	This is constant MidItems.sMaxLengthAttr.
sMethodAttr (see page 5692)	This is constant MidItems.sMethodAttr.
sMultipleAttr (see page 5692)	This is constant MidItems.sMultipleAttr.
sNameAttr (see page 5693)	This is constant MidItems.sNameAttr.
sOnClickAttr (see page 5693)	This is constant MidItems.sOnClickAttr.
sReadOnlyAttr (see page 5693)	This is constant MidItems.sReadOnlyAttr.
sRowsAttr (see page 5694)	This is constant MidItems.sRowsAttr.
sSelectedAttr (see page 5694)	This is constant MidItems.sSelectedAttr.
sSizeAttr (see page 5694)	This is constant MidItems.sSizeAttr.
sStyleAttr (see page 5695)	This is constant MidItems.sStyleAttr.
sTabIndexAttr (see page 5695)	This is constant MidItems.sTabIndexAttr.
sTypeAttr (see page 5695)	This is constant MidItems.sTypeAttr.
sVAlignAttr (see page 5696)	This is constant MidItems.sVAlignAttr.
sValueAttr (see page 5696)	This is constant MidItems.sValueAttr.
sWidthAttr (see page 5696)	This is constant MidItems.sWidthAttr.
sWrapAttr (see page 5697)	This is constant MidItems.sWrapAttr.
TextAreaWrap (see page 5697)	This is constant MidItems.TextAreaWrap.

52.6.1 MidItems.FormMethodNames Constant

Pascal

```
FormMethodNames: array[TFormMethod] of string = ('POST', 'GET');
```

C++

```
array[TFormMethod] of string FormMethodNames = ('POST', 'GET');
```

File

MidItems

Description

This is constant MidItems.FormMethodNames.

52.6.2 MidItems.HTMLAlignValues Constant

Pascal

```
HTMLAlignValues: array[THTMLAlign] of string = ('', 'left', 'right', 'center');
```

C++

```
array[THTMLAlign] of string HTMLAlignValues = ('', 'left', 'right', 'center');
```

File

MidItems

Description

This is constant MidItems.HTMLAlignValues.

52.6.3 MidItems.HTMLVAlignValues Constant

Pascal

```
HTMLVAlignValues: array[THTMLVAlign] of string = ('', 'top', 'middle', 'bottom', 'baseline');
```

C++

```
array[THTMLVAlign] of string HTMLVAlignValues = ('', 'top', 'middle', 'bottom', 'baseline');
```

File

MidItems

Description

This is constant MidItems.HTMLVAlignValues.

52.6.4 MidItems.sActionAttr Constant

Pascal

```
sActionAttr = 'action';
```

C++

```
sActionAttr = 'action';
```

File

MidItems

Description

This is constant MidItems.sActionAttr.

52.6.5 MidItems.sAlignAttr Constant

Pascal

```
sAlignAttr = 'align';
```

C++

```
sAlignAttr = 'align';
```

File

MidItems

Description

This is constant MidItems.sAlignAttr.

52.6.6 MidItems.sBgColorAttr Constant

Pascal

```
sBgColorAttr = 'bgcolor';
```

C++

```
sBgColorAttr = 'bgcolor';
```

File

MidItems

Description

This is constant MidItems.sBgColorAttr.

52.6.7 MidItems.sBorderAttr Constant

Pascal

```
sBorderAttr = 'border';
```

C++

```
sBorderAttr = 'border';
```

File

MidItems

Description

This is constant MidItems.sBorderAttr.

52.6.8 MidItems.sCellPaddingAttr Constant

Pascal

```
sCellPaddingAttr = 'cellpadding';
```

C++

```
sCellPaddingAttr = 'cellpadding';
```

File

MidItems

Description

This is constant MidItems.sCellPaddingAttr.

52.6.9 MidItems.sCellSpacingAttr Constant

Pascal

```
sCellSpacingAttr = 'cellspacing';
```

C++

```
sCellSpacingAttr = 'cellspacing';
```

File

MidItems

Description

This is constant MidItems.sCellSpacingAttr.

52.6.10 MidItems.sCheckedAttr Constant

Pascal

```
sCheckedAttr = 'checked';
```

C++

```
sCheckedAttr = 'checked';
```

File

MidItems

Description

This is constant MidItems.sCheckedAttr.

52.6.11 MidItems.sClassAttr Constant

Pascal

```
sClassAttr = 'class';
```

C++

```
sClassAttr = 'class';
```

File

MidItems

Description

This is constant MidItems.sClassAttr.

52.6.12 MidItems.sColsAttr Constant

Pascal

```
sColsAttr = 'cols';
```

C++

```
sColsAttr = 'cols';
```

File

MidItems

Description

This is constant MidItems.sColsAttr.

52.6.13 MidItems.sColSpanAttr Constant

Pascal

```
sColSpanAttr = 'colspan';
```

C++

```
sColSpanAttr = 'colspan';
```

File

MidItems

Description

This is constant MidItems.sColSpanAttr.

52.6.14 MidItems.sEncTypeAttr Constant

Pascal

```
sEncTypeAttr = 'enctype';
```

C++

```
sEncTypeAttr = 'enctype';
```

File

MidItems

Description

This is constant MidItems.sEncTypeAttr.

52.6.15 MidItems.sImageHeightAttr Constant

Pascal

```
sImageHeightAttr = 'imageheight';
```

C++

```
sImageHeightAttr = 'imageheight';
```

File

MidItems

Description

This is constant MidItems.sImageHeightAttr.

52.6.16 MidItems.sImageWidthAttr Constant

Pascal

```
sImageWidthAttr = 'imagewidth';
```

C++

```
sImageWidthAttr = 'imagewidth';
```

File

MidItems

Description

This is constant MidItems.sImageWidthAttr.

52.6.17 MidItems.sMaxLengthAttr Constant

Pascal

```
sMaxLengthAttr = 'maxlength';
```

C++

```
sMaxLengthAttr = 'maxlength';
```

File

MidItems

Description

This is constant MidItems.sMaxLengthAttr.

52.6.18 MidItems.sMethodAttr Constant

Pascal

```
sMethodAttr = 'method';
```

C++

```
sMethodAttr = 'method';
```

File

MidItems

Description

This is constant MidItems.sMethodAttr.

52.6.19 MidItems.sMultipleAttr Constant

Pascal

```
sMultipleAttr = 'multiple';
```

C++

```
sMultipleAttr = 'multiple';
```

File

MidItems

Description

This is constant MidItems.sMultipleAttr.

52.6.20 MidItems.sNameAttr Constant

Pascal

```
sNameAttr = 'name';
```

C++

```
sNameAttr = 'name';
```

File

MidItems

Description

This is constant MidItems.sNameAttr.

52.6.21 MidItems.sOnClickAttr Constant

Pascal

```
sOnClickAttr = 'onclick';
```

C++

```
sOnClickAttr = 'onclick';
```

File

MidItems

Description

This is constant MidItems.sOnClickAttr.

52.6.22 MidItems.sReadOnlyAttr Constant

Pascal

```
sReadOnlyAttr = 'readonly';
```

C++

```
sReadOnlyAttr = 'readonly';
```

File

MidItems

Description

This is constant MidItems.sReadOnlyAttr.

52.6.23 MidItems.sRowsAttr Constant

Pascal

```
sRowsAttr = 'rows';
```

C++

```
sRowsAttr = 'rows';
```

File

MidItems

Description

This is constant MidItems.sRowsAttr.

52.6.24 MidItems.sSelectedAttr Constant

Pascal

```
sSelectedAttr = 'selected';
```

C++

```
sSelectedAttr = 'selected';
```

File

MidItems

Description

This is constant MidItems.sSelectedAttr.

52.6.25 MidItems.sSizeAttr Constant

Pascal

```
sSizeAttr = 'size';
```

C++

```
sSizeAttr = 'size';
```

File

MidItems

Description

This is constant MidItems.sSizeAttr.

52.6.26 MidItems.sStyleAttr Constant

Pascal

```
sStyleAttr = 'style';
```

C++

```
sStyleAttr = 'style';
```

File

MidItems

Description

This is constant MidItems.sStyleAttr.

52.6.27 MidItems.sTabIndexAttr Constant

Pascal

```
sTabIndexAttr = 'tabindex';
```

C++

```
sTabIndexAttr = 'tabindex';
```

File

MidItems

Description

This is constant MidItems.sTabIndexAttr.

52.6.28 MidItems.sTypeAttr Constant

Pascal

```
sTypeAttr = 'type';
```

C++

```
sTypeAttr = 'type';
```

File

MidItems

Description

This is constant MidItems.sTypeAttr.

52.6.29 MidItems.sVAlignAttr Constant

Pascal

```
sVAlignAttr = 'valign';
```

C++

```
sVAlignAttr = 'valign';
```

File

MidItems

Description

This is constant MidItems.sVAlignAttr.

52.6.30 MidItems.sValueAttr Constant

Pascal

```
sValueAttr = 'value';
```

C++

```
sValueAttr = 'value';
```

File

MidItems

Description

This is constant MidItems.sValueAttr.

52.6.31 MidItems.sWidthAttr Constant

Pascal

```
sWidthAttr = 'width';
```

C++

```
sWidthAttr = 'width';
```

File

MidItems

Description

This is constant MidItems.sWidthAttr.

52.6.32 MidItems.sWrapAttr Constant

Pascal

```
sWrapAttr = 'wrap';
```

C++

```
sWrapAttr = 'wrap';
```

File

MidItems

Description

This is constant MidItems.sWrapAttr.

52.6.33 MidItems.TextAreaWrap Constant

Pascal

```
TextAreaWrap: array[TTextAreaWrap] of string = ('OFF', 'PHYSICAL', 'VIRTUAL');
```

C++

```
array[TTextAreaWrap] of string TextAreaWrap = ('OFF', 'PHYSICAL', 'VIRTUAL');
```

File

MidItems

Description

This is constant MidItems.TextAreaWrap.

53 MidProd Namespace

This is namespace MidProd.

Topics

Name	Description
Classes (see page 5701)	The following table lists classes in this documentation.
Functions (see page 5741)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5742)	The following table lists structs, records, enums in this documentation.
Types (see page 5743)	The following table lists types in this documentation.
Constants (see page 5745)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseInetXPageProducer (see page 5701)	TBaseInetXPageProducer is an base class for InternetExpress page producer classes.
	TCustomInetXPageProducer (see page 5720)	TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.
	TInetXPageElements (see page 5727)	This is class MidProd.TInetXPageElements.
	TInetXPageProducer (see page 5730)	TInetXPageProducer assembles an HTML document that represents live database information from an application server.
	TMidasPageProducer (see page 5740)	TMidasPageProducer assembles an HTML document that represents live database information from an application server.

Constants

Name	Description
sXMLBroker (see page 5745)	This is constant MidProd.sXMLBroker.
sXMLReadyVar (see page 5745)	This is constant MidProd.sXMLReadyVar.

Functions

	Name	Description
	AddStdIncludes (see page 5741)	This is function MidProd.AddStdIncludes.
	PathInfoToRelativePath (see page 5741)	This is function MidProd.PathInfoToRelativePath.

Structs, Records, Enums

	Name	Description
	TStdInclude (see page 5742)	This is record MidProd.TStdInclude.

Types

Name	Description
TBaseMidasPageProducer (see page 5743)	This is type MidProd.TBaseMidasPageProducer.
TCustomMidasPageProducer (see page 5743)	This is type MidProd.TCustomMidasPageProducer.
TMidasPageElements (see page 5744)	This is type MidProd.TMidasPageElements.
TStdIncludes (see page 5744)	This is type MidProd.TStdIncludes.
TXMLDataEvent (see page 5744)	This is type MidProd.TXMLDataEvent.

53.1 Classes

The following table lists classes in this documentation.

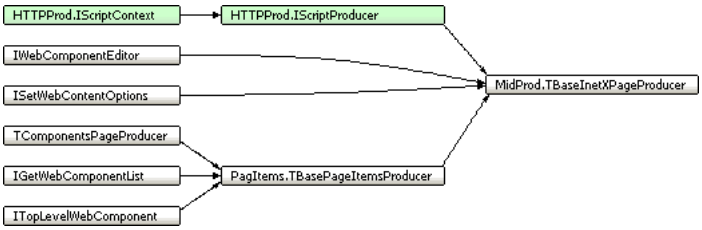
Classes

	Name	Description
	TBaseInetXPageProducer (see page 5701)	TBaseInetXPageProducer is an base class for InternetExpress page producer classes.
	TCustomInetXPageProducer (see page 5720)	TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.
	TInetXPageElements (see page 5727)	This is class MidProd.TInetXPageElements.
	TInetXPageProducer (see page 5730)	TInetXPageProducer assembles an HTML document that represents live database information from an application server.
	TMidasPageProducer (see page 5740)	TMidasPageProducer assembles an HTML document that represents live database information from an application server.

53.1.1 TBaseInetXPageProducer Class

TBaseInetXPageProducer is an base class for InternetExpress page producer classes.

Class Hierarchy



Pascal

```
TBaseInetXPageProducer = class(TBasePageItemsProducer, IWebComponentEditor, IScriptProducer,
ISetWebContentOptions);
```

C++

```
class TBaseInetXPageProducer : public TBasePageItemsProducer, public IWebComponentEditor,
public IScriptProducer, public ISetWebContentOptions;
```

File

MidProd

Description

Do not create instances of TBaseInetXPageProducer. Instead, use one of its descendant classes, such as TInetXPageProducer (see page 5730).

TBaseInetXPageProducer introduces the logic for working with an XML broker to obtain database information and for translating the special HTML-transparent tags that are used in InternetExpress applications. Unlike its immediate descendant (TCustomInetXPageProducer (see page 5720)) however, TBaseInetXPageProducer does not initialize the HTMLDoc property to the default template for an InternetExpress application, although it dynamically generates the InternetExpress-specific tags.

You can use TBaseInetXPageProducer as a base class when writing a page producer that does not use the default template, but

which translates InternetExpress tags into HTML and javascript.

See Also

TInetXPageProducer (see page 5730)

TXMLBroker

53.1.1.1 TBaseInetXPageProducer Members

The following tables list the members exposed by TBaseInetXPageProducer.

Protected Methods

	Name	Description
	AddIncludes (see page 5705)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 5705)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	CanAddClass (see page 5705)	Indicates whether a specified class can be used as a child Web item.
	CreatePageElements (see page 5706)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 5707)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 5707)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 5707)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 5707)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 5708)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 5708)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 5708)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 5708)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 5708)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DoAfterGetContent (see page 5709)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 5709)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 5709)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 5709)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 5710)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	GeneratePageElements (see page 5711)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetContentOptions (see page 5711)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetDefaultTemplate (see page 5711)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetScriptManager (see page 5712)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetUseXMLIslands (see page 5712)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 5713)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 5713)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	ImplCanAddClass (see page 5713)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetWebContentOptions (see page 5714)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.

Protected Properties

	Name	Description
	PageElements (see page 5716)	This is PageElements, a member of class TBaseInetXPageProducer.
	ScriptManager (see page 5716)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Events

	Name	Description
	OnAfterGetContent (see page 5718)	Occurs after the page producer generates its HTML document.

	OnAfterGetXMLData (see page 5718)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 5719)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 5719)	Occurs before the page producer fetches an XML data packet from an XML broker.

Public Methods

	Name	Description
V	ContentFromStream (see page 5706)	Returns the content produced in response to input read from a stream.
V	Create (see page 5703)	Creates an instance of a TBaseInetXPageProducer (see page 5701).
V	Destroy (see page 5704)	Destroys the instance of the TBasePageItemsProducer.
	Evaluate (see page 3509)	This is Evaluate, a member of class IScriptProducer.
V	FindXMLBroker (see page 5710)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
V	FindXMLBrokerName (see page 5710)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	GetContent (see page 3509)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 3509)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 3510)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 3510)	This is GetHTMLBlockCount, a member of class IScriptProducer.
V	GetRequiredTags (see page 5711)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.
V	GetXMLData (see page 5712)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 3510)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 3510)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 3510)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 3511)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 3511)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 5713)	Assigns a value to the IncludePathURL (see page 5715) property.
	SetStyles (see page 5714)	Assigns a value to the Styles (see page 5716) property.
	SetStylesFile (see page 5714)	Assigns a value to the StylesFile (see page 5717) property.
	Write (see page 3511)	This is the overview for the Write method overload.
	WriteItem (see page 3512)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
	Content (see page 3512)	This is Content, a member of class IScriptProducer.
	EnableXMLIslands (see page 5715)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
R	Errors (see page 3512)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 3513)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 3513)	This is HTMLBlocks, a member of class IScriptProducer.
	IncludePathURL (see page 5715)	Indicates the URL where the javascript libraries are installed.
	Styles (see page 5716)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 5717)	Specifies a file containing the individual styles for the page producer's style sheet.
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.
R	WebPageItems (see page 6393)	Maintains a list of components that generate HTML content for the producer.

53.1.1.2 TBaseInetXPageProducer.Create Constructor

Creates an instance of a TBaseInetXPageProducer (see page 5701).

Pascal

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

C++

```
virtual __fastcall TBaseInetXPageProducer(TComponent * AComponent);
```

Description

Call Create to instantiate a TBaseInetXPageProducer (see page 5701) component at runtime if it was not created at design time.

AOwner is the component, typically the Web module, that becomes the value of this page producer's Owner property. The Owner is responsible for freeing the TBaseInetXPageProducer (see page 5701) instance.

See Also

Owner

53.1.1.3 TBaseInetXPageProducer.Destroy Destructor

Destroys the instance of the TBasePageItemsProducer.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseInetXPageProducer();
```

Description

Do not call Destroy directly. Instead call Free, which checks that the TBasePageItemsProducer reference is not nil before calling Destroy.

See Also

Create (see page 6391)

System::TObject::Free

53.1.1.4 TBaseInetXPageProducer Methods

The methods of the TBaseInetXPageProducer class are listed here.

Protected Methods

	Name	Description
	AddIncludes (see page 5705)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 5705)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	CanAddClass (see page 5705)	Indicates whether a specified class can be used as a child Web item.
	CreatePageElements (see page 5706)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 5707)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 5707)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 5707)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 5707)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 5708)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 5708)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 5708)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 5708)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 5708)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DoAfterGetContent (see page 5709)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.

	DoAfterGetXMLData (see page 5709)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 5709)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 5709)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
V	DoTagEvent (see page 5710)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
V	GeneratePageElements (see page 5711)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
V	GetContentOptions (see page 5711)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
V	GetDefaultTemplate (see page 5711)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetScriptManager (see page 5712)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
V	GetUseXMLIslands (see page 5712)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 5713)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 5713)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
V	ImplCanAddClass (see page 5713)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	SetWebContentOptions (see page 5714)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.

Public Methods

	Name	Description
V	ContentFromStream (see page 5706)	Returns the content produced in response to input read from a stream.
V	FindXMLBroker (see page 5710)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
V	FindXMLBrokerName (see page 5710)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
V	GetRequiredTags (see page 5711)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
V	GetXMLData (see page 5712)	Obtains an XML data packet from a specified XML broker.
	SetIncludePathURL (see page 5713)	Assigns a value to the IncludePathURL (see page 5715) property.
	SetStyles (see page 5714)	Assigns a value to the Styles (see page 5716) property.
	SetStylesFile (see page 5714)	Assigns a value to the StylesFile (see page 5717) property.

53.1.1.4.1 TBaseInetXPageProducer.AddIncludes Method

Pascal

```
procedure AddIncludes; virtual;
```

C++

```
virtual __fastcall AddIncludes();
```

Description

This is AddIncludes, a member of class TBaseInetXPageProducer.

53.1.1.4.2 TBaseInetXPageProducer.AddScriptComponents Method

Pascal

```
procedure AddScriptComponents; virtual;
```

C++

```
virtual __fastcall AddScriptComponents();
```

Description

This is AddScriptComponents, a member of class TBaseInetXPageProducer.

53.1.1.4.3 TBaseInetXPageProducer.CanAddClass Method

Indicates whether a specified class can be used as a child Web item.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the Web item that implements this interface.

AParent is the Web item that will act as a parent to the added Web item.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web item can use content generated by a component of type AClass, false otherwise.

53.1.1.4.4 TBaseInetXPageProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; override;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream InStream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter. ContentFromStream

1. Generates an OnBeforeGetContent (see page 5719) event.
2. Translates the HTML template that is the value of Stream, using information produced by the items listed in the WebPageItems property.
3. Generates an OnAfterGetContent (see page 5718) event.
4. Returns the HTML produced in step 2.

See Also

Content (see page 3466)

OnBeforeGetContent (see page 5719)

WebPageItems (see page 6393)

OnAfterGetContent (see page 5718)

53.1.1.4.5 TBaseInetXPageProducer.CreatePageElements Method

Pascal

```
function CreatePageElements: TInetxPageElements; virtual;
```

C++

```
virtual __fastcall TInetxPageElements CreatePageElements();
```

Description

This is CreatePageElements, a member of class TBaseInetXPageProducer.

53.1.1.4.6 TBaseInetXPageProducer.CreateScriptManager Method

Pascal

```
function CreateScriptManager(Options: TWebContentOptions): TObject; virtual;
```

C++

```
virtual __fastcall TObject * CreateScriptManager(TWebContentOptions Options);
```

Description

This is CreateScriptManager, a member of class TBaseInetXPageProducer.

53.1.1.4.7 TBaseInetXPageProducer.DeclareFunctions Method

Pascal

```
function DeclareFunctions: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareFunctions();
```

Description

This is DeclareFunctions, a member of class TBaseInetXPageProducer.

53.1.1.4.8 TBaseInetXPageProducer.DeclareIncludes Method

Pascal

```
function DeclareIncludes: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareIncludes();
```

Description

This is DeclareIncludes, a member of class TBaseInetXPageProducer.

53.1.1.4.9 TBaseInetXPageProducer.DeclareRowSets Method

Pascal

```
function DeclareRowSets: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareRowSets();
```

Description

This is DeclareRowSets, a member of class TBaseInetXPageProducer.

53.1.1.4.10 TBaseInetXPageProducer.DeclareScript Method

Pascal

```
function DeclareScript(XMLIslands: Boolean): string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareScript(Boolean XMLIslands);
```

Description

This is DeclareScript, a member of class TBaseInetXPageProducer.

53.1.1.4.11 TBaseInetXPageProducer.DeclareStyles Method

Pascal

```
function DeclareStyles: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareStyles();
```

Description

This is DeclareStyles, a member of class TBaseInetXPageProducer.

53.1.1.4.12 TBaseInetXPageProducer.DeclareWarnings Method

Pascal

```
function DeclareWarnings(Warnings: TStrings): string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareWarnings(TStrings Warnings);
```

Description

This is DeclareWarnings, a member of class TBaseInetXPageProducer.

53.1.1.4.13 TBaseInetXPageProducer.DeclareXMLDocuments Method

Pascal

```
function DeclareXMLDocuments: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareXMLDocuments();
```

Description

This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.

53.1.1.4.14 TBaseInetXPageProducer.DeclareXMLIslands Method

Pascal

```
function DeclareXMLIslands: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareXMLIslands();
```

Description

This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.

53.1.1.4.15 TBaseInetXPageProducer.DoAfterGetContent Method

Pascal

```
procedure DoAfterGetContent;
```

C++

```
__fastcall DoAfterGetContent();
```

Description

This is DoAfterGetContent, a member of class TBaseInetXPageProducer.

53.1.1.4.16 TBaseInetXPageProducer.DoAfterGetXMLData Method

Pascal

```
procedure DoAfterGetXMLData(XMLBroker: TXMLBroker; var OwnerData: OleVariant);
```

C++

```
__fastcall DoAfterGetXMLData(TXMLBroker XMLBroker, OleVariant OwnerData);
```

Description

This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.

53.1.1.4.17 TBaseInetXPageProducer.DoBeforeGetContent Method

Pascal

```
procedure DoBeforeGetContent;
```

C++

```
__fastcall DoBeforeGetContent();
```

Description

This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.

53.1.1.4.18 TBaseInetXPageProducer.DoBeforeGetXMLData Method

Pascal

```
procedure DoBeforeGetXMLData(XMLBroker: TXMLBroker; var OwnerData: OleVariant);
```

C++

```
__fastcall DoBeforeGetXMLData(TXMLBroker XMLBroker, OleVariant OwnerData);
```

Description

This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.

53.1.1.4.19 TBaseInetXPageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceText);
```

Description

This is DoTagEvent, a member of class TBaseInetXPageProducer.

53.1.1.4.20 TBaseInetXPageProducer.FindXMLBroker Method

Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.

Pascal

```
function FindXMLBroker(TagParams: TStrings): TXMLBroker; virtual;
```

C++

```
virtual __fastcall TXMLBroker FindXMLBroker(TStrings TagParams);
```

Description

The page producer translates HTML-transparent tags of the form

```
<#DATAPACKET XMLBroker=BrokerName>
```

into the XML returned by the XML broker. To do so, it calls FindXMLBroker to locate the XML broker specified by the XMLBroker parameter.

TagParams is a set of Name=Value pairs. This comes from the parameter portion of the DATAPACKET tag.

FindXMLBroker returns the XML broker with the name specified by the XMLBroker parameter. If TagParams does not contain a string of the form XMLBroker=BrokerName, FindXMLBroker returns nil (Delphi) or NULL (C++).

See Also

FindXMLBrokerName (see page 5710)

GetXMLData (see page 5712)

53.1.1.4.21 TBaseInetXPageProducer.FindXMLBrokerName Method

Returns the name of an XML broker given the parameters in a DATAPACKET tag..

Pascal

```
function FindXMLBrokerName(TagParams: TStrings): string; virtual;
```

C++

```
virtual __fastcall UnicodeString FindXMLBrokerName(TStrings TagParams);
```

Description

The page producer translates HTML-transparent tags of the form

```
<#DATAPACKET XMLBroker=BrokerName>
```

into the XML returned by the XML broker. To do so, it calls FindXMLBrokerName to parse the parameters for the value of the

XMLBroker parameter.

TagParams is a set of Name=Value pairs. This comes from the parameter portion of the DATAPACKET tag.

FindXMLBrokerName returns the value string from the XMLBroker parameter. If TagParams does not contain a string of the form XMLBroker=BrokerName, FindXMLBrokerName returns an empty string.

See Also

FindXMLBroker (see page 5710)

GetXMLData (see page 5712)

53.1.1.4.22 TBaseInetXPageProducer.GeneratePageElements Method

Pascal

```
procedure GeneratePageElements; virtual;
```

C++

```
virtual __fastcall GeneratePageElements();
```

Description

This is GeneratePageElements, a member of class TBaseInetXPageProducer.

53.1.1.4.23 TBaseInetXPageProducer.GetContentOptions Method

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; override;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

This is GetContentOptions, a member of class TBaseInetXPageProducer.

53.1.1.4.24 TBaseInetXPageProducer.GetDefaultTemplate Method

Pascal

```
function GetDefaultTemplate: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultTemplate();
```

Description

This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.

53.1.1.4.25 TBaseInetXPageProducer.GetRequiredTags Method

Returns the default set of HTML-transparent tags for an InternetExpress page producer.

Pascal

```
class function GetRequiredTags: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetRequiredTags();
```

Description

GetRequiredTags dynamically generates the InternetExpress-specific tags that form the basis of an InternetExpress page producer's default template.

GetRequiredTags returns the string

```
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
```

See Also

HTMLDoc (see page 5726)

Content (see page 5723)

53.1.1.4.26 TBaseInetXPageProducer.GetScriptManager Method

Pascal

```
function GetScriptManager: IScriptManager;
```

C++

```
__fastcall IScriptManager GetScriptManager();
```

Description

This is GetScriptManager, a member of class TBaseInetXPageProducer.

53.1.1.4.27 TBaseInetXPageProducer.GetUseXMLIslands Method

Pascal

```
function GetUseXMLIslands: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetUseXMLIslands();
```

Description

This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.

53.1.1.4.28 TBaseInetXPageProducer.GetXMLData Method

Obtains an XML data packet from a specified XML broker.

Pascal

```
function GetXMLData(XMLBroker: TXMLBroker; XMLOptions: TXMLOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetXMLData(TXMLBroker XMLBroker, TXMLOptions XMLOptions);
```

Description

GetXMLData first generates an OnBeforeGetXMLData (see page 5719) event. It then calls the RequestRecords method of the XML broker specified by XMLBroker, passing in the formatting options specified by XMLOptions. Finally, it generates an OnAfterGetXMLData (see page 5718) event.

GetXMLData returns the XML data packet that XMLBroker supplies.

See Also

OnAfterGetXMLData (see page 5718)

OnBeforeGetXMLData (see page 5719)

Content (see page 3466)

RequestRecords

53.1.1.4.29 TBaseInetXPageProducer.GetXMLDelta Method

Pascal

```
function GetXMLDelta(XMLBroker: TXMLBroker): string;
```

C++

```
__fastcall UnicodeString GetXMLDelta(TXMLBroker XMLBroker);
```

Description

This is GetXMLDelta, a member of class TBaseInetXPageProducer.

53.1.1.4.30 TBaseInetXPageProducer.GetXMLErrors Method

Pascal

```
function GetXMLErrors(XMLBroker: TXMLBroker): string;
```

C++

```
__fastcall UnicodeString GetXMLErrors(TXMLBroker XMLBroker);
```

Description

This is GetXMLErrors, a member of class TBaseInetXPageProducer.

53.1.1.4.31 TBaseInetXPageProducer.ImplCanAddClass Method

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is ImplCanAddClass, a member of class TBaseInetXPageProducer.

53.1.1.4.32 TBaseInetXPageProducer.SetIncludePathURL Method

Assigns a value to the IncludePathURL (see page 5715) property.

Pascal

```
procedure SetIncludePathURL(const Value: string);
```

C++

```
__fastcall SetIncludePathURL(const UnicodeString Value);
```

Description

SetIncludePathURL is the write implementation of the IncludePathURL (see page 5715) property.

See Also

IncludePathURL (see page 5715)

53.1.1.4.33 TBaseInetXPageProducer.SetStyles Method

Assigns a value to the Styles (see page 5716) property.

Pascal

```
procedure SetStyles(Value: TStrings);
```

C++

```
__fastcall SetStyles(TStrings Value);
```

Description

SetStyles is the write implementation of the Styles (see page 5716) property.

See Also

Styles (see page 5716)

53.1.1.4.34 TBaseInetXPageProducer.SetStylesFile Method

Assigns a value to the StylesFile (see page 5717) property.

Pascal

```
procedure SetStylesFile(const Value: TFileName);
```

C++

```
__fastcall SetStylesFile(const TFileName Value);
```

Description

SetStylesFile is the write implementation of the StylesFile (see page 5717) property.

See Also

StylesFile (see page 5717)

53.1.1.4.35 TBaseInetXPageProducer.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class TBaseInetXPageProducer.

53.1.1.5 TBaseInetXPageProducer Properties

The properties of the TBaseInetXPageProducer class are listed here.

Protected Properties

	Name	Description
	PageElements (see page 5716)	This is PageElements, a member of class TBaseInetXPageProducer.
	ScriptManager (see page 5716)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Properties

	Name	Description
	EnableXMLIslands (see page 5715)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	IncludePathURL (see page 5715)	Indicates the URL where the javascript libraries are installed.
	Styles (see page 5716)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 5717)	Specifies a file containing the individual styles for the page producer's style sheet.

53.1.1.5.1 TBaseInetXPageProducer.EnableXMLIslands Property

Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.

Pascal

```
property EnableXMLIslands: Boolean;
```

C++

```
__property Boolean EnableXMLIslands;
```

Description

Set EnableXMLIslands to indicate whether the XML in the generated HTML document should be declared using XML islands. XML islands are only used if the document is created in response to a request message whose headers indicate that the browser supports XML islands.

Set EnableXMLIslands to false to prevent the page producer from declaring XML islands even if the requesting browser supports them.

See Also

GetXMLData (see page 5712)

Content (see page 3466)

53.1.1.5.2 TBaseInetXPageProducer.IncludePathURL Property

Indicates the URL where the javascript libraries are installed.

Pascal

```
property IncludePathURL: string;
```

C++

```
__property UnicodeString IncludePathURL;
```

Description

Set IncludePathURL to indicate where the javascript libraries are installed. The page producer uses this value to create HTML statements that include the libraries in the generated HTML document. These statements have the form

```
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldb.js"> </SCRIPT>
```

The javascript libraries define classes and methods that allow the HTML document to interpret XML data packets if the Web browser is not XML-enabled, and that enable controls in the document to navigate the data, post changes, and apply updates.

See Also

SetIncludePathURL (see page 5713)

Content (see page 3466)

Creating Web Pages with an InternetExpress Page Producer

53.1.1.5.3 TBaseInetXPageProducer.PageElements Property

Pascal

```
property PageElements: TInetxPageElements;
```

C++

```
__property TInetxPageElements PageElements;
```

Description

This is PageElements, a member of class TBaseInetXPageProducer.

53.1.1.5.4 TBaseInetXPageProducer.ScriptManager Property

Pascal

```
property ScriptManager: IScriptManager;
```

C++

```
__property IScriptManager ScriptManager;
```

Description

This is ScriptManager, a member of class TBaseInetXPageProducer.

53.1.1.5.5 TBaseInetXPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseInetXPageProducer (see page 5701). Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile (see page 5717) property instead. If StylesFile (see page 5717) is defined, it takes precedence over the Styles property.

See Also

SetStyles (see page 5714)

StylesFile (see page 5717)

Content (see page 3466)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

53.1.1.5.6 TBaseInetXPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles (see page 5716) for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles (see page 5716) property instead.

See Also

SetStylesFile (see page 5714)

Styles (see page 5716)

Content (see page 3466)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

53.1.1.6 TBaseInetXPageProducer Events

The events of the TBaseInetXPageProducer class are listed here.

Public Events

	Name	Description
	OnAfterGetContent (see page 5718)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 5718)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 5719)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 5719)	Occurs before the page producer fetches an XML data packet from an XML broker.

53.1.1.6.1 TBaseInetXPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its HTML document.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the page producer generates an HTML document. This event occurs immediately before the Content method returns the generated document. Typically it is used in applications where the page producer automatically responds to messages as the Provider (see page 6441) property of an action item or the Response producer property of an XML broker.

See Also

OnBeforeGetContent (see page 5719)

Content (see page 3466)

Producer (see page 3342)

53.1.1.6.2 TBaseInetXPageProducer.OnAfterGetXMLData Event

Occurs after the page producer translates fetches an XML data packet from an XML broker.

Pascal

```
property OnAfterGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnAfterGetXMLData;
```

Description

Write an OnAfterGetXMLData event handler to take specific action immediately after the page producer obtains an XML data packet from an XML broker. Typically, the OnAfterGetXMLData event handler responds to any values returned by the provider on the application server as the OwnerData parameter.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnAfterGetXMLData occurs after the page producer has returned an XML data packet but before it is assigned as part of the emerging content.

See Also

OnBeforeGetXMLData (see page 5719)

GetXMLData (see page 5712)

OnHTMLTag (see page 3472)

Content (see page 3466)

53.1.1.6.3 TBaseInetXPageProducer.OnBeforeGetContent Event

Occurs when the Content method is called.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the page producer generates an HTML document. This event occurs when the Content method is called before the page producer starts assembling its HTML document. Typically it is used in applications where the page producer automatically responds to messages as the Provider (see page 6441) property of an action item or the Response producer property of an XML broker.

See Also

OnAfterGetContent (see page 5718)

Content (see page 3466)

Producer (see page 3342)

53.1.1.6.4 TBaseInetXPageProducer.OnBeforeGetXMLData Event

Occurs before the page producer fetches an XML data packet from an XML broker.

Pascal

```
property OnBeforeGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnBeforeGetXMLData;
```

Description

Write an OnBeforeGetXMLData event handler to take specific action immediately before the page producer requests an XML data packet from an XML broker. Typically, the OnBeforeGetXMLData event handler encodes any values for the provider on the application server as the OwnerData parameter or initializes properties on the XML broker that affect the data packet.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnBeforeGetXMLData occurs immediately before the page producer requests an XML data packet. Any value assigned to the OwnerData parameter is passed to the XML broker's OnRequestRecords event handler, and from there to the provider's OnBeforeGetRecords event handler.

See Also

OnAfterGetXMLData (see page 5718)

OnRequestRecords

GetXMLData (see page 5712)

Content (see page 3466)

OnHTMLTag (see page 3472)

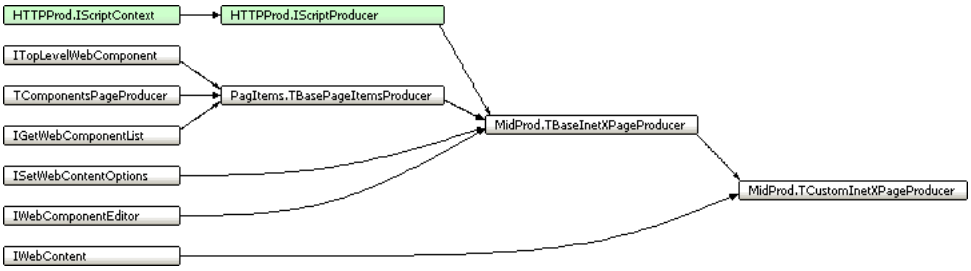
MaxRecords

Params

53.1.2 TCustomInetXPageProducer Class

TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.

Class Hierarchy



Pascal

```
TCustomInetXPageProducer = class(TBaseInetXPageProducer, IWebContent);
```

C++

```
class TCustomInetXPageProducer : public TBaseInetXPageProducer, public IWebContent;
```

File

MidProd

Description

Use TCustomInetXPageProducer as a base class for components that create Web pages in an InternetExpress Web application. These pages display database information from an application server and include data packets encoded in XML and javascript that lets Web browsers navigate that data, post changes, and apply updates back to the application server.

The InetX page producer creates its HTML document from a template. The InetX page producer has a default template that coordinates several top-level declarations for the HTML document with content produced by other components. The other components that generate HTML content for the InetX page producer can be added at design time using the Web Page editor.

Do not instantiate TCustomInetXPageProducer. To create a producer that generates HTML pages with javascript database functions and XML data packets, use a descendant class such as TInetXPageProducer (see page 5730).

See Also

TXMLBroker

53.1.2.1 TCustomInetXPageProducer Members

The following tables list the members exposed by TCustomInetXPageProducer.

Protected Methods

	Name	Description
	AddIncludes (see page 5705)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 5705)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	CanAddClass (see page 5705)	Indicates whether a specified class can be used as a child Web item.
	CreatePageElements (see page 5706)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 5707)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 5707)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 5707)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 5707)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 5708)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 5708)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 5708)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 5708)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 5708)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DoAfterGetContent (see page 5709)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 5709)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 5709)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 5709)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 5710)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	GeneratePageElements (see page 5711)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetContentOptions (see page 5711)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetDefaultTemplate (see page 5711)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetScriptManager (see page 5712)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetTemplateFileName (see page 5725)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
	GetTemplateStream (see page 5725)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	GetUseXMLIslands (see page 5712)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 5713)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 5713)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	ImplCanAddClass (see page 5713)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetWebContentOptions (see page 5714)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.
	WebContent (see page 5725)	This is WebContent, a member of class TCustomInetXPageProducer.

Protected Properties

	Name	Description
	PageElements (see page 5716)	This is PageElements, a member of class TBaseInetXPageProducer.
	ScriptManager (see page 5716)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Events

	Name	Description
	OnAfterGetContent (see page 5718)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 5718)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 5719)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 5719)	Occurs before the page producer fetches an XML data packet from an XML broker.

Public Methods

	Name	Description
	Content (see page 5723)	Returns the HTML document assembled by the InternetExpress page producer.

	ContentFromStream (see page 5706)	Returns the content produced in response to input read from a stream.
	Create (see page 5722)	Creates and initializes an instance of TCustomInetXPageProducer (see page 5720).
	Destroy (see page 5723)	Frees an instance of TCustomInetXPageProducer (see page 5720).
	Evaluate (see page 3509)	This is Evaluate, a member of class IScriptProducer.
	FindXMLBroker (see page 5710)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 5710)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	GetContent (see page 3509)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 3509)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 3510)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 3510)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetRequiredTags (see page 5711)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.
	GetXMLData (see page 5712)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 3510)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 3510)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 3510)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 3511)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 3511)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 5713)	Assigns a value to the IncludePathURL (see page 5715) property.
	SetStyles (see page 5714)	Assigns a value to the Styles (see page 5716) property.
	SetStylesFile (see page 5714)	Assigns a value to the StylesFile (see page 5717) property.
	Write (see page 3511)	This is the overview for the Write method overload.
	WriteItem (see page 3512)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
	EnableXMLIslands (see page 5715)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
R	Errors (see page 3512)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 3513)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 3513)	This is HTMLBlocks, a member of class IScriptProducer.
	HTMLDoc (see page 5726)	Specifies the HTML template used to generate the value returned by the Content (see page 5723) method.
	HTMLFile (see page 5726)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 5723) method.
	IncludePathURL (see page 5715)	Indicates the URL where the javascript libraries are installed.
	Styles (see page 5716)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 5717)	Specifies a file containing the individual styles for the page producer's style sheet.
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.
R	WebPageItems (see page 6393)	Maintains a list of components that generate HTML content for the producer.

53.1.2.2 TCustomInetXPageProducer.Create Constructor

Creates and initializes an instance of TCustomInetXPageProducer (see page 5720).

Pascal

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

C++

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

Description

Call Create to instantiate an InternetExpress page producer at runtime. InternetExpress page producers placed in a Web module at design time are instantiated automatically.

AOwner is a component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for freeing the TCustomInetXPageProducer (see page 5720) instance.

See Also

Owner

53.1.2.3 TCustomInetXPageProducer.Destroy Destructor

Frees an instance of TCustomInetXPageProducer (see page 5720).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomInetXPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free

53.1.2.4 TCustomInetXPageProducer Methods

The methods of the TCustomInetXPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 5725)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
	GetTemplateStream (see page 5725)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	WebContent (see page 5725)	This is WebContent, a member of class TCustomInetXPageProducer.

Public Methods

	Name	Description
	Content (see page 5723)	Returns the HTML document assembled by the InternetExpress page producer.

53.1.2.4.1 TCustomInetXPageProducer.Content Method

Returns the HTML document assembled by the InternetExpress page producer.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall UnicodeString Content();
```

Description

Call Content to get the HTML document produced by TCustomInetXPageProducer (see page 5720). This document contains a mixture of HTML, XML, and javascript. It allows Web browsers to view and edit data from an application server.

The InternetExpress page producer creates its content by replacing HTML-transparent tags in the template specified by HTMLDoc (see page 5726) or HTMLFile (see page 5726).

By default, this template has the form

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Although the template can be changed to include any combination of these tags with tags of the following forms

```
<#COMPONENT Name=WebItemName>
<#DATAPACKET XMLBroker=BrokerName>
```

which are also translated automatically. The template can replace the default HTML-transparent tags with the tag

```
<#BODYELEMENTS>
```

Finally, the template can include any tags translated by the OnHTMLTag event handler.

The Content method translates the tags listed above as follows:

<#INCLUDES> generates the statements that include the javascript libraries at the location specified by IncludePathURL (see page 5715).

<#STYLES> generates the statements that defines a style sheet from the definitions listed in the Styles (see page 5716) or StylesFile (see page 5717) property.

<#WARNINGS> generates nothing at runtime. At design time, it adds warning messages for problems detected while generating the HTML document. You can see these messages in the Web page editor.

<#FORMS> generates the HTML produced by the components listed in the WebPageItems property. The HTML from each component is listed in the order it appears in WebPageItems.

<#SCRIPT> generates a block of javascript declarations that are used by the components listed in WebPageItems.

<#BODYELEMENTS> generates the same HTML as results from the 5 tags in the default template. It is useful when generating a template in an HTML editor when you want to use the default layout but add additional elements (such as style sheets) using the editor.

<#COMPONENT> is replaced with the content produced by the specified component. Typically, this is a component listed in the WebPageItems property, but it can be any component that supports the IWebContent interface and has the same Owner as the INETX page producer.

<#DATAPACKET> is replaced with the XML data packet obtained from the specified XML broker.

The string returned by Content can be used as the Content property of the response to an HTTP request message.

See Also

IWebContent

IncludePathURL (see page 5715)

Styles (see page 5716)

StylesFile (see page 5717)

OnHTMLTag (see page 3472)

HTMLDoc (see page 5726)

HTMLFile (see page 5726)

WebPageItems (see page 6393)

Customizing the InternetExpress Page Producer Template

53.1.2.4.2 TCustomInetXPageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TCustomInetXPageProducer.

53.1.2.4.3 TCustomInetXPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomInetXPageProducer.

53.1.2.4.4 TCustomInetXPageProducer.WebContent Method

Pascal

```
function WebContent(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString WebContent(TWebContentOptions Options, TLayout Layout);
```

Description

This is WebContent, a member of class TCustomInetXPageProducer.

53.1.2.5 TCustomInetXPageProducer Properties

The properties of the TCustomInetXPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 5726)	Specifies the HTML template used to generate the value returned by the Content (see page 5723) method.
	HTMLFile (see page 5726)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 5723) method.

53.1.2.5.1 TCustomInetXPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content (see page 5723) method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomInetXPageProducer (see page 5720) automatically handles the conversion of the HTML-transparent tags in the default template, as well as the tags described in the help for the Content (see page 5723) method. If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile (see page 5726) property.

See Also

Content (see page 5723)

HTMLFile (see page 5726)

OnHTMLTag (see page 3472)

Creating Web Pages with an InternetExpress Page Producer

Customizing the InternetExpress Page Producer Template

53.1.2.5.2 TCustomInetXPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 5723) method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 5726) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 5723)

HTMLDoc (see page 5726)

Customizing the InternetExpress Page Producer Template

53.1.3 TInetXPageElements Class

Class Hierarchy

Pascal

```
TInetXPageElements = class(TObject);
```

C++

```
class TInetXPageElements : public TObject;
```

File

MidProd

Description

This is class MidProd.TInetXPageElements.

53.1.3.1 TInetXPageElements Members

The following tables list the members exposed by TInetXPageElements.

Public Fields

	Name	Description
	FormsContent (see page 5728)	This is FormsContent, a member of class TInetXPageElements.
	IncludesContent (see page 5728)	This is IncludesContent, a member of class TInetXPageElements.
	ScriptContent (see page 5729)	This is ScriptContent, a member of class TInetXPageElements.
	StylesContent (see page 5729)	This is StylesContent, a member of class TInetXPageElements.
	WarningsContent (see page 5729)	This is WarningsContent, a member of class TInetXPageElements.

Public Methods

	Name	Description
	BodyContent (see page 5729)	This is BodyContent, a member of class TInetXPageElements.

53.1.3.2 TInetXPageElements Fields

The fields of the TInetXPageElements class are listed here.

Public Fields

	Name	Description
	FormsContent (see page 5728)	This is FormsContent, a member of class TInetXPageElements.
	IncludesContent (see page 5728)	This is IncludesContent, a member of class TInetXPageElements.
	ScriptContent (see page 5729)	This is ScriptContent, a member of class TInetXPageElements.
	StylesContent (see page 5729)	This is StylesContent, a member of class TInetXPageElements.
	WarningsContent (see page 5729)	This is WarningsContent, a member of class TInetXPageElements.

53.1.3.2.1 TInetXPageElements.FormsContent Field

Pascal

```
FormsContent: string;
```

C++

```
UnicodeString FormsContent;
```

Description

This is FormsContent, a member of class TInetXPageElements.

53.1.3.2.2 TInetXPageElements.IncludesContent Field

Pascal

```
IncludesContent: string;
```

C++

```
UnicodeString IncludesContent;
```

Description

This is IncludesContent, a member of class TInetXPageElements.

53.1.3.2.3 TInetXPageElements.ScriptContent Field

Pascal

```
ScriptContent: string;
```

C++

```
UnicodeString ScriptContent;
```

Description

This is ScriptContent, a member of class TInetXPageElements.

53.1.3.2.4 TInetXPageElements.StylesContent Field

Pascal

```
StylesContent: string;
```

C++

```
UnicodeString StylesContent;
```

Description

This is StylesContent, a member of class TInetXPageElements.

53.1.3.2.5 TInetXPageElements.WarningsContent Field

Pascal

```
WarningsContent: string;
```

C++

```
UnicodeString WarningsContent;
```

Description

This is WarningsContent, a member of class TInetXPageElements.

53.1.3.3 TInetXPageElements Methods

The methods of the TInetXPageElements class are listed here.

Public Methods

	Name	Description
	BodyContent (see page 5729)	This is BodyContent, a member of class TInetXPageElements.

53.1.3.3.1 TInetXPageElements.BodyContent Method

Pascal

```
function BodyContent: string; virtual;
```

C++

```
virtual __fastcall UnicodeString BodyContent();
```

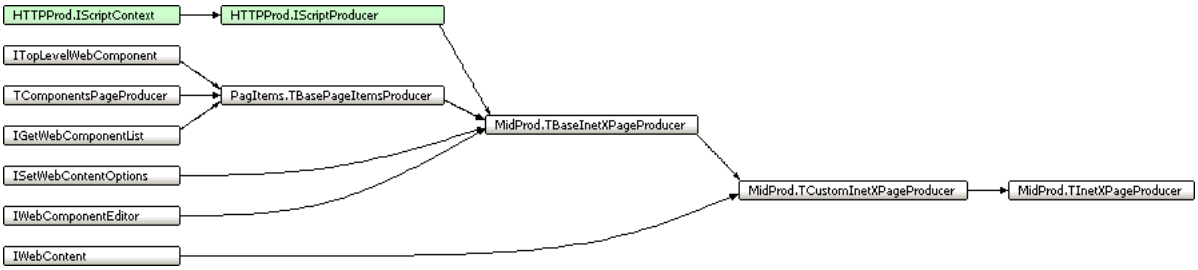
Description

This is BodyContent, a member of class TInetXPageElements.

53.1.4 TInetXPageProducer Class

TInetXPageProducer assembles an HTML document that represents live database information from an application server.

Class Hierarchy



Pascal

```
TInetXPageProducer = class(TCustomInetXPageProducer);
```

C++

```
class TInetXPageProducer : public TCustomInetXPageProducer;
```

File

MidProd

Description

Use TInetXPageProducer to create Web pages for an InternetExpress application. These pages display database information from an application server and include data packets encoded in XML and javascript that lets Web browsers navigate that data, post changes, and apply updates back to the application server.

The InetX page producer is assigned as the Producer property of a Web action item or called from an OnAction event handler.

The InetX page producer creates its HTML document from a template. Unlike other page producers, the InetX page producer has a default template that coordinates several top-level declarations for the HTML document with content produced by other components. The other components that generate HTML content for the InetX page producer can be added at design time using the Web Page editor.

See Also

TWebActionItem (see page 3335)

TWebActionItem (see page 3335)

TXMLBroker

The Web Module

53.1.4.1 TInetXPageProducer Members

The following tables list the members exposed by TInetXPageProducer.

Protected Methods

	Name	Description
	AddIncludes (see page 5705)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 5705)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	CanAddClass (see page 5705)	Indicates whether a specified class can be used as a child Web item.
	CreatePageElements (see page 5706)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 5707)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 5707)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 5707)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 5707)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 5708)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 5708)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 5708)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 5708)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 5708)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DoAfterGetContent (see page 5709)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 5709)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 5709)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 5709)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 5710)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	GeneratePageElements (see page 5711)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetContentOptions (see page 5711)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetDefaultTemplate (see page 5711)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetScriptManager (see page 5712)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetTemplateFileName (see page 5725)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
	GetTemplateStream (see page 5725)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	GetUseXMLIslands (see page 5712)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 5713)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 5713)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	ImplCanAddClass (see page 5713)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetWebContentOptions (see page 5714)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.
	WebContent (see page 5725)	This is WebContent, a member of class TCustomInetXPageProducer.

Protected Properties

	Name	Description
	PageElements (see page 5716)	This is PageElements, a member of class TBaseInetXPageProducer.
	ScriptManager (see page 5716)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Methods

	Name	Description
	Content (see page 5723)	Returns the HTML document assembled by the InternetExpress page producer.
	ContentFromStream (see page 5706)	Returns the content produced in response to input read from a stream.
	Create (see page 5703)	Creates an instance of a TBaseInetXPageProducer (see page 5701).
	Create (see page 5722)	Creates and initializes an instance of TCustomInetXPageProducer (see page 5720).
	Create (see page 6391)	Creates an instance of a TBasePageItemsProducer (see page 6390).
	Destroy (see page 5704)	Destroys the instance of the TBasePageItemsProducer.
	Destroy (see page 5723)	Frees an instance of TCustomInetXPageProducer (see page 5720).
	Destroy (see page 6391)	Destroys the instance of the TBasePageItemsProducer (see page 6390).
	Evaluate (see page 3509)	This is Evaluate, a member of class IScriptProducer.

	FindXMLBroker (see page 5710)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 5710)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	GetContent (see page 3509)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 3509)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 3510)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 3510)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetRequiredTags (see page 5711)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 3507)	This is GetWebModuleContext, a member of class IScriptContext.
	GetXMLData (see page 5712)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 3510)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 3510)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 3510)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 3511)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 3511)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 5713)	Assigns a value to the IncludePathURL (see page 5715) property.
	SetStyles (see page 5714)	Assigns a value to the Styles (see page 5716) property.
	SetStylesFile (see page 5714)	Assigns a value to the StylesFile (see page 5717) property.
	Write (see page 3511)	This is the overview for the Write method overload.
	WriteItem (see page 3512)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
	Content (see page 3512)	This is Content, a member of class IScriptProducer.
R	Errors (see page 3512)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 3513)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 3513)	This is HTMLBlocks, a member of class IScriptProducer.
R	WebModuleContext (see page 3507)	This is WebModuleContext, a member of class IScriptContext.

Published Events

	Name	Description
	OnAfterGetContent (see page 5738)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 5738)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 5739)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 5739)	Occurs before the page producer fetches an XML data packet from an XML broker.

Published Properties

	Name	Description
	EnableXMLIslands (see page 5733)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	HTMLDoc (see page 5733)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 5734)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	IncludePathURL (see page 5735)	Indicates the URL where the javascript libraries are installed.
	ScriptEngine (see page 5735)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	Styles (see page 5736)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 5736)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 5737)	Maintains a list of components that generate HTML content for the producer.

53.1.4.2 TInetXPageProducer Properties

The properties of the TInetXPageProducer class are listed here.

Published Properties

	Name	Description
	EnableXMLIslands (see page 5733)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	HTMLDoc (see page 5733)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 5734)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	IncludePathURL (see page 5735)	Indicates the URL where the javascript libraries are installed.
	ScriptEngine (see page 5735)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	Styles (see page 5736)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 5736)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 5737)	Maintains a list of components that generate HTML content for the producer.

53.1.4.2.1 TInetXPageProducer.EnableXMLIslands Property

Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.

Pascal

```
property EnableXMLIslands: Boolean;
```

C++

```
__property Boolean EnableXMLIslands;
```

Description

Set EnableXMLIslands to indicate whether the XML in the generated HTML document should be declared using XML islands. XML islands are only used if the document is created in response to a request message whose headers indicate that the browser supports XML islands.

Set EnableXMLIslands to false to prevent the page producer from declaring XML islands even if the requesting browser supports them.

See Also

GetXMLData (see page 5712)

Content (see page 3466)

53.1.4.2.2 TInetXPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomInetXPageProducer automatically handles the conversion of the HTML-transparent tags in the default template, as well as the tags described in the help for the Content method. If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 5723)

HTMLFile (see page 5726)

OnHTMLTag (see page 3472)

Creating Web Pages with an InternetExpress Page Producer

Customizing the InternetExpress Page Producer Template

53.1.4.2.3 TInetXPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where

there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 5723)

HTMLDoc (see page 5726)

Customizing the InternetExpress Page Producer Template

53.1.4.2.4 TInetXPageProducer.IncludePathURL Property

Indicates the URL where the javascript libraries are installed.

Pascal

```
property IncludePathURL: string;
```

C++

```
__property UnicodeString IncludePathURL;
```

Description

Set IncludePathURL to indicate where the javascript libraries are installed. The page producer uses this value to create HTML statements that include the libraries in the generated HTML document. These statements have the form

```
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldb.js"> </SCRIPT>
```

The javascript libraries define classes and methods that allow the HTML document to interpret XML data packets if the Web browser is not XML-enabled, and that enable controls in the document to navigate the data, post changes, and apply updates.

See Also

SetIncludePathURL (see page 5713)

Content (see page 3466)

Creating Web Pages with an InternetExpress Page Producer

53.1.4.2.5 TInetXPageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine;
```

C++

```
__property ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer

generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 3472)

53.1.4.2.6 TInetXPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseInetXPageProducer. Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile property instead. If StylesFile is defined, it takes precedence over the Styles property.

See Also

SetStyles (see page 5714)

StylesFile (see page 5717)

Content (see page 3466)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

53.1.4.2.7 TInetXPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles property instead.

See Also

SetStylesFile (see page 5714)

Styles (see page 5716)

Content (see page 3466)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

53.1.4.2.8 TInetXPageProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems: TWebComponentList;
```

C++

```
__property TWebComponentList WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer. Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent

Creating Web Pages with an InternetExpress Page Producer

53.1.4.3 TInetXPageProducer Events

The events of the TInetXPageProducer class are listed here.

Published Events

	Name	Description
	OnAfterGetContent (see page 5738)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 5738)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 5739)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 5739)	Occurs before the page producer fetches an XML data packet from an XML broker.

53.1.4.3.1 TInetXPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its HTML document.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the page producer generates an HTML document. This event occurs immediately before the Content method returns the generated document. Typically it is used in applications where the page producer automatically responds to messages as the Provider property of an action item or the Response producer property of an XML broker.

See Also

OnBeforeGetContent (see page 5719)

Content (see page 3466)

Producer (see page 3342)

53.1.4.3.2 TInetXPageProducer.OnAfterGetXMLData Event

Occurs after the page producer translates fetches an XML data packet from an XML broker.

Pascal

```
property OnAfterGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnAfterGetXMLData;
```

Description

Write an OnAfterGetXMLData event handler to take specific action immediately after the page producer obtains an XML data packet from an XML broker. Typically, the OnAfterGetXMLData event handler responds to any values returned by the provider on the application server as the OwnerData parameter.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnAfterGetXMLData occurs after the page producer has returned an XML data packet but before it is assigned as part of the emerging content.

See Also

OnBeforeGetXMLData (see page 5719)

GetXMLData (see page 5712)

OnHTMLTag (see page 3472)

Content (see page 3466)

53.1.4.3.3 TInetXPageProducer.OnBeforeGetContent Event

Occurs when the Content method is called.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the page producer generates an HTML document. This event occurs when the Content method is called before the page producer starts assembling its HTML document. Typically it is used in applications where the page producer automatically responds to messages as the Provider property of an action item or the Response producer property of an XML broker.

See Also

OnAfterGetContent (see page 5718)

Content (see page 3466)

Producer (see page 3342)

53.1.4.3.4 TInetXPageProducer.OnBeforeGetXMLData Event

Occurs before the page producer fetches an XML data packet from an XML broker.

Pascal

```
property OnBeforeGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnBeforeGetXMLData;
```

Description

Write an OnBeforeGetXMLData event handler to take specific action immediately before the page producer requests an XML data packet from an XML broker. Typically, the OnBeforeGetXMLData event handler encodes any values for the provider on the application server as the OwnerData parameter or initializes properties on the XML broker that affect the data packet.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnBeforeGetXMLData occurs immediately before the page producer requests an XML data packet. Any value assigned to the OwnerData parameter is passed to the XML broker's OnRequestRecords event handler, and from there to the provider's OnBeforeGetRecords event handler.

See Also

OnAfterGetXMLData (see page 5718)

OnRequestRecords

GetXMLData (see page 5712)

Content (see page 3466)

OnHTMLTag (see page 3472)

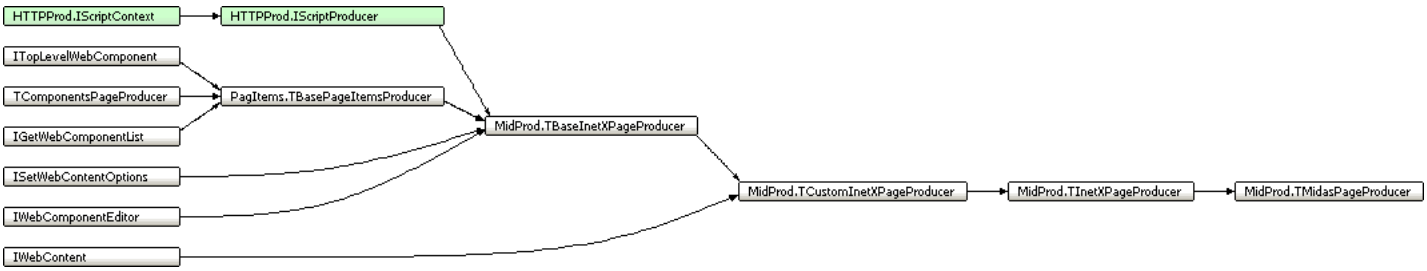
MaxRecords

Params

53.1.5 TMidasPageProducer Class

TMidasPageProducer assembles an HTML document that represents live database information from an application server.

Class Hierarchy



Pascal

```
TMidasPageProducer = class(TInetXPageProducer);
```

C++

```
class TMidasPageProducer : public TInetXPageProducer;
```

File

MidProd

Description

Do not use TMidasPageProducer. It is included only for backward compatibility. To add a page producer that assembles Web pages in an InternetExpress application, use TInetXPageProducer (see page 5730) instead.

See Also

TWebActionItem (see page 3335)

TWebActionItem (see page 3335)

TXMLBroker

53.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddStdIncludes (see page 5741)	This is function MidProd.AddStdIncludes.
	PathInfoToRelativePath (see page 5741)	This is function MidProd.PathInfoToRelativePath.

53.2.1 MidProd.AddStdIncludes Function

Pascal

```
procedure AddStdIncludes(StdIncludes: TStdIncludes; AddIntf: IAddScriptElements);
```

C++

```
AddStdIncludes(TStdIncludes StdIncludes, IAddScriptElements AddIntf);
```

File

MidProd

Description

This is function MidProd.AddStdIncludes.

53.2.2 MidProd.PathInfoToRelativePath Function

Pascal

```
function PathInfoToRelativePath(const PathInfo: string): string;
```

C++

```
UnicodeString PathInfoToRelativePath(const UnicodeString PathInfo);
```

File

MidProd

Description

This is function MidProd.PathInfoToRelativePath.

53.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStdInclude (see page 5742)	This is record MidProd.TStdInclude.

53.3.1 MidProd.TStdInclude Enumeration

Pascal

```
TStdInclude = (
  jsDom,
  jsDB,
  jsDisp,
  jsBind,
  jsErrDisp,
  jsShow
);
```

C++

```
enum TStdInclude {
  jsDom,
  jsDB,
  jsDisp,
  jsBind,
  jsErrDisp,
  jsShow
};
```

File

MidProd

Description

This is record MidProd.TStdInclude.

53.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBaseMidasPageProducer (see page 5743)	This is type MidProd.TBaseMidasPageProducer.
TCustomMidasPageProducer (see page 5743)	This is type MidProd.TCustomMidasPageProducer.
TMidasPageElements (see page 5744)	This is type MidProd.TMidasPageElements.
TStdIncludes (see page 5744)	This is type MidProd.TStdIncludes.
TXMLDataEvent (see page 5744)	This is type MidProd.TXMLDataEvent.

53.4.1 MidProd.TBaseMidasPageProducer Type

Pascal

```
TBaseMidasPageProducer = TBaseInetXPageProducer;
```

C++

```
TBaseInetXPageProducer TBaseMidasPageProducer;
```

File

MidProd

Description

This is type MidProd.TBaseMidasPageProducer.

53.4.2 MidProd.TCustomMidasPageProducer Type

Pascal

```
TCustomMidasPageProducer = TCustomInetXPageProducer;
```

C++

```
TCustomInetXPageProducer TCustomMidasPageProducer;
```

File

MidProd

Description

This is type MidProd.TCustomMidasPageProducer.

53.4.3 MidProd.TMidasPageElements Type

Pascal

```
TMidasPageElements = TInetXPageElements;
```

C++

```
TInetXPageElements TMidasPageElements;
```

File

MidProd

Description

This is type MidProd.TMidasPageElements.

53.4.4 MidProd.TStdIncludes Type

Pascal

```
TStdIncludes = set of TStdInclude;
```

C++

```
TStdInclude TStdIncludes;
```

File

MidProd

Description

This is type MidProd.TStdIncludes.

53.4.5 MidProd.TXMLDataEvent Type

Pascal

```
TXMLDataEvent = procedure (Sender: TObject; Request: TWebRequest; XMLBroker: TXMLBroker; var
OwnerData: OleVariant) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; XMLBroker: TXMLBroker; var OwnerData: OleVariant) (
TXMLDataEvent)();
```

File

MidProd

Description

This is type MidProd.TXMLDataEvent.

53.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sXMLBroker (see page 5745)	This is constant MidProd.sXMLBroker.
sXMLReadyVar (see page 5745)	This is constant MidProd.sXMLReadyVar.

53.5.1 MidProd.sXMLBroker Constant

Pascal

```
sXMLBroker = 'XMLBROKER';
```

C++

```
sXMLBroker = 'XMLBROKER';
```

File

MidProd

Description

This is constant MidProd.sXMLBroker.

53.5.2 MidProd.sXMLReadyVar Constant

Pascal

```
sXMLReadyVar = 'xml_ready';
```

C++

```
sXMLReadyVar = 'xml_ready';
```

File

MidProd

Description

This is constant MidProd.sXMLReadyVar.

54 MPlayer Namespace

This is namespace MPlayer.

Topics

Name	Description
Classes (see page 5749)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 5819)	The following table lists structs, records, enums in this documentation.
Types (see page 5827)	The following table lists types in this documentation.

Classes

	Name	Description
	EMCIDeviceError (see page 5749)	EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.
	TMediaPlayer (see page 5749)	TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

Structs, Records, Enums

	Name	Description
	TMPBtnType (see page 5819)	TMPBtnType defines the possible buttons in a TMediaPlayer (see page 5749) component and TButtonSet (see page 5828) represents a set of those buttons.
	TMPButton (see page 5820)	This is record MPlayer.TMPButton.
	TMPDevCaps (see page 5821)	TMPDevCaps defines the capabilities of a TMediaPlayer (see page 5749) component and TMPDevCapsSet (see page 5828) represents a set of those capabilities.
	TMPDeviceTypes (see page 5821)	TMPDeviceTypes identifies a multimedia device.
	TMPGlyph (see page 5823)	This is record MPlayer.TMPGlyph.
	TMPModes (see page 5823)	TMPModes identifies the state of a TMediaPlayer (see page 5749) component.
	TMPNotifyValues (see page 5824)	TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 5749) component.
	TMPTimeFormats (see page 5825)	TMPTimeFormats indicates the format a media player uses to indicate position.

Types

Name	Description
EMPNotify (see page 5827)	EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 5749) component.
EMPPostNotify (see page 5827)	EMPPostNotify is the type of a media player's OnPostClick event handler.
TButtonSet (see page 5828)	TButtonSet is a set of TMPBtnType (see page 5819) values.

TMediaDevice (see page 5828)	This is type MPlayer.TMediaDevice.
TMPDevCapsSet (see page 5828)	This is type MPlayer.TMPDevCapsSet.

54.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMCIDeviceError (see page 5749)	EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.
	TMediaPlayer (see page 5749)	TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

54.1.1 EMCIDeviceError Class

EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.

Class Hierarchy

Pascal

```
EMCIDeviceError = class(Exception);
```

C++

```
class EMCIDeviceError : public Exception;
```

File

MPlayer

Description

EMCIDeviceError is raised if an error occurs when accessing a multimedia device. The most common cause for the exception is trying to access a multimedia device before it has been opened with the Open method of TMediaPlayer (see page 5749).

See Also

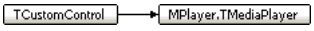
TMediaPlayer (see page 5749)

TMediaPlayer (see page 5749)

54.1.2 TMediaPlayer Class

TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

Class Hierarchy



Pascal

```
TMediaPlayer = class(TCustomControl);
```

C++

```
class TMediaPlayer : public TCustomControl;
```

File

MPlayer

Description

The TMediaPlayer component includes a set of buttons (Play (see page 5767), Stop (see page 5774), Eject (see page 5759), and so on) that control a multimedia device such as a CD-ROM drive, MIDI sequencer, or VCR. A multimedia device may be hardware or software.

The media player component contains of multiple buttons. These buttons can be clicked with the mouse, but are not separate objects or button components.

ButtonValueAction

Button	Value	Action
Play (see page 5767)	btPlay	Plays the media player
Pause (see page 5766)	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Stop (see page 5774)	btStop	Stops playing or recording
Next (see page 5762)	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step (see page 5774)	btStep	Moves forward a number of frames
Back (see page 5757)	btBack	Moves backward a number of frames
Record	btRecord	Starts recording
Eject (see page 5759)	btEject	Ejects the medium

The multimedia device is played, paused, stopped, and so on when the user clicks the corresponding button on the TMediaPlayer component. The device can also be controlled by the control methods that correspond to the buttons (Play (see page 5767), Pause (see page 5766), Stop (see page 5774), Next (see page 5762), Previous (see page 5771), Step (see page 5774), Back (see page 5757), StartRecording (see page 5773), and Eject (see page 5759)).

The type of multimedia device (such as dtWaveAudio or dtVideodisc) is specified by the DeviceType (see page 5783) property. If the device stores its media in a file, the name of the media file is specified by the FileName (see page 5791) property. If DeviceType (see page 5783) is dtAutoSelect, the media player attempts to determine the type of device from the extension of the file specified by FileName (see page 5791).

To have the media player attempt to open the device specified by DeviceType (see page 5783) automatically when the media player component is created at runtime, set the AutoOpen (see page 5780) property to true.

C++ Examples:

```
//
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
//the buttons manually.
```

```
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
    end;
  end;

  OpenMediaDialog.Free;
end;

procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
  Button: TMPBtnType);
```

```
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

Working with Multimedia

54.1.2.1 TMediaPlayer Members

The following tables list the members exposed by TMediaPlayer.

Protected Methods

	Name	Description
	AutoButtonSet (see page 5756)	Enables or Disables the media player's buttons.
	Click (see page 5757)	Generates an OnClick (see page 5810) event.
	DoNotify (see page 5759)	Generates an OnNotify (see page 5817) event.
V	KeyDown (see page 5761)	Provides responses when the user types arrow keys or the space bar.
V	Loaded (see page 5761)	Initializes the media player after it is loaded from a stream.
	MMNotify (see page 5762)	Responds to a notification that the MCI device has completed an operation.
V	Notification (see page 5763)	Responds when components are about to be inserted or removed.
V	Paint (see page 5766)	Renders the image of the media player.
	PostClick (see page 5770)	Determines the action that takes place when an OnPostClick (see page 5818) event occurs.
V	Updated (see page 5775)	Signals that the media player has finished updating.

Public Methods

	Name	Description
	Back (see page 5757)	Steps backward a number of frames (determined by the value of the Frames (see page 5793) property) in the currently loaded medium.
	Close (see page 5758)	Closes the open multimedia device.
V	Create (see page 5755)	Creates a media player object.
V	Destroy (see page 5755)	Destroys an instance of the media player.
	Eject (see page 5759)	Releases the loaded medium from the open multimedia device.
	Next (see page 5762)	Moves to the beginning of the next track of the currently loaded medium.
	Open (see page 5763)	Opens a multimedia device.
	Pause (see page 5766)	Toggles the open multimedia device on or off.
	PauseOnly (see page 5767)	Pauses the open multimedia device.
	Play (see page 5767)	Plays the media loaded in the open multimedia device.
	Previous (see page 5771)	Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.
	Resume (see page 5771)	Resumes playing or recording the currently paused multimedia device.
	Rewind (see page 5772)	Sets the current position to the beginning of the the medium, which is stored in the Start (see page 5804) property.
	Save (see page 5772)	Saves the currently loaded medium to the file specified in the FileName (see page 5791) property.
	StartRecording (see page 5773)	Begins recording from the current Position (see page 5801) or from the position specified in StartPos (see page 5804).
	Step (see page 5774)	Moves forward a number of frames (determined by the Frames (see page 5793) property) in the currently loaded medium.
	Stop (see page 5774)	Halts playing or recording.

Public Properties

	Name	Description
R	Capabilities (see page 5781)	Determines the capabilities of the open multimedia device.
R	DeviceID (see page 5782)	Specifies the device ID for the currently open multimedia device.

	DisplayRect (see page 5784)	Specifies a rectangular area in the control specified by the Display (see page 5783) property that is used to display output from a multimedia device.
	EndPos (see page 5789)	Specifies the position within the currently loaded medium at which to stop playing or recording.
R	Error (see page 5789)	Specifies the MCI error code returned by the media control method.
R	ErrorMessage (see page 5790)	Describes the error code stored in the Error (see page 5789) property.
	Frames (see page 5793)	Specifies the number of frames moved forward or backward.
R	Length (see page 5794)	Specifies the length of the medium in the open multimedia device.
R	Mode (see page 5794)	Indicates the state of the currently open multimedia device.
	Notify (see page 5794)	Determines whether an OnNotify (see page 5817) event is generated.
R	NotifyValue (see page 5795)	Indicates the result of the last media control method that requested a notification.
	Position (see page 5801)	Specifies the current position within the currently loaded medium.
R	Start (see page 5804)	Specifies the starting position within the currently loaded medium.
	StartPos (see page 5804)	Specifies the position within the currently loaded medium from which to begin playing or recording.
	TimeFormat (see page 5805)	Determines the format used to specify position information.
R	TrackLength (see page 5806)	Reports the length of the track.
R	TrackPosition (see page 5806)	Reports the starting position of the track.
R	Tracks (see page 5807)	Specifies the number of playable tracks on the open multimedia device.
	Wait (see page 5809)	Determines whether a media control method returns control to the application only after it has been completed.

Published Events

	Name	Description
	OnClick (see page 5810)	Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.
	OnContextPopup (see page 5811)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 5811)	Occurs when a control receives the input focus.
	OnExit (see page 5814)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 5817)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5817)	Occurs when the user moves the mouse outside of a control.
	OnNotify (see page 5817)	Occurs upon the completion of a media control method.
	OnPostClick (see page 5818)	Occurs after the OnClick (see page 5810) event handler is called.

Published Properties

	Name	Description
	Anchors (see page 5776)	Specifies how the control is anchored to its parent.
	AutoEnable (see page 5777)	Determines whether the media player automatically enables and disables individual buttons in the component.
	AutoOpen (see page 5780)	Determines if the media player is opened automatically when the application is run.
	AutoRewind (see page 5781)	Determines if the media player control rewinds before playing or recording.
	ColoredButtons (see page 5781)	Determines which buttons on the media player control have color.
	Constraints (see page 5782)	Specifies the size constraints for the control.
	DeviceType (see page 5783)	Specifies a multimedia device type to open the media player.
	Display (see page 5783)	Specifies the display window for a multimedia device that uses a window for output.
	DoubleBuffered (see page 5784)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5786)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnabledButtons (see page 5786)	Determines which buttons on the media player are usable.
	FileName (see page 5791)	Specifies the opened or saved media file.
	ParentDoubleBuffered (see page 5795)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 5796)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5796)	Identifies the pop-up menu associated with the control.

	Shareable (see page 5802)	Determines whether more than one application can share a multimedia device.
	ShowHint (see page 5802)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5804)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5805)	Determines if the user can tab to a control.
	Visible (see page 5807)	Determines whether the component appears onscreen.
	VisibleButtons (see page 5809)	Indicates which of the buttons on the media player are visible.

54.1.2.2 TMediaPlayer.Create Constructor

Creates a media player object.

Pascal

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

C++

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

Description

Call Create to instantiate a media player at runtime. For media players created at design time, Create is called automatically.

Create allocates memory for a media player and calls the inherited Create. Then it loads the bitmaps for the media player's buttons and initializes its properties, setting AutoEnable (see page 5777), AutoRewind (see page 5781), Colored, Enabled (see page 5786), and Visible (see page 5807) to true, AutoOpen (see page 5780) to false, and DeviceType (see page 5783) to dtAutoSelect.

See Also

Create

ParentWindow

54.1.2.3 TMediaPlayer.Destroy Destructor

Destroys an instance of the media player.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMediaPlayer();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free checks that the TMediaPlayer (see page 5749) reference is not nil before it calls Destroy.

Destroy first verifies that no device is open, then destroys the bitmaps used for the media player's buttons and calls the inherited Destroy.

See Also

Destroy

System::TObject::Free

54.1.2.4 TMediaPlayer Methods

The methods of the TMediaPlayer class are listed here.

Protected Methods

	Name	Description
	AutoButtonSet (see page 5756)	Enables or Disables the media player's buttons.
	Click (see page 5757)	Generates an OnClick (see page 5810) event.
	DoNotify (see page 5759)	Generates an OnNotify (see page 5817) event.
V	KeyDown (see page 5761)	Provides responses when the user types arrow keys or the space bar.
V	Loaded (see page 5761)	Initializes the media player after it is loaded from a stream.
	MMNotify (see page 5762)	Responds to a notification that the MCI device has completed an operation.
V	Notification (see page 5763)	Responds when components are about to be inserted or removed.
V	Paint (see page 5766)	Renders the image of the media player.
	PostClick (see page 5770)	Determines the action that takes place when an OnPostClick (see page 5818) event occurs.
V	Updated (see page 5775)	Signals that the media player has finished updating.

Public Methods

	Name	Description
	Back (see page 5757)	Steps backward a number of frames (determined by the value of the Frames (see page 5793) property) in the currently loaded medium.
	Close (see page 5758)	Closes the open multimedia device.
	Eject (see page 5759)	Releases the loaded medium from the open multimedia device.
	Next (see page 5762)	Moves to the beginning of the next track of the currently loaded medium.
	Open (see page 5763)	Opens a multimedia device.
	Pause (see page 5766)	Toggles the open multimedia device on or off.
	PauseOnly (see page 5767)	Pauses the open multimedia device.
	Play (see page 5767)	Plays the media loaded in the open multimedia device.
	Previous (see page 5771)	Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.
	Resume (see page 5771)	Resumes playing or recording the currently paused multimedia device.
	Rewind (see page 5772)	Sets the current position to the beginning of the the medium, which is stored in the Start (see page 5804) property.
	Save (see page 5772)	Saves the currently loaded medium to the file specified in the FileName (see page 5791) property.
	StartRecording (see page 5773)	Begins recording from the current Position (see page 5801) or from the position specified in StartPos (see page 5804).
	Step (see page 5774)	Moves forward a number of frames (determined by the Frames (see page 5793) property) in the currently loaded medium.
	Stop (see page 5774)	Halts playing or recording.

54.1.2.4.1 TMediaPlayer.AutoButtonSet Method

Enables or Disables the media player's buttons.

Pascal

```
procedure AutoButtonSet(Btn: TMPBtnType); dynamic;
```

C++

```
__fastcall AutoButtonSet(TMPBtnType Btn);
```

Description

AutoButtonSet implements the automatic enabling and disabling of the media player's buttons when the AutoEnable (see page 5777) property is true. The Btn parameter indicates which of the media player's buttons the user pressed.

See Also

AutoEnable (see page 5777)

EnabledButtons (see page 5786)

54.1.2.4.2 TMediaPlayer.Back Method

Steps backward a number of frames (determined by the value of the Frames (see page 5793) property) in the currently loaded medium.

Pascal

```
procedure Back;
```

C++

```
__fastcall Back();
```

Description

Back is called when the Back button on the media player control is clicked at runtime.

Upon completion, Back stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Back method has been completed. The Notify (see page 5794) property determines whether Back generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.3 TMediaPlayer.Click Method

Generates an OnClick (see page 5810) event.

Pascal

```
procedure Click(Button: TMPBtnType; var DoDefault: Boolean); dynamic; reintroduce;
```

C++

```
__fastcall Click(TMPBtnType Button, Boolean DoDefault);
```

Description

Click for media-player components is the implementation method for the OnClick (see page 5810) event. By default, Click does nothing other than call any event handler attached to the media player's OnClick (see page 5810) event. Click can be overridden in descendant classes to customize responses to clicks.

See Also

OnClick (see page 5810)

54.1.2.4.4 TMediaPlayer.Close Method

Closes the open multimedia device.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Upon completion, Close stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Close method is completed. The Notify (see page 5794) property determines whether Close generates an OnNotify (see page 5817) event.

Close is called automatically when the application is terminated.

C++ Examples:

```
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
```

```
    TimerOver: Word;

  procedure TForm1.FormClick(Sender: TObject);
  begin
    MediaPlayer1.DeviceType := dtCDAudio;
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;

  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    if TimeOver = 10 then
    begin
      MediaPlayer1.Eject;
      MediaPlayer1.Close;
      { disable the timer, as we are done }
      Timer1.Enabled := False;
    end
    else
      if (MediaPlayer1.Mode = mpOpen) then
       Inc(TimeOver);
  end;
```

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

54.1.2.4.5 TMediaPlayer.DoNotify Method

Generates an OnNotify (see page 5817) event.

Pascal

```
procedure DoNotify; dynamic;
```

C++

```
__fastcall DoNotify();
```

Description

DoNotify is the implementation method for a media-player component's OnNotify (see page 5817) event. The media player's MMNotify (see page 5762) method calls DoNotify after correcting the enabling and disabling of buttons and setting internal flags from the values passed in the notification message parameters.

By default, DoNotify does nothing except call any event handler attached to the media player's OnNotify (see page 5817) event. DoNotify can be overridden to provide other responses in addition to the inherited event-handler call.

See Also

MMNotify (see page 5762)

OnNotify (see page 5817)

54.1.2.4.6 TMediaPlayer.Eject Method

Releases the loaded medium from the open multimedia device.

Pascal

```
procedure Eject;
```

C++

```
__fastcall Eject();
```

Description

Eject is called when the Eject button on the media player control is clicked at runtime. It ejects the loaded medium from the open multimedia device.

Upon completion, Eject stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Eject method has been completed. The Notify (see page 5794) property determines whether Eject generates an OnNotify (see page 5817) event.

C++ Examples:

```
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
  TimerOver: Word;

procedure TForm1.FormClick(Sender: TObject);
begin
```

```
    MediaPlayer1.DeviceType := dtCDAudio;
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;

  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    if TimeOver = 10 then
    begin
      MediaPlayer1.Eject;
      MediaPlayer1.Close;
      { disable the timer, as we are done }
      Timer1.Enabled := False;
    end
    else
      if (MediaPlayer1.Mode = mpOpen) then
       Inc(TimeOver);
  end;
```

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.7 TMediaPlayer.KeyDown Method

Provides responses when the user types arrow keys or the space bar.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown overrides the inherited method to implement the media player's response to certain keystrokes. It changes the current button when the user types the left or right arrow keys. When the user presses the space bar, KeyDown simulates a click on the current button.

See Also

EnabledButtons (see page 5786)

KeyDown

54.1.2.4.8 TMediaPlayer.Loaded Method

Initializes the media player after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded provides an opportunity for a component to initialize itself after all its parts have loaded from a stream. TMediaPlayer (see page 5749) overrides the inherited method to implement the AutoOpen (see page 5780) property.

See Also

Create (see page 5755)

AutoOpen (see page 5780)

54.1.2.4.9 TMediaPlayer.MMNotify Method

Responds to a notification that the MCI device has completed an operation.

Pascal

```
procedure MMNotify(var Message: TMessage); message MM_MCINOTIFY;
```

C++

```
__fastcall MMNotify(TMessage Message);
```

Description

MMNotify responds to a notification message from Windows that a multimedia command completed by updating button states if the AutoEnable (see page 5777) property is true, setting internal flags to reflect the outcome of the command, then calling the DoNotify (see page 5759) method to trigger the OnNotify (see page 5817) event and any other special responses.

See Also

AutoEnable (see page 5777)

DoNotify (see page 5759)

OnNotify (see page 5817)

54.1.2.4.10 TMediaPlayer.Next Method

Moves to the beginning of the next track of the currently loaded medium.

Pascal

```
procedure Next;
```

C++

```
__fastcall Next();
```

Description

If the current position is at the last track when Next is called, Next makes the current position the beginning of the last track. If the multimedia device doesn't use tracks, Next goes to the end of the medium. Next is called when the Next button on the media player control is clicked at runtime.

Upon completion, Next stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Next method has completed. The Notify (see page 5794) property determines whether Next generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.11 TMediaPlayer.Notification Method

Responds when components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

By default, components pass along the notification to their owned components, if any. Notification calls the inherited Notification and then, if the Display (see page 5783) window is deleted at runtime, it sets the video display back to original window.

See Also

Notification

Display (see page 5783)

54.1.2.4.12 TMediaPlayer.Open Method

Opens a multimedia device.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Use Open to open a multimedia device. The multimedia device type must be specified in the DeviceType (see page 5783) property before a device can be opened.

Upon completion, Open stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Open method is completed. The Notify (see page 5794) property determines whether Open generates an OnNotify (see page 5817) event.

C++ Examples:

```
//
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
```

```
//the buttons manually.
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
```

```
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
    end;
  end;

  OpenMediaDialog.Free;
end;

procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
```

```
  Button: TMPBtnType);
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.13 TMediaPlayer.Paint Method

Renders the image of the media player.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

When the media player component receives a WM_PAINT (see page 5198) message, like other custom controls, it responds by initializing the control's canvas and calling Paint. Paint draws the frame and all visible buttons.

When creating a descendant media player class, Paint can be overridden to draw the image of the corresponding media player object.

See Also

VisibleButtons (see page 5809)

54.1.2.4.14 TMediaPlayer.Pause Method

Toggles the open multimedia device on or off.

Pascal

```
procedure Pause;
```

C++

```
__fastcall Pause();
```

Description

If the device is already paused when Pause is called, the device resumes playing or recording by calling the Resume (see page 5771) method. Pause is called when the Pause button on the media player control is clicked at runtime.

Upon completion, Pause stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Pause method has completed. The Notify (see page 5794) property determines whether Pause generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Resume (see page 5771)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.15 TMediaPlayer.PauseOnly Method

Pauses the open multimedia device.

Pascal

```
procedure PauseOnly;
```

C++

```
__fastcall PauseOnly();
```

Description

If the device is already paused when PauseOnly is called, the device will remain paused.

Upon completion, PauseOnly stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the PauseOnly method has completed. The Notify (see page 5794) property determines whether PauseOnly generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

54.1.2.4.16 TMediaPlayer.Play Method

Plays the media loaded in the open multimedia device.

Pascal

```
procedure Play;
```

C++

```
__fastcall Play();
```

Description

Play is called when the Play button on the media player control is clicked at runtime.

Upon completion, Play stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Play method has completed. The Notify (see page 5794) property determines whether Play generates an OnNotify (see page 5817) event.

If the StartPos (see page 5804) property is set, playing starts at the position specified in StartPos (see page 5804). Otherwise, playing starts at the current position, specified in the Position (see page 5801) property. Similarly, if the EndPos (see page 5789) property is set, playing stops at the position specified in EndPos (see page 5789). Otherwise, playing stops at the end of the medium.

Whether the medium (specified in the Position (see page 5801) property) is rewound before playing starts depends on the AutoRewind (see page 5781) property.

C++ Examples:

```
//
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
//the buttons manually.
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
```

```
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
    end;
  end;

  OpenMediaDialog.Free;
end;
```

```
procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
  Button: TMPBtnType);
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

AutoRewind (see page 5781)

EndPos (see page 5789)

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Position (see page 5801)

Resume (see page 5771)

StartPos (see page 5804)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.17 TMediaPlayer.PostClick Method

Determines the action that takes place when an OnPostClick (see page 5818) event occurs.

Pascal

```
procedure PostClick(Button: TMPBtnType); dynamic;
```

C++

```
__fastcall PostClick(TMPBtnType Button);
```

Description

PostClick is the implementation method for the OnPostClick (see page 5818) event. By default, PostClick does nothing other than call any event handler attached to the media player's OnPostClick (see page 5818) event. PostClick can be overridden in

descendant types to customize responses to clicks.

See Also

OnPostClick (see page 5818)

54.1.2.4.18 TMediaPlayer.Previous Method

Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.

Pascal

```
procedure Previous;
```

C++

```
__fastcall Previous();
```

Description

If the position is at the first track or somewhere other than the beginning of a track when Previous was called, Previous sets the current position to the beginning of the current track. If the device doesn't use tracks, Previous sets the current position to the beginning of the medium, which is specified in the Start (see page 5804) property. Previous is called when the Previous button on the media player control is clicked at runtime.

Upon completion, Previous stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Previous method has completed. The Notify (see page 5794) property determines whether Previous generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Start (see page 5804)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.19 TMediaPlayer.Resume Method

Resumes playing or recording the currently paused multimedia device.

Pascal

```
procedure Resume;
```

C++

```
__fastcall Resume();
```

Description

Resume is called when the Pause (see page 5766) button on the media player control is clicked at runtime, when the device is paused.

Upon completion, Resume stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Resume method has completed. The Notify (see page 5794) property determines whether Resume generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

54.1.2.4.20 TMediaPlayer.Rewind Method

Sets the current position to the beginning of the the medium, which is stored in the Start (see page 5804) property.

Pascal

```
procedure Rewind;
```

C++

```
__fastcall Rewind();
```

Description

Upon completion, Rewind stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Rewind method has completed. The Notify (see page 5794) property determines whether Rewind generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Start (see page 5804)

Wait (see page 5809)

54.1.2.4.21 TMediaPlayer.Save Method

Saves the currently loaded medium to the file specified in the FileName (see page 5791) property.

Pascal

```
procedure Save;
```

C++

```
__fastcall Save();
```

Description

Save is ignored for devices that don't use media stored in files (videodiscs, for example).

Upon completion, Save stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Save method has completed. The Notify (see page 5794) property determines whether Save generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

FileName (see page 5791)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

54.1.2.4.22 TMediaPlayer.StartRecording Method

Begins recording from the current Position (see page 5801) or from the position specified in StartPos (see page 5804).

Pascal

```
procedure StartRecording;
```

C++

```
__fastcall StartRecording();
```

Description

StartRecording is called when the Record button on the media player control is clicked at runtime.

Upon completion, StartRecording stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the StartRecording method has completed. The Notify (see page 5794) property determines whether StartRecording generates an OnNotify (see page 5817) event.

By default, the Notify (see page 5794) property becomes true, and the Wait (see page 5809) property becomes false upon completion of the StartRecording method. However, if these properties have been set to specific values prior to calling StartRecording, they remain unchanged.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Notify (see page 5794)

OnNotify (see page 5817)

Position (see page 5801)

StartPos (see page 5804)

Wait (see page 5809)

54.1.2.4.23 TMediaPlayer.Step Method

Moves forward a number of frames (determined by the Frames (see page 5793) property) in the currently loaded medium.

Pascal

```
procedure Step;
```

C++

```
__fastcall Step();
```

Description

Step is called when the Step button on the media player control is clicked at runtime.

Upon completion, Step stores a numerical error code in the Error (see page 5789) property and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Step method has completed. The Notify (see page 5794) property determines whether Step generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

Frames (see page 5793)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.24 TMediaPlayer.Stop Method

Halts playing or recording.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Stop is called when the Stop button on the media player control is clicked at runtime.

Upon completion, Stop stores a numerical error code in the Error (see page 5789) property, and the corresponding error message in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before the Stop method has completed. The Notify (see page 5794) property determines whether Stop generates an OnNotify (see page 5817) event.

See Also

Error (see page 5789)

ErrorMessage (see page 5790)

FileName (see page 5791)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.4.25 TMediaPlayer.Updated Method

Signals that the media player has finished updating.

Pascal

```
procedure Updated; override;
```

C++

```
virtual __fastcall Updated();
```

Description

Updated calls the inherited Updated before it modifies the layout of the media player buttons and forces a repaint.

See Also

Updated

54.1.2.5 TMediaPlayer Properties

The properties of the TMediaPlayer class are listed here.

Public Properties

	Name	Description
R	Capabilities (see page 5781)	Determines the capabilities of the open multimedia device.
R	DeviceID (see page 5782)	Specifies the device ID for the currently open multimedia device.
	DisplayRect (see page 5784)	Specifies a rectangular area in the control specified by the Display (see page 5783) property that is used to display output from a multimedia device.
	EndPos (see page 5789)	Specifies the position within the currently loaded medium at which to stop playing or recording.
R	Error (see page 5789)	Specifies the MCI error code returned by the media control method.
R	ErrorMessage (see page 5790)	Describes the error code stored in the Error (see page 5789) property.
	Frames (see page 5793)	Specifies the number of frames moved forward or backward.
R	Length (see page 5794)	Specifies the length of the medium in the open multimedia device.
R	Mode (see page 5794)	Indicates the state of the currently open multimedia device.
	Notify (see page 5794)	Determines whether an OnNotify (see page 5817) event is generated.
R	NotifyValue (see page 5795)	Indicates the result of the last media control method that requested a notification.
	Position (see page 5801)	Specifies the current position within the currently loaded medium.
R	Start (see page 5804)	Specifies the starting position within the currently loaded medium.
	StartPos (see page 5804)	Specifies the position within the currently loaded medium from which to begin playing or recording.
	TimeFormat (see page 5805)	Determines the format used to specify position information.
R	TrackLength (see page 5806)	Reports the length of the track.
R	TrackPosition (see page 5806)	Reports the starting position of the track.
R	Tracks (see page 5807)	Specifies the number of playable tracks on the open multimedia device.
	Wait (see page 5809)	Determines whether a media control method returns control to the application only after it has been completed.

Published Properties

	Name	Description
	Anchors (see page 5776)	Specifies how the control is anchored to its parent.
	AutoEnable (see page 5777)	Determines whether the media player automatically enables and disables individual buttons in the component.
	AutoOpen (see page 5780)	Determines if the media player is opened automatically when the application is run.
	AutoRewind (see page 5781)	Determines if the media player control rewinds before playing or recording.
	ColoredButtons (see page 5781)	Determines which buttons on the media player control have color.
	Constraints (see page 5782)	Specifies the size constraints for the control.
	DeviceType (see page 5783)	Specifies a multimedia device type to open the media player.
	Display (see page 5783)	Specifies the display window for a multimedia device that uses a window for output.
	DoubleBuffered (see page 5784)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5786)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnabledButtons (see page 5786)	Determines which buttons on the media player are usable.
	FileName (see page 5791)	Specifies the opened or saved media file.
	ParentDoubleBuffered (see page 5795)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 5796)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5796)	Identifies the pop-up menu associated with the control.
	Shareable (see page 5802)	Determines whether more than one application can share a multimedia device.
	ShowHint (see page 5802)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5804)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5805)	Determines if the user can tab to a control.
	Visible (see page 5807)	Determines whether the component appears onscreen.
	VisibleButtons (see page 5809)	Indicates which of the buttons on the media player are visible.

54.1.2.5.1 TMediaPlayer.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

54.1.2.5.2 TMediaPlayer.AutoEnable Property

Determines whether the media player automatically enables and disables individual buttons in the component.

Pascal

```
property AutoEnable: Boolean;
```

C++

```
__property Boolean AutoEnable;
```

Description

Use AutoEnable to automatically enable or disable the control buttons on the media player. If AutoEnable is true, the media player automatically enables or disables its control buttons. The media player determines which buttons to enable or disable by the current mode specified in the Mode (see page 5794) property, and the current multimedia device type specified in the DeviceType (see page 5783) property.

AutoEnable overrides the EnabledButtons (see page 5786) property. The buttons enabled or disabled automatically by the media player supersede any buttons enabled or disabled with EnabledButtons (see page 5786).

If AutoEnable is false, the media player does not enable or disable buttons. The buttons must enabled or disabled with the EnabledButtons (see page 5786) property.

The following table shows whether buttons are automatically enabled or disabled for each device mode:

Button	Play (see page 5767)	Record	Pause (see page 5766)	Stop (see page 5774)	Not Open (see page 5763)
Back (see page 5757)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Eject (see page 5759)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Next (see page 5762)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Pause (see page 5766)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled	Disabled
Play (see page 5767)	Disabled	Disabled	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Prev	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Record	Disabled	Disabled	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Step (see page 5774)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled
Stop (see page 5774)	Enabled (see page 5786)	Enabled (see page 5786)	Disabled	Disabled	Disabled

C++ Examples:

```
//
```

```
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
//the buttons manually.
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
```

```
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
    end;
  end;

  OpenMediaDialog.Free;
end;

procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
```

```
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
  Button: TMPBtnType);
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

AutoButtonSet (see page 5756)

DeviceType (see page 5783)

EnabledButtons (see page 5786)

MMNotify (see page 5762)

Mode (see page 5794)

Adding Audio and/or Video Clips to an Application

54.1.2.5.3 TMediaPlayer.AutoOpen Property

Determines if the media player is opened automatically when the application is run.

Pascal

```
property AutoOpen: Boolean;
```

C++

```
__property Boolean AutoOpen;
```

Description

Use AutoOpen to have the media player automatically open at runtime.

If AutoOpen is true, the media player attempts to open the multimedia device specified by the DeviceType (see page 5783) property (or FileName (see page 5791) if DeviceType (see page 5783) is dtAutoSelect) when the form containing the media player component is created at runtime.

If AutoOpen is false, the device must be opened with a call to the Open (see page 5763) method.

If an error occurs when opening the device, an exception of type EMCIDeviceError (see page 5749) exception is raised that contains the error message. Upon completion, a numerical error code is stored in the Error (see page 5789) property, and the corresponding error message is stored in the ErrorMessage (see page 5790) property.

The Wait (see page 5809) property determines whether control is returned to the application before opening the multimedia device. The Notify (see page 5794) property determines whether opening the device generates an OnNotify (see page 5817) event.

See Also

DeviceType (see page 5783)

EMCIDeviceError (see page 5749)

Error (see page 5789)

ErrorMessage (see page 5790)

FileName (see page 5791)

Notify (see page 5794)

OnNotify (see page 5817)

Wait (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.5.4 TMediaPlayer.AutoRewind Property

Determines if the media player control rewinds before playing or recording.

Pascal

```
property AutoRewind: Boolean;
```

C++

```
__property Boolean AutoRewind;
```

Description

If AutoRewind is true and the current position is at the end of the medium, Play (see page 5767) or StartRecording (see page 5773) moves the current position to the beginning of the medium before playing or recording. If AutoRewind is false, the user must click the Prev button or write code to call Previous (see page 5771) to move to the beginning.

If values have been assigned to StartPos (see page 5804) or EndPos (see page 5789) or if the multimedia device uses tracks, AutoRewind has no effect on playing or recording. When calling Play (see page 5767) or StartRecording (see page 5773), the current position remains at the end of the medium.

See Also

Previous (see page 5771)

54.1.2.5.5 TMediaPlayer.Capabilities Property

Determines the capabilities of the open multimedia device.

Pascal

```
property Capabilities: TMPDevCapsSet;
```

C++

```
__property TMPDevCapsSet Capabilities;
```

Description

The various capabilities specified in Capabilities are determined when the device is opened with the Open (see page 5763) method.

Note: Currently, there is no way to check whether a device can step forward or backward. Capabilities includes mpCanStep only if the device type (specified in the DeviceType (see page 5783) property) is Animation, AVI Video, Digital Video, Overlay, or VCR.

See Also

DeviceType (see page 5783)

Open (see page 5763)

54.1.2.5.6 TMediaPlayer.ColoredButtons Property

Determines which buttons on the media player control have color.

Pascal

```
property ColoredButtons: TButtonSet;
```

C++

```
__property TButtonSet ColoredButtons;
```

Description

ColoredButtons controls which buttons on the media player are colored.

If a button is not colored with ColoredButtons, it appears in black-and-white when visible. All media player control buttons are colored by default.

See Also

EnabledButtons (see page 5786)

VisibleButtons (see page 5809)

54.1.2.5.7 TMediaPlayer.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

54.1.2.5.8 TMediaPlayer.DeviceID Property

Specifies the device ID for the currently open multimedia device.

Pascal

```
property DeviceID: Word;
```

C++

```
__property Word DeviceID;
```

Description

The value of DeviceID is determined when a device is opened with the Open (see page 5763) method. If no device is open, DeviceID is 0.

See Also

Open (see page 5763)

54.1.2.5.9 TMediaPlayer.DeviceType Property

Specifies a multimedia device type to open the media player.

Pascal

```
property DeviceType: TMPDeviceTypes;
```

C++

```
__property TMPDeviceTypes DeviceType;
```

Description

DeviceType specifies a multimedia device type to open with the Open (see page 5763) method. The default is dtAutoSelect.

If DeviceType is dtAutoSelect, the device type is determined by the file extension specified in the FileName (see page 5791) property. If no device type is associated with the extension, the correct device type must be explicitly specified by setting DeviceType to a value other than dtAutoSelect.

A multimedia device is typically associated with an appropriate file name extension when the device is installed. Associations are specified in the registry or SYSTEM.INI file. See the documentation for the specific device for instructions about how to associate file name extensions with the device.

See Also

FileName (see page 5791)

Open (see page 5763)

Adding Audio and/or Video Clips to an Application

54.1.2.5.10 TMediaPlayer.Display Property

Specifies the display window for a multimedia device that uses a window for output.

Pascal

```
property Display: TWinControl;
```

C++

```
__property TWinControl * Display;
```

Description

To use Display, assign the name of a windowed control such as a form or panel to Display to display output in that control.

The default value of Display is nil (Delphi) or NULL (C++), meaning that the device creates its own window to display output.

Also, by freeing the control assigned to Display after the device has been opened, video output will be in its own default window.

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

See Also

DisplayRect (see page 5784)

Adding Audio and/or Video Clips to an Application

54.1.2.5.11 TMediaPlayer.DisplayRect Property

Specifies a rectangular area in the control specified by the Display (see page 5783) property that is used to display output from a multimedia device.

Pascal

```
property DisplayRect: TRect;
```

C++

```
__property TRect DisplayRect;
```

Description

To display output in a specific rectangle area on a form assign a TRect record to DisplayRect. The Rect function can be used to create a TRect record.

Media that use a rectangle to display output usually perform best if the default DisplayRect size is used. To set DisplayRect to the default size, position the rectangle in the upper left corner and use 0, 0 for the lower right corner.

DisplayRect is ignored if Display (see page 5783) is nil (Delphi) or NULL (C++).

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

Note: DisplayRect can be set only after the media device is opened.

See Also

Display (see page 5783)

Open (see page 5763)

54.1.2.5.12 TMediaPlayer.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

54.1.2.5.13 TMediaPlayer.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

54.1.2.5.14 TMediaPlayer.EnabledButtons Property

Determines which buttons on the media player are usable.

Pascal

```
property EnabledButtons: TButtonSet;
```

C++

```
__property TButtonSet EnabledButtons;
```

Description

EnabledButtons controls which buttons on the media player are enabled and usable.

An enabled button is colored and usable. A disabled button is dimmed and not usable. If a button is not enabled with EnabledButtons, it is disabled. By default, all buttons are enabled.

If the AutoEnable (see page 5777) property is true, AutoEnable (see page 5777) supersedes EnabledButtons. The buttons automatically enabled or disabled by the media player override any buttons enabled or disabled with the EnabledButtons property.

C++ Examples:

```
//
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
//the buttons manually.
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
```

```
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
    end;
  end;

  OpenMediaDialog.Free;
end;
```

```
procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
  Button: TMPBtnType);
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

ColoredButtons (see page 5781)

VisibleButtons (see page 5809)

Adding Audio and/or Video Clips to an Application

54.1.2.5.15 TMediaPlayer.EndPos Property

Specifies the position within the currently loaded medium at which to stop playing or recording.

Pascal

```
property EndPos: Longint;
```

C++

```
__property Longint EndPos;
```

Description

EndPos is specified using the current time format, which is specified in the TimeFormat (see page 5805) property.

The EndPos property affects only the next Play (see page 5767) or StartRecording (see page 5773) method called after setting EndPos. Reset EndPos to affect any subsequent calls to Play (see page 5767) or StartRecording (see page 5773).

See Also

Play (see page 5767)

StartRecording (see page 5773)

TimeFormat (see page 5805)

54.1.2.5.16 TMediaPlayer.Error Property

Specifies the MCI error code returned by the media control method.

Pascal

```
property Error: Longint;
```

C++

```
__property Longint Error;
```

Description

Error specifies the MCI error code returned by the most recent media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), StartRecording (see page 5773), Resume (see page 5771), Rewind (see page 5772), Step (see page 5774), or Stop (see page 5774)). The ErrorMessage (see page 5790) property describes the Error.

The error code returned by media control methods are the same error code returned by the mciSendCommand function. The message describing the error code is stored in the ErrorMessage (see page 5790) property.

The value of Error is zero if the most recent media control method didn't cause an error. If a method results in an error, a value other than zero is stored in Error. If the error occurs during the opening of the device, an EMCIDeviceError (see page 5749) exception occurs.

See Also

EMCIDeviceError (see page 5749)

ErrorMessage (see page 5790)

Play (see page 5767)

StartRecording (see page 5773)

TimeFormat (see page 5805)

54.1.2.5.17 TMediaPlayer.ErrorMessage Property

Describes the error code stored in the Error (see page 5789) property.

Pascal

```
property ErrorMessage: string;
```

C++

```
__property UnicodeString ErrorMessage;
```

Description

ErrorMessage specifies the error message that describes the error code returned from the most recent media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), StartRecording (see page 5773), Resume (see page 5771), Rewind (see page 5772), Step (see page 5774), or Stop (see page 5774)).

The error code described by the message is stored in the Error (see page 5789) property.

See Also

Error (see page 5789)

54.1.2.5.18 TMediaPlayer.FileName Property

Specifies the opened or saved media file.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

FileName specifies the media file to be opened by the Open (see page 5763) method, or the file to save by the Save (see page 5772) method.

At design time, use a file open dialog box to specify the FileName property by clicking the ellipsis button (...) in the Object Inspector.

C++ Examples:

```
//
//This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
//for Open and Stop are present on the form.
//Note: AutoEnable must be set to false to be able to control
//the buttons manually.
//
void __fastcall TForm2::btOpenClick(TObject *Sender)
{
    TOpenDialog* OpenMediaDialog;

    OpenMediaDialog = new TOpenDialog(this);
    OpenMediaDialog->Filter = "All Video Files (*.avi)|*.avi";
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
    if (OpenMediaDialog->Execute())
    {
        /* Assign a file to the media player */
        MediaPlayer1->FileName = OpenMediaDialog->FileName;

        /* Check if the file exists and is not a directory */
        if ((FileExists(OpenMediaDialog->FileName)) &&
           (!DirectoryExists(OpenMediaDialog->FileName)))
        {
            /* Open the files */
            MediaPlayer1->Wait = true;
            MediaPlayer1->Open();
            MediaPlayer1->Play();

            /* Override automatic button controlling */
            MediaPlayer1->EnabledButtons =
                TButtonSet() <<
                TMPBtnType::btPause <<
                TMPBtnType::btStop <<
                TMPBtnType::btPlay;


            /* Enable  the stop button */
            btStop->Enabled = true;
            btOpen->Enabled = false;
        }
    }

    delete OpenMediaDialog;
```

```
}

void __fastcall TForm2::btStopClick(TObject *Sender)
{
    /* Stop and close the media */
    MediaPlayer1->Stop();
    MediaPlayer1->Close();

    MediaPlayer1->EnabledButtons = TButtonSet();

    /* Enable open button again */
    btOpen->Enabled = true;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Disable all buttons */
    MediaPlayer1->AutoEnable = false;
    MediaPlayer1->EnabledButtons = TButtonSet();
}

void __fastcall TForm2::MediaPlayer1PostClick(TObject *Sender,
          TMPBtnType Button)
{
    if (Button == TMPBtnType::btStop)
        btStop->Click();
}
```

Delphi Examples:

```
{
This example demonstrates the use of TMediaPlayer component. Assuming 2 buttons
for Open and Stop are present on the form.
Note: AutoEnable must be set to false to be able to control
the buttons manually.
}

procedure TForm2.btOpenClick(Sender: TObject);
var
  OpenMediaDialog : TOpenDialog;
begin
  OpenMediaDialog := TOpenDialog.Create(Self);
  OpenMediaDialog.Filter := 'All Video Files (*.avi)|*.avi';
  // There are avi files in
  // C:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\DelphiWin32\VCLWin32\CoolStuff
  if OpenMediaDialog.Execute() then
  begin
    { Assign a file to the media player }
    MediaPlayer1.FileName := OpenMediaDialog.FileName;

    { Check if the file exists and is not a directory }
    if (FileExists(OpenMediaDialog.FileName)) and
       (not DirectoryExists(OpenMediaDialog.FileName)) then
    begin
      { Open the files }
      MediaPlayer1.Wait := true;
      MediaPlayer1.Open;
      MediaPlayer1.Play;

      { Override automatic button controlling }
      MediaPlayer1.EnabledButtons :=
        [TMPBtnType.btPause, TMPBtnType.btStop, TMPBtnType.btPlay];

      { Enable  the stop button }
      btStop.Enabled := true;
      btOpen.Enabled := false;
```

```
    end;
  end;

  OpenMediaDialog.Free;
end;

procedure TForm2.btStopClick(Sender: TObject);
begin
  { Stop and close the media }
  MediaPlayer1.Stop;
  MediaPlayer1.Close;

  MediaPlayer1.EnabledButtons := [];

  { Enable open button again }
  btOpen.Enabled := true;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Disable all buttons }
  MediaPlayer1.AutoEnable := false;
  MediaPlayer1.EnabledButtons := [];
end;

procedure TForm2.MediaPlayer1PostClick(Sender: TObject;
  Button: TMPBtnType);
begin
  if Button = TMPBtnType.btStop then
     btStop.Click;
end;
```

See Also

Open (see page 5763)

Save (see page 5772)

Adding Audio and/or Video Clips to an Application

54.1.2.5.19 TMediaPlayer.Frames Property

Specifies the number of frames moved forward or backward.

Pascal

```
property Frames: Longint;
```

C++

```
__property Longint Frames;
```

Description

Frames specifies the number of frames the Step (see page 5774) method steps forward or the Back (see page 5757) method steps backward. Frames defaults to ten percent of the length of the currently loaded medium, which is specified by the Length (see page 5794) property.

The definition of a frame varies by multimedia device. For display media, a frame is one still image.

See Also

Back (see page 5757)

Length (see page 5794)

Step (see page 5774)

54.1.2.5.20 TMediaPlayer.Length Property

Specifies the length of the medium in the open multimedia device.

Pascal

```
property Length: Longint;
```

C++

```
__property Longint Length;
```

Description

Length is specified using the current time format, which is specified by the TimeFormat (see page 5805) property.

See Also

TimeFormat (see page 5805)

54.1.2.5.21 TMediaPlayer.Mode Property

Indicates the state of the currently open multimedia device.

Pascal

```
property Mode: TMPModes;
```

C++

```
__property TMPModes Mode;
```

Description

Mode specifies the current state or mode of the currently open multimedia device.

The TMPModes (see page 5823) type defines the modes for a multimedia device used with a TMediaPlayer (see page 5749).

See Also

OnNotify (see page 5817)

54.1.2.5.22 TMediaPlayer.Notify Property

Determines whether an OnNotify (see page 5817) event is generated.

Pascal

```
property Notify: Boolean;
```

C++

```
__property Boolean Notify;
```

Description

Notify determines whether the next call to a media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), StartRecording (see page 5773), Resume (see page 5771), Rewind (see page 5772), Step (see page 5774), or Stop (see page 5774)) generates an OnNotify (see page 5817) event when the method has completed.

If Notify is true, the next media control method generates OnNotify (see page 5817) event upon completion and stores the notification message in the NotifyValue (see page 5795) property. If Notify is false, the method does not generate an OnNotify

(see page 5817) event and NotifyValue (see page 5795) remains unchanged.

Notify affects only the next call to a media control method. After an OnNotify (see page 5817) event, Notify must be reset to affect any subsequent media control methods.

By default, Play (see page 5767) and StartRecording (see page 5773) function as if Notify is true. Set Notify to false before calling Play (see page 5767) or StartRecording (see page 5773) to prevent an OnNotify (see page 5817) event from being generated when playing or recording has finished. By default, all other media control methods function as if Notify is false.

Tip: Set Notify to true if the next media control is expected to take a long time, so the application is notified when the media control method has completed. If Notify is set to true, it is recommended to set Wait (see page 5809) to false so that control returns to the application before the media control method is finished.

Note: When trying to resume a device that doesn't support Resume (see page 5771), the device is resumed as if the Play (see page 5767) method was called. If Notify was assigned true before calling Resume (see page 5771) (or any other media control method), Notify doesn't affect the call to Resume (see page 5771). Resume (see page 5771) does not generate an OnNotify (see page 5817) event upon completion, and NotifyValue (see page 5795) remains unchanged.

See Also

NotifyValue (see page 5795)

OnNotify (see page 5817)

Resume (see page 5771)

Wait (see page 5809)

54.1.2.5.23 TMediaPlayer.NotifyValue Property

Indicates the result of the last media control method that requested a notification.

Pascal

```
property NotifyValue: TMPNotifyValues;
```

C++

```
__property TMPNotifyValues NotifyValue;
```

Description

NotifyValue reports the result of the last media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), StartRecording (see page 5773), Resume (see page 5771), Rewind (see page 5772), Step (see page 5774), or Stop (see page 5774)) that requested a notification.

To request notification, set Notify (see page 5794) to true before calling a media control method.

See Also

OnNotify (see page 5817)

Notify (see page 5794)

54.1.2.5.24 TMediaPlayer.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

54.1.2.5.25 TMediaPlayer.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

54.1.2.5.26 TMediaPlayer.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup

property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

54.1.2.5.27 TMediaPlayer.Position Property

Specifies the current position within the currently loaded medium.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

The value of Position is specified according to the current time format, which is specified in the TimeFormat (see page 5805) property.

Position defaults to the beginning of the medium. If the medium supports multiple tracks, Position defaults to the beginning of the first track.

See Also

TimeFormat (see page 5805)

54.1.2.5.28 TMediaPlayer.Shareable Property

Determines whether more than one application can share a multimedia device.

Pascal

```
property Shareable: Boolean;
```

C++

```
__property Boolean Shareable;
```

Description

If Shareable is false, no other components or applications can access the device. If Shareable is true, more than one component or application can access the device. Shareable defaults to false.

Be sure to set Shareable before opening a device.

Some devices are not shareable. If Shareable is set to true for a device that isn't shareable, the Open (see page 5763) method fails and the error code is returned to the Error (see page 5789) property.

See Also

Error (see page 5789)

Open (see page 5763)

54.1.2.5.29 TMediaPlayer.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
```

```
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

54.1.2.5.30 TMediaPlayer.Start Property

Specifies the starting position within the currently loaded medium.

Pascal

```
property Start: Longint;
```

C++

```
__property Longint Start;
```

Description

Start is the beginning of the medium for devices that don't use tracks, or the beginning of the first track for devices that use tracks. Start is defined when a multimedia device is opened with the Open (see page 5763) method. Start is specified according to the current time format, which is stored in the TimeFormat (see page 5805) property. Start is read-only at runtime and is unavailable at design time.

See Also

Open (see page 5763)

TimeFormat (see page 5805)

54.1.2.5.31 TMediaPlayer.StartPos Property

Specifies the position within the currently loaded medium from which to begin playing or recording.

Pascal

```
property StartPos: Longint;
```

C++

```
__property Longint StartPos;
```

Description

StartPos is specified using the current time format, which is specified in the TimeFormat (see page 5805) property.

The StartPos property affects only the next Play (see page 5767) or StartRecording (see page 5773) method called after setting StartPos. Reset must be set to StartPos to affect any subsequent calls to Play (see page 5767) or StartRecording (see page 5773).

StartPos does not affect the current position of the medium (specified in the Position (see page 5801) property) until the next Play (see page 5767) or StartRecording (see page 5773) method is called.

See Also

Play (see page 5767)

StartRecording (see page 5773)

TimeFormat (see page 5805)

54.1.2.5.32 TMediaPlayer.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

54.1.2.5.33 TMediaPlayer.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

54.1.2.5.34 TMediaPlayer.TimeFormat Property

Determines the format used to specify position information.

Pascal

```
property TimeFormat: TMPTimeFormats;
```

C++

```
__property TMPTimeFormats TimeFormat;
```

Description

TimeFormat determines how the StartPos (see page 5804), Length (see page 5794), Position (see page 5801), Start (see page 5804), and EndPos (see page 5789) properties are interpreted. For example, if Position (see page 5801) is 180 and TimeFormat is tfMilliseconds, the current position is 180 milliseconds into the medium. If Position (see page 5801) is 180 and TimeFormat is tfMSF, the current position is 180 minutes into the medium.

Not all formats are supported by every device. When trying to set an unsupported format, the assignment is ignored.

The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really one integer, but single bytes of information packed in the long integer.

See Also

EndPos (see page 5789)

Length (see page 5794)

Position (see page 5801)

Start (see page 5804)

StartPos (see page 5804)

54.1.2.5.35 TMediaPlayer.TrackLength Property

Reports the length of the track.

Pascal

```
property TrackLength [TrackNum: Integer]: Longint;
```

C++

```
__property Longint TrackLength[int TrackNum];
```

Description

TrackLength reports the length of the track specified by the TrackNum index.

The value of TrackLength is specified according to the current time format, which is specified in the TimeFormat (see page 5805) property.

See Also

TimeFormat (see page 5805)

Tracks (see page 5807)

54.1.2.5.36 TMediaPlayer.TrackPosition Property

Reports the starting position of the track.

Pascal

```
property TrackPosition [TrackNum: Integer]: Longint;
```

C++

```
__property Longint TrackPosition[int TrackNum];
```

Description

TrackPosition reports the starting position of the track specified by the TrackNum index.

The value of TrackPosition is specified according to the current time format, which is specified in the TimeFormat (see page 5805) property.

See Also

TimeFormat (see page 5805)

Tracks (see page 5807)

54.1.2.5.37 TMediaPlayer.Tracks Property

Specifies the number of playable tracks on the open multimedia device.

Pascal

```
property Tracks: Longint;
```

C++

```
__property Longint Tracks;
```

Description

Tracks indicates how many of playable tracks are available on the open multimedia device.

Tracks is undefined for devices that don't use tracks.

See Also

TrackLength (see page 5806)

TrackPosition (see page 5806)

Adding Audio and/or Video Clips to an Application

54.1.2.5.38 TMediaPlayer.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

54.1.2.5.39 TMediaPlayer.VisibleButtons Property

Indicates which of the buttons on the media player are visible.

Pascal

```
property VisibleButtons: TButtonSet;
```

C++

```
__property TButtonSet VisibleButtons;
```

Description

VisibleButtons determines which buttons on the media player that are visible. If a button is not made visible with VisibleButtons, it does not appear on the media player control. By default, all buttons are visible when a media player component is added to a form.

See Also

ColoredButtons (see page 5781)

EnabledButtons (see page 5786)

54.1.2.5.40 TMediaPlayer.Wait Property

Determines whether a media control method returns control to the application only after it has been completed.

Pascal

```
property Wait: Boolean;
```

C++

```
__property Boolean Wait;
```

Description

Wait determines whether a media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), StartRecording (see page 5773), Resume (see page 5771), Rewind (see page 5772), Step (see page 5774), or Stop (see page 5774)) returns control to the application only after it has been completed.

Wait is unavailable at design time.

If Wait is true, the media player component waits until the next media control method has completed before returning control to the application. If Wait is false, the application won't wait for the next media control method to finish before continuing.

Wait affects only the next media control method called after setting Wait. Wait must be reset to affect any subsequent call to a media control method.

By default, Play (see page 5767) and StartRecording (see page 5773) function as if Wait is false. Wait must be set to true before calling Play (see page 5767) or StartRecording (see page 5773) to prevent control from returning to the application before playing or recording has finished. By default, all other media control methods function as if Wait is true.

Note: Wait is usually set to false only if the next media control is expected to take a long time, so that the application can execute other code before the media control method has completed. If Wait is set to false, you should set Notify (see page 5794) to true so the application is notified when the media control method completes.

See Also

Notify (see page 5794)

54.1.2.6 TMediaPlayer Events

The events of the TMediaPlayer class are listed here.

Published Events

	Name	Description
	OnClick (see page 5810)	Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.
	OnContextPopup (see page 5811)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 5811)	Occurs when a control receives the input focus.
	OnExit (see page 5814)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 5817)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5817)	Occurs when the user moves the mouse outside of a control.
	OnNotify (see page 5817)	Occurs upon the completion of a media control method.
	OnPostClick (see page 5818)	Occurs after the OnClick (see page 5810) event handler is called.

54.1.2.6.1 TMediaPlayer.OnClick Event

Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.

Pascal

```
property OnClick: EMPNotify;
```

C++

```
__property EMPNotify OnClick;
```

Description

When the media player control has focus and the Spacebar is pressed, the user can use the Left Arrow or Right Arrow keys to select which control button to click.

The EMPNotify (see page 5827) type is a method pointer that is called when an OnClick event for a TMediaPlayer (see page 5749) components occurs. The Button argument can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

The default value of the event handler's DoDefault argument is true. If DoDefault is true, the media player control calls the method that corresponds to the clicked button. For example, if the user clicks the Play (see page 5767) button (btPlay), the Play (see page 5767) method is called.

If DoDefault is false, the user must write code that executes when a media player control button is clicked in the OnClick event handler. The following table lists the default methods corresponding to the media player control buttons:

Control button	Button value	Method called
Play (see page 5767)	btPlay	Play (see page 5767)
Record	btRecord	StartRecording (see page 5773)
Stop (see page 5774)	btStop	Stop (see page 5774)
Next (see page 5762)	btNext	Next (see page 5762)

Prev	btPrev	Previous (see page 5771)
Step (see page 5774)	btStep	Step (see page 5774)
Back (see page 5757)	btBack	Back (see page 5757)
Pause (see page 5766)	btPause	Pause (see page 5766)
Eject (see page 5759)	btEject	Eject (see page 5759)

See Also

Click (see page 5757)

54.1.2.6.2 TMediaPlayer.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

54.1.2.6.3 TMediaPlayer.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

54.1.2.6.4 TMediaPlayer.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
```

```
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;
```

```
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

54.1.2.6.5 TMediaPlayer.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

54.1.2.6.6 TMediaPlayer.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

54.1.2.6.7 TMediaPlayer.OnNotify Event

Occurs upon the completion of a media control method.

Pascal

```
property OnNotify: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnNotify;
```

Description

OnNotify occurs upon the completion of a media control method (Back (see page 5757), Close (see page 5758), Eject (see page 5759), Next (see page 5762), Open (see page 5763), Pause (see page 5766), PauseOnly (see page 5767), Play (see page 5767), Previous (see page 5771), Resume (see page 5771), Rewind (see page 5772), StartRecording (see page 5773), Step (see page 5774), or Stop (see page 5774)) when the Notify (see page 5794) property is set to true before the call to the media control method. After an OnNotify event, the Notify (see page 5794) property must be reset to

true for the next OnNotify event to occur.

See Also

DoNotify (see page 5759)

MMNotify (see page 5762)

Notify (see page 5794)

54.1.2.6.8 TMediaPlayer.OnPostClick Event

Occurs after the OnClick (see page 5810) event handler is called.

Pascal

```
property OnPostClick: EMPPostNotify;
```

C++

```
__property EMPPostNotify OnPostClick;
```

Description

OnPostClick is generated after the code of the OnClick (see page 5810) event handler has been called. If Wait (see page 5809) is true when the media player was clicked, OnPostClick won't be called until the completion of the OnClick (see page 5810) code. If Wait (see page 5809) is false, control can return to the application before completion of the OnClick (see page 5810) code; therefore, the OnPostClick event may occur before the actions initiated by the OnClick (see page 5810) event have completed.

For example, if the user clicks the Play (see page 5767) button and the DoDefault parameter of the OnClick (see page 5810) event handler for the media player is true, the media is played. If the media is long enough, it will still be playing when the OnPostClick event is generated if Wait (see page 5809) is true. If Wait (see page 5809) is false, however, OnPostClick won't occur until the media has finished playing.

See Also

OnClick (see page 5810)

PostClick (see page 5770)

Wait (see page 5809)

54.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMPBtnType (see page 5819)	TMPBtnType defines the possible buttons in a TMediaPlayer (see page 5749) component and TButtonSet (see page 5828) represents a set of those buttons.
	TMPDevCaps (see page 5821)	TMPDevCaps defines the capabilities of a TMediaPlayer (see page 5749) component and TMPDevCapsSet (see page 5828) represents a set of those capabilities.
	TMPDeviceTypes (see page 5821)	TMPDeviceTypes identifies a multimedia device.
	TMPGlyph (see page 5823)	This is record MPlayer.TMPGlyph.
	TMPModes (see page 5823)	TMPModes identifies the state of a TMediaPlayer (see page 5749) component.
	TMPNotifyValues (see page 5824)	TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 5749) component.
	TMPTimeFormats (see page 5825)	TMPTimeFormats indicates the format a media player uses to indicate position.

Records

	Name	Description
	TMPButton (see page 5820)	This is record MPlayer.TMPButton.

54.2.1 MPlayer.TMPBtnType Enumeration

TMPBtnType defines the possible buttons in a TMediaPlayer (see page 5749) component and TButtonSet (see page 5828) represents a set of those buttons.

Pascal

```
TMPBtnType = (
  btPlay,
  btPause,
  btStop,
  btNext,
  btPrev,
  btStep,
  btBack,
  btRecord,
  btEject
);
```

C++

```
enum TMPBtnType {
  btPlay,
  btPause,
  btStop,
  btNext,
  btPrev,
  btStep,
  btBack,
  btRecord,
  btEject
};
```

File

MPlayer

Description

TMPBtnType represents the type of a media player button. TButtonSet (see page 5828) is a set of button types for the media player.

Each button type has an associated action on the media player. The following table lists the button types and their associated actions:

Value	Action
btPlay	Plays the media player.
btRecord	Starts recording.
btStop	Stops playing or recording.
btNext	Skips to the next track, or to the end if the medium doesn't use tracks.
btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks.
btStep	Moves forward a number of frames.
btBack	Moves backward a number of frames.
btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
btEject	Ejects the medium.

See Also

TBitBtn

TSpeedButton

54.2.2 MPlayer.TMPButton Record

Pascal

```
TMPButton = record
  Visible: Boolean;
  Enabled: Boolean;
  Colored: Boolean;
  Auto: Boolean;
  Bitmaps: array[TMPGlyph] of TBitmap;
end;
```

C++

```
struct TMPButton {
  Boolean Visible;
  Boolean Enabled;
  Boolean Colored;
  Boolean Auto;
  array[TMPGlyph] of TBitmap Bitmaps;
};
```

File

MPlayer

Description

This is record MPlayer.TMPButton.

54.2.3 MPlayer.TMPDevCaps Enumeration

TMPDevCaps defines the capabilities of a TMediaPlayer (see page 5749) component and TMPDevCapsSet (see page 5828) represents a set of those capabilities.

Pascal

```
TMPDevCaps = (
  mpCanStep,
  mpCanEject,
  mpCanPlay,
  mpCanRecord,
  mpUsesWindow
);
```

C++

```
enum TMPDevCaps {
  mpCanStep,
  mpCanEject,
  mpCanPlay,
  mpCanRecord,
  mpUsesWindow
};
```

File

MPlayer

Description

TMPDevCaps represents the capabilities of a media player control. TMPDevCapsSet (see page 5828) is a set of TMPDevCaps values.

The following table lists the possible TMPDevCaps values:

Value	Capability
mpCanEject	Can eject media
mpCanPlay	Can play media
mpCanRecord	Can record media
mpCanStep	Can step forward or backward within media
mpUsesWindow	Uses a window for displaying output

See Also

TMediaPlayer

54.2.4 MPlayer.TMPDeviceTypes Enumeration

TMPDeviceTypes identifies a multimedia device.

Pascal

```
TMPDeviceTypes = (
  dtAutoSelect,
  dtAVIVideo,
  dtCDAudio,
  dtDAT,
  dtDigitalVideo,
  dtMMMovie,
  dtOther,
  dtOverlay,
  dtScanner,
  dtSequencer,
  dtVCR,
  dtVideodisc,
  dtWaveAudio
);
```

C++

```
enum TMPDeviceTypes {
  dtAutoSelect,
  dtAVIVideo,
  dtCDAudio,
  dtDAT,
  dtDigitalVideo,
  dtMMMovie,
  dtOther,
  dtOverlay,
  dtScanner,
  dtSequencer,
  dtVCR,
  dtVideodisc,
  dtWaveAudio
};
```

File

MPlayer

Description

TMPDeviceTypes type identifies the multimedia device types that can be opened by a TMediaPlayer (see page 5749) component.

The following table lists the possible values:

Value	Meaning
dtAutoSelect	the device type is determined by the file extension.
dtAVIVideo	AVI clip
dtCDAudio	Audio CD recording
dtDAT	DAT file
dtDigitalVideo	Digital video recording
dtMMMovie	MM recording
dtOther	media type not identified by other categories
dtOverlay	Overlay
dtScanner	scanner
dtSequencer	sequencer
dtVCR	VCR

dtVideodisc	Videodisc
dtWaveAudio	.wav file

See Also

TMediaPlayer

54.2.5 MPlayer.TMPGlyph Enumeration

Pascal

```
TMPGlyph = (
  mgEnabled,
  mgDisabled,
  mgColored
);
```

C++

```
enum TMPGlyph {
  mgEnabled,
  mgDisabled,
  mgColored
};
```

File

MPlayer

Description

This is record MPlayer.TMPGlyph.

54.2.6 MPlayer.TMPModes Enumeration

TMPModes identifies the state of a TMediaPlayer (see page 5749) component.

Pascal

```
TMPModes = (
  mpNotReady,
  mpStopped,
  mpPlaying,
  mpRecording,
  mpSeeking,
  mpPaused,
  mpOpen
);
```

C++

```
enum TMPModes {
  mpNotReady,
  mpStopped,
  mpPlaying,
  mpRecording,
  mpSeeking,
  mpPaused,
  mpOpen
```

```
};
```

File

MPlayer

Description

TMPModes type identifies the state of a media player. The following table lists the possible values:

Value	Mode
mpNotReady	Not ready
mpStopped	Stopped
mpPlaying	Playing
mpRecording	Recording
mpSeeking	Seeking
mpPaused	Paused
mpOpen	Open

See Also

TMediaPlayer

54.2.7 MPlayer.TMPNotifyValues Enumeration

TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 5749) component.

Pascal

```
TMPNotifyValues = (
  nvSuccessful,
  nvSuperseded,
  nvAborted,
  nvFailure
);
```

C++

```
enum TMPNotifyValues {
  nvSuccessful,
  nvSuperseded,
  nvAborted,
  nvFailure
};
```

File

MPlayer

Description

The TMPNotifyValues type defines the notification values for a multimedia device used with a TMediaPlayer (see page 5749). The possible values for the TMPNotifyValues type are as follows.

Value	Meaning
nvSuccessful	Command completed successfully
nvSuperseded	Command was superseded by another command

nvAborted	Command was aborted by the user
nvFailure	Command failed

See Also

TMediaPlayer

54.2.8 MPlayer.TMPTimeFormats Enumeration

TMPTimeFormats indicates the format a media player uses to indicate position.

Pascal

```
TMPTimeFormats = (
  tfMilliseconds,
  tfHMS,
  tfMSF,
  tfFrames,
  tfSMPTE24,
  tfSMPTE25,
  tfSMPTE30,
  tfSMPTE30Drop,
  tfBytes,
  tfSamples,
  tfTMSF
);
```

C++

```
enum TMPTimeFormats {
  tfMilliseconds,
  tfHMS,
  tfMSF,
  tfFrames,
  tfSMPTE24,
  tfSMPTE25,
  tfSMPTE30,
  tfSMPTE30Drop,
  tfBytes,
  tfSamples,
  tfTMSF
};
```

File

MPlayer

Description

The TMPTimeFormats type defines the time formats for a multimedia device used with a TMediaPlayer (see page 5749). The following table lists the possible values for the TMPTimeFormats type:

Value	Time format
tfMilliseconds	Milliseconds are stored as a 4-byte integer variable.
tfHMS	Hours, minutes, and seconds packed into a 4-byte integer. From least significant to most significant byte, the data values are
Hours (least significant byte)	
Minutes	

Seconds	
Unused (most significant byte)	
tfMSF	Minutes, seconds, and frames packed into a 4-byte integer. From least significant to most significant byte, the data values are
Minutes (least significant byte)	
Seconds	
Frames	
Unused (most significant byte)	
tfFrames	Frames are stored as a 4-byte integer variable.
tfSMPTE24	24-frame SMPTE packs values in a 4-byte variable. From least significant to most significant byte, the data values are
Hours (least significant byte)	
Minutes	
Seconds	
Frames (most significant byte)	
SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second.	
tfSMPTE25	25-frame SMPTE packs data into a 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30	30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30Drop	30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfBytes	Bytes are stored as a 4-byte integer variable.
tfSamples	Samples are stored as a 4-byte integer variable.
tfTMSF	Tracks, minutes, seconds, and frames are packed in the 4-byte variable. From least significant to most significant byte, the data values are
Tracks (least significant byte)	
Minutes	
Seconds	
Frames (most significant byte)	

Note that MCI uses continuous track numbering.

Note: Functions provided with MCI to help decoding the 4-byte integer specified in a given time format are documented under MCI Macros for Encoding and Decoding Time Data in the MMSYSTEM.HLP Help file.

See Also

TMediaPlayer

54.3 Types

The following table lists types in this documentation.

Types

Name	Description
EMPNotify (see page 5827)	EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 5749) component.
EMPPostNotify (see page 5827)	EMPPostNotify is the type of a media player's OnPostClick event handler.
TButtonSet (see page 5828)	TButtonSet is a set of TMPBtnType (see page 5819) values.
TMediaDevice (see page 5828)	This is type MPlayer.TMediaDevice.
TMPDevCapsSet (see page 5828)	This is type MPlayer.TMPDevCapsSet.

54.3.1 MPlayer.EMPNotify Type

EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 5749) component.

Pascal

```
EMPNotify = procedure (Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean) of object;
```

C++

```
(Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean) ( EMPNotify)();
```

File

MPlayer

Description

Sender is the TMediaPlayer (see page 5749) component whose button was clicked.

Button indicates the button that was clicked. It can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

DoDefault returns an indication of whether the media player should make its default response after the event handler exits.

See Also

TMediaPlayer

54.3.2 MPlayer.EMPPostNotify Type

EMPPostNotify is the type of a media player's OnPostClick event handler.

Pascal

```
EMPPostNotify = procedure (Sender: TObject; Button: TMPBtnType) of object;
```

C++

```
(Sender: TObject; Button: TMPBtnType) ( EMPPostNotify)();
```

File

MPlayer

Description

Sender is the TMediaPlayer (see page 5749) component whose button was clicked.

Button indicates the button that was clicked. It can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

See Also

TMediaPlayer (see page 5749)

54.3.3 MPlayer.TButtonSet Type

TButtonSet is a set of TMPBtnType (see page 5819) values.

Pascal

```
TButtonSet = set of TMPBtnType;
```

C++

```
TMPBtnType TButtonSet;
```

File

MPlayer

Description

TButtonSet represents a set of zero or more of the buttons that can appear on a TMPDeviceType control.

54.3.4 MPlayer.TMediaDevice Type

Pascal

```
TMediaDevice = Word;
```

C++

```
Word TMediaDevice;
```

File

MPlayer

Description

This is type MPlayer.TMediaDevice.

54.3.5 MPlayer.TMPDevCapsSet Type

Pascal

```
TMPDevCapsSet = set of TMPDevCaps;
```

C++

```
TMPDevCaps TMPDevCapsSet;
```

File

MPlayer

Description

This is type MPlayer.TMPDevCapsSet.

55 Mtsobj Namespace

This is namespace Mtsobj.

Topics

Name	Description
Classes (see page 5832)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsAutoObject (see page 5832)	TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

55.1 Classes

The following table lists classes in this documentation.

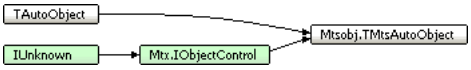
Classes

	Name	Description
	TMtsAutoObject (see page 5832)	TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

55.1.1 TMtsAutoObject Class

TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

Class Hierarchy



Pascal

```
TMtsAutoObject = class(TAutoObject, IObjectControl);
```

C++

```
class TMtsAutoObject : public TAutoObject, public IObjectControl;
```

File

Mtsobj

Description

Use a TMtsAutoObject object as the central repository for all objects in an MTS-enabled application server.

MTS Automation objects are dual-interface Automation servers that implement the IObjectControl (see page 5866) interface. MTS Automation objects are like any Automation server with additional features that encapsulate MTS services such as transaction support and security.

See Also

TMTSDataModule

Writing Applications Using COM

55.1.1.1 TMtsAutoObject Members

The following tables list the members exposed by TMtsAutoObject.

Protected Methods

	Name	Description
	Activate (see page 5833)	This is Activate, a member of class TMtsAutoObject.
	CanBePooled (see page 5834)	This is CanBePooled, a member of class TMtsAutoObject.
	Deactivate (see page 5834)	This is Deactivate, a member of class TMtsAutoObject.

	OnActivate (see page 5836)	This is OnActivate, a member of class TMtsAutoObject.
	OnDeactivate (see page 5837)	This is OnDeactivate, a member of class TMtsAutoObject.

Protected Properties

	Name	Description
	ObjectContext (see page 5838)	This is ObjectContext, a member of class TMtsAutoObject.

Public Methods

	Name	Description
	DisableCommit (see page 5834)	Prevents clients from completing transactions.
	EnableCommit (see page 5835)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 5835)	Indicates whether the current client matches a particular security role.
	IsInTransaction (see page 5836)	Indicates whether the MTS Automation object is running in a transaction.
	IsSecurityEnabled (see page 5836)	This is IsSecurityEnabled, a member of class TMtsAutoObject.
	SetAbort (see page 5837)	Allows the MTS Automation object to be deactivated and aborts any current transactions.
	SetComplete (see page 5837)	Indicates that the MTS Automation object has successfully completed its current task.

Public Properties

	Name	Description
	Pooled (see page 5838)	Indicates whether the object can be pooled when installed under COM+.

55.1.1.2 TMtsAutoObject Methods

The methods of the TMtsAutoObject class are listed here.

Protected Methods

	Name	Description
	Activate (see page 5833)	This is Activate, a member of class TMtsAutoObject.
	CanBePooled (see page 5834)	This is CanBePooled, a member of class TMtsAutoObject.
	Deactivate (see page 5834)	This is Deactivate, a member of class TMtsAutoObject.
	OnActivate (see page 5836)	This is OnActivate, a member of class TMtsAutoObject.
	OnDeactivate (see page 5837)	This is OnDeactivate, a member of class TMtsAutoObject.

Public Methods

	Name	Description
	DisableCommit (see page 5834)	Prevents clients from completing transactions.
	EnableCommit (see page 5835)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 5835)	Indicates whether the current client matches a particular security role.
	IsInTransaction (see page 5836)	Indicates whether the MTS Automation object is running in a transaction.
	IsSecurityEnabled (see page 5836)	This is IsSecurityEnabled, a member of class TMtsAutoObject.
	SetAbort (see page 5837)	Allows the MTS Automation object to be deactivated and aborts any current transactions.
	SetComplete (see page 5837)	Indicates that the MTS Automation object has successfully completed its current task.

55.1.1.2.1 TMtsAutoObject.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TMtsAutoObject.

55.1.1.2.2 TMtsAutoObject.CanBePooled Method

Pascal

```
function CanBePooled: Bool; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class TMtsAutoObject.

55.1.1.2.3 TMtsAutoObject.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class TMtsAutoObject.

55.1.1.2.4 TMtsAutoObject.DisableCommit Method

Prevents clients from completing transactions.

Pascal

```
procedure DisableCommit;
```

C++

```
__fastcall DisableCommit();
```

Description

Call DisableCommit to indicate that the MTS Automation object cannot complete its current transaction or release state information until it receives further method invocations from the client. After calling DisableCommit, the current transaction (if any) cannot be completed until the MTS Automation object calls EnableCommit (see page 5835) or SetComplete (see page 5837).

With transactions that are started automatically when the MTS Automation object's interface is called, DisableCommit is not necessary. These transactions are not committed until the MTS Automation object calls SetComplete (see page 5837). However, when the MTS Automation object is enlisted in a transaction that was started previously, the client controls the transaction and can attempt to commit it before the MTS Automation object has completed its part. Calling DisableCommit prevents the client from prematurely committing a transaction. If a client tries to commit a transaction after the MTS Automation object has called DisableCommit, the transaction aborts.

Tip: Call DisableCommit at the beginning of a method to protect your MTS Automation object from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling EnableCommit (see page 5835) or SetComplete (see page 5837) reverses this affect and allows the transaction to complete.

See Also

EnableCommit (see page 5835)

SetAbort (see page 5837)

SetComplete (see page 5837)

IsInTransaction (see page 5836)

55.1.1.2.5 TMtsAutoObject.EnableCommit Method

Allows clients to complete transactions without releasing state information.

Pascal

```
procedure EnableCommit;
```

C++

```
__fastcall EnableCommit();
```

Description

Call EnableCommit to indicate that the MTS Automation object can complete its current transaction without releasing state information. Unlike the SetComplete (see page 5837) method (which also signals that the current transaction can be committed), EnableCommit does not allow MTS to deactivate the MTS Automation object. This lets the MTS Automation object preserve state information over multiple interface calls, while still allowing clients that have enlisted the object in a transaction to commit.

By default, transactions can be completed when an interface call ends, unless the DisableCommit (see page 5834) method is called. There is no need to call EnableCommit unless the MTS Automation object has previously called DisableCommit (see page 5834).

Note: Although EnableCommit permits the current transaction to be completed, it does not automatically commit the transaction. The client that enlisted the MTS Automation object must still explicitly commit that transaction. If MTS started the transaction automatically (because the MTS Automation object requires a transaction for its interface calls), the transaction is not committed until the data module calls SetComplete (see page 5837).

See Also

DisableCommit (see page 5834)

SetComplete (see page 5837)

SetAbort (see page 5837)

IsInTransaction (see page 5836)

55.1.1.2.6 TMtsAutoObject.IsCallerInRole Method

Indicates whether the current client matches a particular security role.

Pascal

```
function IsCallerInRole(const Role: WideString): Bool;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR Role);
```

Description

MTS provides role-based security. Each level of access to your application server can be assigned a role. Using the MTS

Explorer, administrators can prevent clients from using an interface based on their role. (This is called declarative security.) In addition, however, programs can provide more fine-grained security programmatically. To do this, use IsCallerInRole to provide access to blocks of code based on the caller's security role.

IsCallerInRole checks whether the current client is part of the role named by the Role parameter. It returns true if the client matches the named role, and False otherwise. Use IsCallerInRole to prevent clients from invoking certain blocks of code if they do not have the appropriate security role.

Note: MTS security is not available on servers running Windows95. If the MTS application is running on Windows95, IsCallerInRole always returns true . In addition, IsCallerInRole always returns True if the application is part of an MTS Library package (which runs in the client's process space).

55.1.1.2.7 TMtsAutoObject.IsInTransaction Method

Indicates whether the MTS Automation object is running in a transaction.

Pascal

```
function IsInTransaction: Bool;
```

C++

```
__fastcall Bool IsInTransaction();
```

Description

Call IsInTransaction to determine whether the MTS Automation object is currently running in a transaction. For example, if the MTS Automation object requires a transaction, use IsInTransaction to check that this requirement has not been turned off by an administrator using the MTS Explorer, and to raise an exception if it has.

Note: If IsInTransaction returns False, the EnableCommit (see page 5835) and DisableCommit (see page 5834) methods have no effect.

See Also

EnableCommit (see page 5835)

DisableCommit (see page 5834)

55.1.1.2.8 TMtsAutoObject.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool;
```

C++

```
__fastcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class TMtsAutoObject.

55.1.1.2.9 TMtsAutoObject.OnActivate Method

Pascal

```
procedure OnActivate; virtual;
```

C++

```
virtual __fastcall OnActivate();
```

Description

This is OnActivate, a member of class TMtsAutoObject.

55.1.1.2.10 TMtsAutoObject.OnDeactivate Method

Pascal

```
procedure OnDeactivate; virtual;
```

C++

```
virtual __fastcall OnDeactivate();
```

Description

This is OnDeactivate, a member of class TMtsAutoObject.

55.1.1.2.11 TMtsAutoObject.SetAbort Method

Allows the MTS Automation object to be deactivated and aborts any current transactions.

Pascal

```
procedure SetAbort;
```

C++

```
__fastcall SetAbort();
```

Description

Call SetAbort to indicate that any current transaction that includes the MTS Automation object to abort. If MTS started a transaction automatically, the transaction aborts when the current method call exits. If the MTS Automation object was enlisted in a pre-existing transaction, the client transaction will eventually fail. Usually, SetAbort is called in the except block of a try...except statement.

In addition to aborting any current transaction, SetAbort indicates that the MTS Automation object can be deactivated when the current interface call exits. This means that state information is not preserved for the next time and the client calls the Automation object's interface. To prevent transactions from committing successfully without losing state information, use the DisableCommit (see page 5834) method instead.

Tip: Call SetAbort at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling SetComplete (see page 5837) reverses this affect and allows the transaction to complete.

55.1.1.2.12 TMtsAutoObject.SetComplete Method

Indicates that the MTS Automation object has successfully completed its current task.

Pascal

```
procedure SetComplete;
```

C++

```
__fastcall SetComplete();
```

Description

Call SetComplete to indicate that the MTS Automation object no longer needs to maintain state information. After calling SetComplete, the MTS Automation object is deactivated when the current method call exits.

If the MTS Automation object is part of a transaction, SetComplete indicates that this part of the transaction can commit. If MTS started the transaction automatically when the client called the MTS Automation object's methods, MTS tries to commit the transaction when the current method exits.

Note: This transaction may still fail if the current transaction included another MTS object that called SetAbort (see page 5837) or DisableCommit (see page 5834).

See Also

EnableCommit (see page 5849)

55.1.1.3 TMtsAutoObject Properties

The properties of the TMtsAutoObject class are listed here.

Protected Properties

	Name	Description
	ObjectContext (see page 5838)	This is ObjectContext, a member of class TMtsAutoObject.

Public Properties

	Name	Description
	Pooled (see page 5838)	Indicates whether the object can be pooled when installed under COM+.

55.1.1.3.1 TMtsAutoObject.ObjectContext Property

Pascal

```
property ObjectContext: IObjectContext;
```

C++

```
__property IObjectContext ObjectContext;
```

Description

This is ObjectContext, a member of class TMtsAutoObject.

55.1.1.3.2 TMtsAutoObject.Pooled Property

Indicates whether the object can be pooled when installed under COM+.

Pascal

```
property Pooled: Boolean;
```

C++

```
__property Boolean Pooled;
```

Description

Set Pooled to indicate whether instances of the MTS Automation object can be pooled when installed under COM+. The MTS Automation object uses the Pooled property when implementing the IObjectControl (see page 5866) method CanBePooled (see page 5834). This instructs COM+ that the object can be moved to an object pool when deactivated, so that the single instance is shared by all clients.

Warning: Do not set Pooled to True unless the MTS Automation object is stateless.

See Also

Object Pooling

56 MtsRdm Namespace

This is namespace MtsRdm.

Topics

Name	Description
Classes (see page 5840)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsDataModule (see page 5840)	TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

56.1 Classes

The following table lists classes in this documentation.

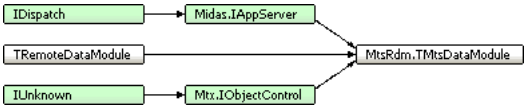
Classes

	Name	Description
	TMtsDataModule (see page 5840)	TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

56.1.1 TMtsDataModule Class

TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

Class Hierarchy



Pascal

```
TMtsDataModule = class(TRemoteDataModule, IAppServer, IObjectControl);
```

C++

```
class TMtsDataModule : public TRemoteDataModule, public IAppServer, public IObjectControl;
```

File

MtsRdm

Description

Use a TMtsDataModule object as the central repository for all objects in a transactional application server that is written in Delphi. MTS data modules can contain any nonvisual components. . Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. MTS data modules can also contain other nonvisual components, such as TTimer, or TImageList.

MTS data modules are dual-interface Automation servers that implement the IAppServer (see page 5222) interface. Connection components on client applications are designed to look for this interface. The methods in this interface, however, do not call the SetComplete (see page 5852) method, so the MTS data module is not transactional. You may want to expand the MTS data module's interface to provide alternate versions of these methods that use the MTS data module's transaction.

The default interaction between a client dataset and an MTS data module does not rely on state information, so it is safe to use the MTS data module with just-in-time activation. However, you may need to provide a mechanism for communicating persistent state information to take advantage of some features such as incremental fetches.

To create an MTS data module at design time, choose File|New|Other, and select Transactional Data Module from the multitier page of the new items dialog. If you are writing the application server in Delphi, the Transactional Data Module Wizard then creates a new descendant of TMtsDataModule that supports an interface that descends from IAppServer (see page 5222).

See Also

IAppServer (see page 5222)

Writing Applications Using COM

56.1.1.1 TMtsDataModule Members

The following tables list the members exposed by TMtsDataModule.

Protected Methods

	Name	Description
	Activate (see page 5842)	This is Activate, a member of class TMtsDataModule.
	AS_ApplyUpdates (see page 5843)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 5843)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 5844)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 5845)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 5846)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 5846)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 5847)	Returns information from a specified record of the provider's dataset.
V	CallComplete (see page 5848)	This is CallComplete, a member of class TMtsDataModule.
V	CanBePooled (see page 5848)	This is CanBePooled, a member of class TMtsDataModule.
	Deactivate (see page 5848)	This is Deactivate, a member of class TMtsDataModule.

Protected Properties

	Name	Description
R	ObjectContext (see page 5853)	This is ObjectContext, a member of class TMtsDataModule.

Public Methods

	Name	Description
V	Create (see page 5841)	Creates an MTS data module instance.
	DisableCommit (see page 5849)	Prevents clients from completing transactions.
	EnableCommit (see page 5849)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 5850)	Indicates whether the current client fits into a particular security role.
	IsInTransaction (see page 5850)	Indicates whether the MTS data module is running in a transaction.
	IsSecurityEnabled (see page 5851)	This is IsSecurityEnabled, a member of class TMtsDataModule.
	SetAbort (see page 5851)	Allows the MTS data module to be deactivated and aborts any current transactions.
	SetComplete (see page 5852)	Indicates that the MTS data module has successfully completed its current task.

Published Events

	Name	Description
	OnActivate (see page 5854)	Occurs when an instance of TMtsDataModule (see page 5840) is activated.
	OnDeactivate (see page 5854)	Occurs when an instance of TMtsDataModule (see page 5840) is deactivated.

Published Properties

	Name	Description
	AutoComplete (see page 5853)	Specifies whether the IAppServer (see page 5222) methods are transactional.
	Pooled (see page 5853)	Indicates whether the data module can be pooled when installed under COM+.

56.1.1.2 TMtsDataModule.Create Constructor

Creates an MTS data module instance.

Pascal

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

C++

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

Description

Most applications do not need to call the constructor for TMtsDataModule (see page 5840). Instead, add a remote data module to a server application using the transactional data module wizard on the multi-tier page of the new items dialog.

AOwner specifies the component that is responsible for freeing the remote data module (typically the application).

See Also

Create

56.1.1.3 TMtsDataModule Methods

The methods of the TMtsDataModule class are listed here.

Protected Methods

	Name	Description
	Activate (see page 5842)	This is Activate, a member of class TMtsDataModule.
	AS_ApplyUpdates (see page 5843)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 5843)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 5844)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 5845)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 5846)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 5846)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 5847)	Returns information from a specified record of the provider's dataset.
	CallComplete (see page 5848)	This is CallComplete, a member of class TMtsDataModule.
	CanBePooled (see page 5848)	This is CanBePooled, a member of class TMtsDataModule.
	Deactivate (see page 5848)	This is Deactivate, a member of class TMtsDataModule.

Public Methods

	Name	Description
	DisableCommit (see page 5849)	Prevents clients from completing transactions.
	EnableCommit (see page 5849)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 5850)	Indicates whether the current client fits into a particular security role.
	IsInTransaction (see page 5850)	Indicates whether the MTS data module is running in a transaction.
	IsSecurityEnabled (see page 5851)	This is IsSecurityEnabled, a member of class TMtsDataModule.
	SetAbort (see page 5851)	Allows the MTS data module to be deactivated and aborts any current transactions.
	SetComplete (see page 5852)	Indicates that the MTS data module has successfully completed its current task.

56.1.1.3.1 TMtsDataModule.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TMtsDataModule.

56.1.1.3.2 TMtsDataModule.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server. ProviderName specifies the name of the provider component that manages the update operation. Delta is a Variant containing the updated, inserted, and deleted records to write to the database. MaxErrors indicates the maximum number of errors to permit before rolling back the update operation; ErrorCount returns the actual number of errors encountered during the update operation. OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler. It is passed to the provider's BeforeAS_ApplyUpdates event handler and returned from the provider's AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates The AS_ApplyUpdates_result parameter returns a Variant that is a data packet containing all records that could not be applied.

Note: Applications can only call the protected AS_AS_ApplyUpdates method using the IAppServer interface. To apply updates directly in code from the application server, use the provider component's AS_ApplyUpdates method.

See Also

ApplyUpdates

applyupdates (see page 6468)

BeforeApplyUpdates (see page 6487)

AfterApplyUpdates (see page 6483)

BeforeApplyUpdates

AfterApplyUpdates

providers

IAppServer (see page 5222)

56.1.1.3.3 TMtsDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This

method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value AS_DataRequest_result parameter, which is returned to the caller.

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest (see page 6490)

providers

DataRequest

IAppServer (see page 5222)

56.1.1.3.4 TMtsDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 6525)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

Params

Execute

Execute (see page 6474)

BeforeExecute

AfterExecute

Options (see page 6455)

PackageParams

UnpackParams

Communicating with the Client Dataset

56.1.1.3.5 TMtsDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 6525)

Params

FetchParams

GetParams (see page 6475)

BeforeGetParams (see page 6488)

aftergetparams (see page 6485)

AS_Execute (see page 5224)

UnpackParams

Communicating with the Client Dataset

56.1.1.3.6 TMtsDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the remote data module. When calling any of the remote data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName

providers

IAppServer (see page 5222)

Exported (see page 6482)

56.1.1.3.7 TMtsDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant parameter, AS_GetRecords_result.

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data

Params

GetNextPacket

GetRecords (see page 6476)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

providers

IAppServer (see page 5222)

56.1.1.3.8 TMtsDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest (see page 6480)

BeforeRowRequest (see page 6490)

AfterRowRequest (see page 6486)

FetchBlobs

FetchDetails

RefreshRecord

providers

IAppServer (see page 5222)

56.1.1.3.9 TMtsDataModule.CallComplete Method

Pascal

```
procedure CallComplete(Complete: Boolean); virtual;
```

C++

```
virtual __fastcall CallComplete(Boolean Complete);
```

Description

This is CallComplete, a member of class TMtsDataModule.

56.1.1.3.10 TMtsDataModule.CanBePooled Method

Pascal

```
function CanBePooled: Bool; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class TMtsDataModule.

56.1.1.3.11 TMtsDataModule.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class TMtsDataModule.

56.1.1.3.12 TMtsDataModule.DisableCommit Method

Prevents clients from completing transactions.

Pascal

```
procedure DisableCommit;
```

C++

```
__fastcall DisableCommit();
```

Description

Call DisableCommit to indicate that the MTS data module can't complete its current transaction or release state information (properties) until it receives further method invocations from the client. After calling DisableCommit, the current transaction (if any) can't be completed until the MTS data module calls EnableCommit (see page 5849) or SetComplete (see page 5852).

With transactions that are started automatically when the MTS data module's interface is called, DisableCommit is not necessary. These automatically generated transactions are not committed until the MTS data module calls SetComplete (see page 5852). However, when the MTS data module is enlisted in a transaction that was started previously, the client controls the transaction and can attempt to commit it before the MTS data module has completed its part. Calling DisableCommit prevents the client from prematurely committing a transaction. If a client tries to commit a transaction after the MTS data module has called DisableCommit, the transaction aborts.

Tip: Call DisableCommit at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling EnableCommit (see page 5849) or SetComplete (see page 5852) reverses this affect and allows the transaction to complete.

See Also

EnableCommit (see page 5849)

SetComplete (see page 5852)

SetAbort (see page 5851)

IsInTransaction (see page 5850)

56.1.1.3.13 TMtsDataModule.EnableCommit Method

Allows clients to complete transactions without releasing state information.

Pascal

```
procedure EnableCommit;
```

C++

```
__fastcall EnableCommit();
```

Description

Call EnableCommit to indicate that the MTS data module can complete its current transaction without releasing state information (properties). Unlike the SetComplete (see page 5852) method (which also signals that the current transaction can be committed), EnableCommit does not allow the data module to be deactivated. This lets the MTS data module preserve state information over multiple interface calls, while still allowing clients that have enlisted the data module in a transaction to commit.

By default, transactions can be completed when interface calls finish, unless the DisableCommit (see page 5849) method is

called. There is no need to call EnableCommit unless the MTS data module has previously called DisableCommit (see page 5849).

Note: Although EnableCommit permits the current transaction to be completed, it does not automatically commit the transaction. The client that enlisted the MTS data module in a transaction must still explicitly commit that transaction. If the transaction was started automatically (because the MTS data module requires a transaction for its interface calls), the transaction is not committed until the data module calls SetComplete (see page 5852).

See Also

DisableCommit (see page 5849)

SetComplete (see page 5852)

SetAbort (see page 5851)

IsInTransaction (see page 5850)

56.1.1.3.14 TMtsDataModule.IsCallerInRole Method

Indicates whether the current client fits into a particular security role.

Pascal

```
function IsCallerInRole(const Role: WideString): Bool;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR Role);
```

Description

MTS provides role-based security. Each level of access to the application server can be assigned a name, called a role. Using the MTS Explorer or COM+ administrator, administrators can prevent clients from using an interface based on their role. (This is called declarative security.) In addition, however, programs can provide more fine-grained security programmatically. To do this, use IsCallerInRole to provide access to blocks of code based on the caller's security role.

IsCallerInRole checks whether the current client is part of the role named by the Role parameter. It returns true if the client fits the named role, and false otherwise. Use IsCallerInRole to prevent clients that do not have the appropriate security roles from invoking certain blocks of code.

Note: MTS security is not available on servers running Windows95. If the application server is running on Windows95, IsCallerInRole always returns true. In addition, IsCallerInRole always returns true if the application server is part of an MTS Library package (which runs in the client's process space).

See Also

issecurityenabled (see page 5851)

Using Transactional Data Modules

56.1.1.3.15 TMtsDataModule.IsInTransaction Method

Indicates whether the MTS data module is running in a transaction.

Pascal

```
function IsInTransaction: Bool;
```

C++

```
__fastcall Bool IsInTransaction();
```

Description

Call IsInTransaction to determine whether the MTS data module is currently running in a transaction. For example, if the MTS data module requires a transaction, use IsInTransaction to check that this requirement has not been turned off using the MTS Explorer or COM+ administrator, and to raise an exception if it has.

Note: If IsInTransaction returns false, the EnableCommit (see page 5849) and DisableCommit (see page 5849) methods have no effect.

See Also

EnableCommit (see page 5849)

DisableCommit (see page 5849)

SetComplete (see page 5852)

SetAbort (see page 5851)

56.1.1.3.16 TMtsDataModule.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool;
```

C++

```
__fastcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class TMtsDataModule.

56.1.1.3.17 TMtsDataModule.SetAbort Method

Allows the MTS data module to be deactivated and aborts any current transactions.

Pascal

```
procedure SetAbort;
```

C++

```
__fastcall SetAbort();
```

Description

Call SetAbort to indicate that any current transaction that includes the MTS data module should abort. If MTS or COM+ started a transaction automatically (because the data module requires one), that transaction aborts when the current method call exits. If the MTS data module was enlisted in a pre-existing transaction, the client transaction will eventually fail. Usually, SetAbort is called in the except block of a try..exceptstatement.

In addition to aborting any current transaction, SetAbort indicates that the MTS data module can be deactivated when the current interface call exits. This means that state information is not preserved for the next time the client calls the data module's interface. To prevent transactions from committing successfully without losing state information, use the DisableCommit (see page 5849) method instead.

Tip: Call SetAbort at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling SetComplete (see page 5852) reverses this affect and allows the transaction to complete.

See Also

SetComplete (see page 5852)

DisableCommit (see page 5849)

EnableCommit (see page 5849)

IsInTransaction (see page 5850)

56.1.1.3.18 TMtsDataModule.SetComplete Method

Indicates that the MTS data module has successfully completed its current task.

Pascal

```
procedure SetComplete;
```

C++

```
__fastcall SetComplete();
```

Description

Call SetComplete to indicate that the MTS data module no longer needs to maintain state information. After calling SetComplete, the MTS data module is deactivated when the current method call exits.

If the MTS data module is part of a transaction, SetComplete indicates that this part of the transaction can commit. If MTS or COM+ started the transaction automatically when the client called the MTS data module's method (because the MTS data module requires a transaction), it tries to commit the transaction when the current method exits.

Note: This transaction may still fail if the current transaction included another MTS or COM+ object that called SetAbort (see page 5851) or DisableCommit (see page 5849).

See Also

SetAbort (see page 5851)

DisableCommit (see page 5849)

EnableCommit (see page 5849)

IsInTransaction (see page 5850)

Extending the Interface of the Application Server

56.1.1.4 TMtsDataModule Properties

The properties of the TMtsDataModule class are listed here.

Protected Properties

	Name	Description
	ObjectContext (see page 5853)	This is ObjectContext, a member of class TMtsDataModule.

Published Properties

	Name	Description
	AutoComplete (see page 5853)	Specifies whether the IAppServer (see page 5222) methods are transactional.
	Pooled (see page 5853)	Indicates whether the data module can be pooled when installed under COM+.

56.1.1.4.1 TMtsDataModule.AutoComplete Property

Specifies whether the IAppServer (see page 5222) methods are transactional.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete controls whether calls made using the IAppServer (see page 5222) interface end in a call to SetComplete (see page 5852) or SetAbort (see page 5851). By calling SetComplete (see page 5852) or SetAbort (see page 5851), the MTS data module indicates that no state information needs to be preserved when the call exits. This allows any MTS transactions to complete (or roll back) and allows COM+ or the MTS proxy to deactivate the MTS data module.

When AutoComplete is true, all IAppServer (see page 5222) methods call SetComplete (see page 5852) when they are done with two exceptions:

AS_ApplyUpdates (see page 5843) calls SetAbort (see page 5851) if an exception is raised or if the error count exceeds the specified maximum.

AS_DataRequest (see page 5843) doesn't call either SetComplete (see page 5852) or SetAbort (see page 5851). You must add the appropriate call to the OnDataRequest event handler.

Set AutoComplete to false if the MTS data module needs to preserve state information (such as the current record in a dataset) after an IAppServer (see page 5222) call exits. When AutoComplete is false, you must add your own interface methods that call SetComplete (see page 5852) or SetAbort (see page 5851) when the MTS data module can be safely deactivated.

See Also

IAppServer (see page 5222)

SetComplete (see page 5852)

SetAbort (see page 5851)

56.1.1.4.2 TMtsDataModule.ObjectContext Property

Pascal

```
property ObjectContext: IObjectContext;
```

C++

```
__property IObjectContext ObjectContext;
```

Description

This is ObjectContext, a member of class TMtsDataModule.

56.1.1.4.3 TMtsDataModule.Pooled Property

Indicates whether the data module can be pooled when installed under COM+.

Pascal

```
property Pooled: Boolean;
```

C++

```
__property Boolean Pooled;
```

Description

Set Pooled to indicate whether instances of the MTS data module can be pooled when installed under COM+. The MTS remote data module uses the Pooled property when implementing the IObjectControl (see page 5866) method CanBePooled (see page 5848). This instructs COM+ that the object can be moved to an object pool when deactivated, so that the single instance is shared by all clients.

Warning: Do not set Pooled to True unless the MTS data module is stateless.

56.1.1.5 TMtsDataModule Events

The events of the TMtsDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 5854)	Occurs when an instance of TMtsDataModule (see page 5840) is activated.
	OnDeactivate (see page 5854)	Occurs when an instance of TMtsDataModule (see page 5840) is deactivated.

56.1.1.5.1 TMtsDataModule.OnActivate Event

Occurs when an instance of TMtsDataModule (see page 5840) is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to take specific action when COMN+ or the MTS proxy activates the MTS data module. This is the point when any database connections should be established. Database connections must not opened before the MTS data module is activated.

Note: In traditional MTS or COM+ programming, objects acquire their context object when they are activated. However, TMtsDataModule (see page 5840) handles this task automatically.

See Also

OnDeactivate (see page 5854)

56.1.1.5.2 TMtsDataModule.OnDeactivate Event

Occurs when an instance of TMtsDataModule (see page 5840) is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to take specific action when COM+ or the MTS proxy releases an instance of the MTS data

module. Use this method to close any database connections that were opened when the MTS data module was activated. Database connections must not be left open after the MTS data module is deactivated.

Note: In traditional COM+ and MTS programming, objects release their context object when they are deactivated. However, TMtsDataModule (see page 5840) handles this task automatically.

See Also

OnActivate (see page 5854)

57 Mtx Namespace

This is namespace Mtx.

Topics

Name	Description
Interfaces (see page 5859)	The following table lists interfaces in this documentation.
Functions (see page 5879)	The following table lists functions in this documentation.
Constants (see page 5884)	The following table lists constants in this documentation.

Constants

Name	Description
CLASS_SharedPropertyGroupManager (see page 5884)	This is constant Mtx.CLASS_SharedPropertyGroupManager.
CLASS_TransactionContext (see page 5884)	This is constant Mtx.CLASS_TransactionContext.
CLASS_TransactionContextEx (see page 5885)	This is constant Mtx.CLASS_TransactionContextEx.
LockMethod (see page 5885)	This is constant Mtx.LockMethod.
LockSetGet (see page 5885)	This is constant Mtx.LockSetGet.
mtsErrCtxAborted (see page 5886)	This is constant Mtx.mtsErrCtxAborted.
mtsErrCtxAborting (see page 5886)	This is constant Mtx.mtsErrCtxAborting.
mtsErrCtxActivityTimeout (see page 5886)	This is constant Mtx.mtsErrCtxActivityTimeout.
mtsErrCtxNoContext (see page 5887)	This is constant Mtx.mtsErrCtxNoContext.
mtsErrCtxNoSecurity (see page 5887)	This is constant Mtx.mtsErrCtxNoSecurity.
mtsErrCtxNotRegistered (see page 5887)	This is constant Mtx.mtsErrCtxNotRegistered.
mtsErrCtxOldReference (see page 5888)	This is constant Mtx.mtsErrCtxOldReference.
mtsErrCtxRoleNotFound (see page 5888)	This is constant Mtx.mtsErrCtxRoleNotFound.
mtsErrCtxTMNotAvailable (see page 5888)	This is constant Mtx.mtsErrCtxTMNotAvailable.
mtsErrCtxWrongThread (see page 5889)	This is constant Mtx.mtsErrCtxWrongThread.
Process (see page 5889)	This is constant Mtx.Process.
Standard (see page 5889)	This is constant Mtx.Standard.

Functions

	Name	Description
	CreateSharedPropertyGroup (see page 5879)	Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.
	CreateSharedPropertyGroupManager (see page 5880)	Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.
	CreateTransactionContext (see page 5880)	Starts an MTS transaction.

	CreateTransactionContextEx (see page 5881)	Instantiates a TransactionContextEx object, which marks the beginning of a transaction.
	GetObjectContext (see page 5882)	Fetches the IObjectContext (see page 5862) interface for the current thread.
	SafeRef (see page 5882)	Returns a reference to a transactional object that is safe to pass outside its context.

Interfaces

	Name	Description
	IEnumNames (see page 5859)	This is class Mtx.IEnumNames.
	IGetContextProperties (see page 5861)	This is class Mtx.IGetContextProperties.
	IObjectContext (see page 5862)	This is class Mtx.IObjectContext.
	IObjectContextActivity (see page 5865)	This is class Mtx.IObjectContextActivity.
	IObjectControl (see page 5866)	This is class Mtx.IObjectControl.
	ISecurityProperty (see page 5867)	This is class Mtx.ISecurityProperty.
	ISharedProperty (see page 5869)	This is class Mtx.ISharedProperty.
	ISharedPropertyGroup (see page 5871)	This is class Mtx.ISharedPropertyGroup.
	ISharedPropertyGroupManager (see page 5873)	This is class Mtx.ISharedPropertyGroupManager.
	ITransactionContext (see page 5875)	This is class Mtx.ITransactionContext.
	ITransactionContextEx (see page 5877)	This is class Mtx.ITransactionContextEx.

57.1 Interfaces

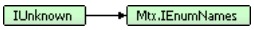
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IEnumNames (see page 5859)	This is class Mtx.IEnumNames.
	IGetContextProperties (see page 5861)	This is class Mtx.IGetContextProperties.
	IObjectContext (see page 5862)	This is class Mtx.IObjectContext.
	IObjectContextActivity (see page 5865)	This is class Mtx.IObjectContextActivity.
	IObjectControl (see page 5866)	This is class Mtx.IObjectControl.
	ISecurityProperty (see page 5867)	This is class Mtx.ISecurityProperty.
	ISharedProperty (see page 5869)	This is class Mtx.ISharedProperty.
	ISharedPropertyGroup (see page 5871)	This is class Mtx.ISharedPropertyGroup.
	ISharedPropertyGroupManager (see page 5873)	This is class Mtx.ISharedPropertyGroupManager.
	ITransactionContext (see page 5875)	This is class Mtx.ITransactionContext.
	ITransactionContextEx (see page 5877)	This is class Mtx.ITransactionContextEx.

57.1.1 IEnumNames Interface

Class Hierarchy



Pascal

```
IEnumNames = interface(IUnknown);
```

C++

```
__interface IEnumNames : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IEnumNames.

57.1.1.1 IEnumNames Members

The following tables list the members exposed by IEnumNames.

Public Methods

	Name	Description
	Clone (see page 5860)	This is Clone, a member of class IEnumNames.
	Next (see page 5860)	This is Next, a member of class IEnumNames.
	Reset (see page 5860)	This is Reset, a member of class IEnumNames.
	Skip (see page 5860)	This is Skip, a member of class IEnumNames.

57.1.1.2 IEnumNames Methods

The methods of the IEnumNames class are listed here.

Public Methods

	Name	Description
	Clone (see page 5860)	This is Clone, a member of class IEnumNames.
	Next (see page 5860)	This is Next, a member of class IEnumNames.
	Reset (see page 5860)	This is Reset, a member of class IEnumNames.
	Skip (see page 5860)	This is Skip, a member of class IEnumNames.

57.1.1.2.1 IEnumNames.Clone Method

Pascal

```
function Clone(out enm: IEnumNames): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Clone(IEnumNames enm);
```

Description

This is Clone, a member of class IEnumNames.

57.1.1.2.2 IEnumNames.Next Method

Pascal

```
function Next(celt: UINT; out rgname: WideString; out pceltFetched: UINT): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Next(UINT celt, BSTR rgname, UINT pceltFetched);
```

Description

This is Next, a member of class IEnumNames.

57.1.1.2.3 IEnumNames.Reset Method

Pascal

```
function Reset: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Reset();
```

Description

This is Reset, a member of class IEnumNames.

57.1.1.2.4 IEnumNames.Skip Method

Pascal

```
function Skip(celt: UINT): HResult; stdcall;
```

C++

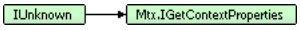
```
__fastcall __stdcall HRESULT Skip(UINT celt);
```

Description

This is Skip, a member of class IEnumNames.

57.1.2 IGetContextProperties Interface

Class Hierarchy



Pascal

```
IGetContextProperties = interface(IUnknown);
```

C++

```
__interface IGetContextProperties : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IGetContextProperties.

57.1.2.1 IGetContextProperties Members

The following tables list the members exposed by IGetContextProperties.

Public Methods

	Name	Description
	Count (see page 5861)	This is Count, a member of class IGetContextProperties.
	EnumNames (see page 5862)	This is EnumNames, a member of class IGetContextProperties.
	GetProperty (see page 5862)	This is GetProperty, a member of class IGetContextProperties.

57.1.2.2 IGetContextProperties Methods

The methods of the IGetContextProperties class are listed here.

Public Methods

	Name	Description
	Count (see page 5861)	This is Count, a member of class IGetContextProperties.
	EnumNames (see page 5862)	This is EnumNames, a member of class IGetContextProperties.
	GetProperty (see page 5862)	This is GetProperty, a member of class IGetContextProperties.

57.1.2.2.1 IGetContextProperties.Count Method

Pascal

```
function Count: Integer; safecall;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class IGetContextProperties.

57.1.2.2.2 IGetContextProperties.EnumNames Method

Pascal

```
function EnumNames: IEnumNames; safecall;
```

C++

```
__fastcall IEnumNames EnumNames();
```

Description

This is EnumNames, a member of class IGetContextProperties.

57.1.2.2.3 IGetContextProperties.GetProperty Method

Pascal

```
function GetProperty(const name: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR name);
```

Description

This is GetProperty, a member of class IGetContextProperties.

57.1.3 IObjectContext Interface

Class Hierarchy

Pascal

```
IObjectContext = interface(IUnknown);
```

C++

```
__interface IObjectContext : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectContext.

57.1.3.1 IObjectContext Members

The following tables list the members exposed by IObjectContext.

Public Methods

	Name	Description
	CreateInstance (see page 5863)	This is CreateInstance, a member of class IObjectContext.
	DisableCommit (see page 5863)	This is DisableCommit, a member of class IObjectContext.
	EnableCommit (see page 5864)	This is EnableCommit, a member of class IObjectContext.
	IsCallerInRole (see page 5864)	This is IsCallerInRole, a member of class IObjectContext.
	IsInTransaction (see page 5864)	This is IsInTransaction, a member of class IObjectContext.
	IsSecurityEnabled (see page 5864)	This is IsSecurityEnabled, a member of class IObjectContext.
	SetAbort (see page 5864)	This is SetAbort, a member of class IObjectContext.
	SetComplete (see page 5865)	This is SetComplete, a member of class IObjectContext.

57.1.3.2 IObjectContext Methods

The methods of the IObjectContext class are listed here.

Public Methods

	Name	Description
	CreateInstance (see page 5863)	This is CreateInstance, a member of class IObjectContext.
	DisableCommit (see page 5863)	This is DisableCommit, a member of class IObjectContext.
	EnableCommit (see page 5864)	This is EnableCommit, a member of class IObjectContext.
	IsCallerInRole (see page 5864)	This is IsCallerInRole, a member of class IObjectContext.
	IsInTransaction (see page 5864)	This is IsInTransaction, a member of class IObjectContext.
	IsSecurityEnabled (see page 5864)	This is IsSecurityEnabled, a member of class IObjectContext.
	SetAbort (see page 5864)	This is SetAbort, a member of class IObjectContext.
	SetComplete (see page 5865)	This is SetComplete, a member of class IObjectContext.

57.1.3.2.1 IObjectContext.CreateInstance Method

Pascal

```
function CreateInstance(const cid: TGUID; const rid: TGUID; out pv): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const TGUID cid, const TGUID rid,  pv);
```

Description

This is CreateInstance, a member of class IObjectContext.

57.1.3.2.2 IObjectContext.DisableCommit Method

Pascal

```
procedure DisableCommit; safecall;
```

C++

```
__fastcall DisableCommit();
```

Description

This is DisableCommit, a member of class IObjectContext.

57.1.3.2.3 IObjectContext.EnableCommit Method

Pascal

```
procedure EnableCommit; safecall;
```

C++

```
__fastcall EnableCommit();
```

Description

This is EnableCommit, a member of class IObjectContext.

57.1.3.2.4 IObjectContext.IsCallerInRole Method

Pascal

```
function IsCallerInRole(const bstrRole: WideString): Bool; safecall;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR bstrRole);
```

Description

This is IsCallerInRole, a member of class IObjectContext.

57.1.3.2.5 IObjectContext.IsInTransaction Method

Pascal

```
function IsInTransaction: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool IsInTransaction();
```

Description

This is IsInTransaction, a member of class IObjectContext.

57.1.3.2.6 IObjectContext.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class IObjectContext.

57.1.3.2.7 IObjectContext.SetAbort Method

Pascal

```
procedure SetAbort; safecall;
```

C++

```
__fastcall SetAbort();
```

Description

This is SetAbort, a member of class IObjectContext.

57.1.3.2.8 IObjectContext.SetComplete Method

Pascal

```
procedure SetComplete; safecall;
```

C++

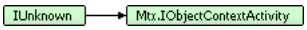
```
__fastcall SetComplete();
```

Description

This is SetComplete, a member of class IObjectContext.

57.1.4 IObjectContextActivity Interface

Class Hierarchy



Pascal

```
IObjectContextActivity = interface(IUnknown);
```

C++

```
__interface IObjectContextActivity : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectContextActivity.

57.1.4.1 IObjectContextActivity Members

The following tables list the members exposed by IObjectContextActivity.

Public Methods

	Name	Description
	GetActivityId (see page 5866)	This is GetActivityId, a member of class IObjectContextActivity.

57.1.4.2 IObjectContextActivity Methods

The methods of the IObjectContextActivity class are listed here.

Public Methods

	Name	Description
	GetActivityId (see page 5866)	This is GetActivityId, a member of class IObjectContextActivity.

57.1.4.2.1 IObjectContextActivity.GetActivityId Method

Pascal

```
function GetActivityId: TGuid; safecall;
```

C++

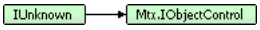
```
__fastcall TGuid GetActivityId();
```

Description

This is GetActivityId, a member of class IObjectContextActivity.

57.1.5 IObjectControl Interface

Class Hierarchy



Pascal

```
IObjectControl = interface(IUnknown);
```

C++

```
__interface IObjectControl : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectControl.

57.1.5.1 IObjectControl Members

The following tables list the members exposed by IObjectControl.

Public Methods

	Name	Description
	Activate (see page 5867)	This is Activate, a member of class IObjectControl.
	CanBePooled (see page 5867)	This is CanBePooled, a member of class IObjectControl.
	Deactivate (see page 5867)	This is Deactivate, a member of class IObjectControl.

57.1.5.2 IObjectControl Methods

The methods of the IObjectControl class are listed here.

Public Methods

	Name	Description
	Activate (see page 5867)	This is Activate, a member of class IObjectControl.
	CanBePooled (see page 5867)	This is CanBePooled, a member of class IObjectControl.
	Deactivate (see page 5867)	This is Deactivate, a member of class IObjectControl.

57.1.5.2.1 IObjectControl.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class IObjectControl.

57.1.5.2.2 IObjectControl.CanBePooled Method

Pascal

```
function CanBePooled: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class IObjectControl.

57.1.5.2.3 IObjectControl.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class IObjectControl.

57.1.6 ISecurityProperty Interface

Class Hierarchy

Pascal

```
ISecurityProperty = interface(IUnknown);
```

C++

```
__interface ISecurityProperty : public IUnknown;
```

File

Mtx

Description

This is class Mtx.ISecurityProperty.

57.1.6.1 ISecurityProperty Members

The following tables list the members exposed by ISecurityProperty.

Public Methods

	Name	Description
	GetDirectCallerSID (see page 5868)	This is GetDirectCallerSID, a member of class ISecurityProperty.
	GetDirectCreatorSID (see page 5868)	This is GetDirectCreatorSID, a member of class ISecurityProperty.
	GetOriginalCallerSID (see page 5868)	This is GetOriginalCallerSID, a member of class ISecurityProperty.
	GetOriginalCreatorSID (see page 5869)	This is GetOriginalCreatorSID, a member of class ISecurityProperty.
	ReleaseSID (see page 5869)	This is ReleaseSID, a member of class ISecurityProperty.

57.1.6.2 ISecurityProperty Methods

The methods of the ISecurityProperty class are listed here.

Public Methods

	Name	Description
	GetDirectCallerSID (see page 5868)	This is GetDirectCallerSID, a member of class ISecurityProperty.
	GetDirectCreatorSID (see page 5868)	This is GetDirectCreatorSID, a member of class ISecurityProperty.
	GetOriginalCallerSID (see page 5868)	This is GetOriginalCallerSID, a member of class ISecurityProperty.
	GetOriginalCreatorSID (see page 5869)	This is GetOriginalCreatorSID, a member of class ISecurityProperty.
	ReleaseSID (see page 5869)	This is ReleaseSID, a member of class ISecurityProperty.

57.1.6.2.1 ISecurityProperty.GetDirectCallerSID Method

Pascal

```
function GetDirectCallerSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDirectCallerSID(PSID sid);
```

Description

This is GetDirectCallerSID, a member of class ISecurityProperty.

57.1.6.2.2 ISecurityProperty.GetDirectCreatorSID Method

Pascal

```
function GetDirectCreatorSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDirectCreatorSID(PSID sid);
```

Description

This is GetDirectCreatorSID, a member of class ISecurityProperty.

57.1.6.2.3 ISecurityProperty.GetOriginalCallerSID Method

Pascal

```
function GetOriginalCallerSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetOriginalCallerSID(PSID sid);
```

Description

This is GetOriginalCallerSID, a member of class ISecurityProperty.

57.1.6.2.4 ISecurityProperty.GetOriginalCreatorSID Method

Pascal

```
function GetOriginalCreatorSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetOriginalCreatorSID(PSID sid);
```

Description

This is GetOriginalCreatorSID, a member of class ISecurityProperty.

57.1.6.2.5 ISecurityProperty.ReleaseSID Method

Pascal

```
function ReleaseSID(sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ReleaseSID(PSID sid);
```

Description

This is ReleaseSID, a member of class ISecurityProperty.

57.1.7 ISharedProperty Interface

Class Hierarchy

Pascal

```
ISharedProperty = interface(IDispatch);
```

C++

```
__interface ISharedProperty : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedProperty.

57.1.7.1 ISharedProperty Members

The following tables list the members exposed by ISharedProperty.

Public Methods

	Name	Description
	Get_Value (see page 5870)	This is Get_Value, a member of class ISharedProperty.
	Set_Value (see page 5870)	This is Set_Value, a member of class ISharedProperty.

Public Properties

	Name	Description
	Value (see page 5871)	This is Value, a member of class ISharedProperty.

57.1.7.2 ISharedProperty Methods

The methods of the ISharedProperty class are listed here.

Public Methods

	Name	Description
	Get_Value (see page 5870)	This is Get_Value, a member of class ISharedProperty.
	Set_Value (see page 5870)	This is Set_Value, a member of class ISharedProperty.

57.1.7.2.1 ISharedProperty.Get_Value Method

Pascal

```
function Get_Value: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Value();
```

Description

This is Get_Value, a member of class ISharedProperty.

57.1.7.2.2 ISharedProperty.Set_Value Method

Pascal

```
procedure Set_Value(pVal: OleVariant); safecall;
```

C++

```
__fastcall Set_Value(OleVariant pVal);
```

Description

This is Set_Value, a member of class ISharedProperty.

57.1.7.3 ISharedProperty Properties

The properties of the ISharedProperty class are listed here.

Public Properties

	Name	Description
	Value (see page 5871)	This is Value, a member of class ISharedProperty.

57.1.7.3.1 ISharedProperty.Value Property

Pascal

```
property Value: OleVariant;
```

C++

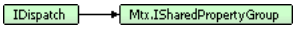
```
__property OleVariant Value;
```

Description

This is Value, a member of class ISharedProperty.

57.1.8 ISharedPropertyGroup Interface

Class Hierarchy



Pascal

```
ISharedPropertyGroup = interface(IDispatch);
```

C++

```
__interface ISharedPropertyGroup : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedPropertyGroup.

57.1.8.1 ISharedPropertyGroup Members

The following tables list the members exposed by ISharedPropertyGroup.

Public Methods

	Name	Description
	CreateProperty (see page 5872)	This is CreateProperty, a member of class ISharedPropertyGroup.
	CreatePropertyByPosition (see page 5872)	This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.
	Get_PropertyByName (see page 5872)	This is Get_PropertyByName, a member of class ISharedPropertyGroup.
	Get_PropertyByPosition (see page 5872)	This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.

Public Properties

	Name	Description
R	PropertyByName (see page 5873)	This is PropertyByName, a member of class ISharedPropertyGroup.
R	PropertyByPosition (see page 5873)	This is PropertyByPosition, a member of class ISharedPropertyGroup.

57.1.8.2 ISharedPropertyGroup Methods

The methods of the ISharedPropertyGroup class are listed here.

Public Methods

	Name	Description
	CreateProperty (see page 5872)	This is CreateProperty, a member of class ISharedPropertyGroup.
	CreatePropertyByPosition (see page 5872)	This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.
	Get_PropertyByName (see page 5872)	This is Get_PropertyByName, a member of class ISharedPropertyGroup.
	Get_PropertyByPosition (see page 5872)	This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.

57.1.8.2.1 ISharedPropertyGroup.CreateProperty Method

Pascal

```
function CreateProperty(const Name: WideString; out fExists: WordBool): ISharedProperty;
safecall;
```

C++

```
__fastcall ISharedProperty CreateProperty(const BSTR Name, WordBool fExists);
```

Description

This is CreateProperty, a member of class ISharedPropertyGroup.

57.1.8.2.2 ISharedPropertyGroup.CreatePropertyByPosition Method

Pascal

```
function CreatePropertyByPosition(Index: Integer; out fExists: WordBool): ISharedProperty;
safecall;
```

C++

```
__fastcall ISharedProperty CreatePropertyByPosition(int Index, WordBool fExists);
```

Description

This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.

57.1.8.2.3 ISharedPropertyGroup.Get_PropertyByName Method

Pascal

```
function Get_PropertyByName(const Name: WideString): ISharedProperty; safecall;
```

C++

```
__fastcall ISharedProperty Get_PropertyByName(const BSTR Name);
```

Description

This is Get_PropertyByName, a member of class ISharedPropertyGroup.

57.1.8.2.4 ISharedPropertyGroup.Get_PropertyByPosition Method

Pascal

```
function Get_PropertyByPosition(Index: Integer): ISharedProperty; safecall;
```

C++

```
__fastcall ISharedProperty Get_PropertyByPosition(int Index);
```

Description

This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.

57.1.8.3 ISharedPropertyGroup Properties

The properties of the ISharedPropertyGroup class are listed here.

Public Properties

	Name	Description
	PropertyByName (see page 5873)	This is PropertyByName, a member of class ISharedPropertyGroup.
	PropertyByPosition (see page 5873)	This is PropertyByPosition, a member of class ISharedPropertyGroup.

57.1.8.3.1 ISharedPropertyGroup.PropertyByName Property

Pascal

```
property PropertyByName [const Name: WideString]: ISharedProperty;
```

C++

```
__property ISharedProperty PropertyByName[BSTR const Name];
```

Description

This is PropertyByName, a member of class ISharedPropertyGroup.

57.1.8.3.2 ISharedPropertyGroup.PropertyByPosition Property

Pascal

```
property PropertyByPosition [Index: Integer]: ISharedProperty;
```

C++

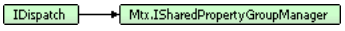
```
__property ISharedProperty PropertyByPosition[int Index];
```

Description

This is PropertyByPosition, a member of class ISharedPropertyGroup.

57.1.9 ISharedPropertyGroupManager Interface

Class Hierarchy



Pascal

```
ISharedPropertyGroupManager = interface(IDispatch);
```

C++

```
__interface ISharedPropertyGroupManager : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedPropertyGroupManager.

57.1.9.1 ISharedPropertyGroupManager Members

The following tables list the members exposed by ISharedPropertyGroupManager.

Public Methods

	Name	Description
	CreatePropertyGroup (see page 5874)	This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.
	Get__NewEnum (see page 5874)	This is Get__NewEnum, a member of class ISharedPropertyGroupManager.
	Get_Group (see page 5875)	This is Get_Group, a member of class ISharedPropertyGroupManager.

Public Properties

	Name	Description
R	_NewEnum (see page 5875)	This is _NewEnum, a member of class ISharedPropertyGroupManager.
R	Group (see page 5875)	This is Group, a member of class ISharedPropertyGroupManager.

57.1.9.2 ISharedPropertyGroupManager Methods

The methods of the ISharedPropertyGroupManager class are listed here.

Public Methods

	Name	Description
	CreatePropertyGroup (see page 5874)	This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.
	Get__NewEnum (see page 5874)	This is Get__NewEnum, a member of class ISharedPropertyGroupManager.
	Get_Group (see page 5875)	This is Get_Group, a member of class ISharedPropertyGroupManager.

57.1.9.2.1 ISharedPropertyGroupManager.CreatePropertyGroup Method

Pascal

```
function CreatePropertyGroup(const Name: WideString; var LockMode: Integer; var RelMode:
Integer; out fExists: WordBool): ISharedPropertyGroup; safecall;
```

C++

```
__fastcall ISharedPropertyGroup CreatePropertyGroup(const BSTR Name, int LockMode, int
RelMode, WordBool fExists);
```

Description

This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.

57.1.9.2.2 ISharedPropertyGroupManager.Get__NewEnum Method

Pascal

```
function Get__NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown Get__NewEnum();
```

Description

This is Get__NewEnum, a member of class ISharedPropertyGroupManager.

57.1.9.2.3 ISharedPropertyGroupManager.Get_Group Method

Pascal

```
function Get_Group(const Name: WideString): ISharedPropertyGroup; safecall;
```

C++

```
__fastcall ISharedPropertyGroup Get_Group(const BSTR Name);
```

Description

This is Get_Group, a member of class ISharedPropertyGroupManager.

57.1.9.3 ISharedPropertyGroupManager Properties

The properties of the ISharedPropertyGroupManager class are listed here.

Public Properties

	Name	Description
	_NewEnum (see page 5875)	This is _NewEnum, a member of class ISharedPropertyGroupManager.
	Group (see page 5875)	This is Group, a member of class ISharedPropertyGroupManager.

57.1.9.3.1 ISharedPropertyGroupManager._NewEnum Property

Pascal

```
property _NewEnum: IUnknown;
```

C++

```
__property IUnknown _NewEnum;
```

Description

This is _NewEnum, a member of class ISharedPropertyGroupManager.

57.1.9.3.2 ISharedPropertyGroupManager.Group Property

Pascal

```
property Group [const Name: WideString]: ISharedPropertyGroup;
```

C++

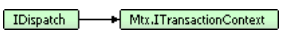
```
__property ISharedPropertyGroup Group[BSTR const Name];
```

Description

This is Group, a member of class ISharedPropertyGroupManager.

57.1.10 ITransactionContext Interface

Class Hierarchy



Pascal

```
ITransactionContext = interface(IDispatch);
```

C++

```
__interface ITransactionContext : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ITransactionContext.

57.1.10.1 ITransactionContext Members

The following tables list the members exposed by ITransactionContext.

Public Methods

	Name	Description
	Abort (see page 5876)	This is Abort, a member of class ITransactionContext.
	Commit (see page 5876)	This is Commit, a member of class ITransactionContext.
	CreateInstance (see page 5877)	This is CreateInstance, a member of class ITransactionContext.

57.1.10.2 ITransactionContext Methods

The methods of the ITransactionContext class are listed here.

Public Methods

	Name	Description
	Abort (see page 5876)	This is Abort, a member of class ITransactionContext.
	Commit (see page 5876)	This is Commit, a member of class ITransactionContext.
	CreateInstance (see page 5877)	This is CreateInstance, a member of class ITransactionContext.

57.1.10.2.1 ITransactionContext.Abort Method

Pascal

```
procedure Abort; safecall;
```

C++

```
__fastcall Abort();
```

Description

This is Abort, a member of class ITransactionContext.

57.1.10.2.2 ITransactionContext.Commit Method

Pascal

```
procedure Commit; safecall;
```

C++

```
__fastcall Commit();
```

Description

This is Commit, a member of class ITransactionContext.

57.1.10.2.3 ITransactionContext.CreateInstance Method

Pascal

```
function CreateInstance(const pszProgId: WideString): OleVariant; safecall;
```

C++

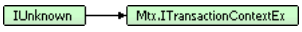
```
__fastcall OleVariant CreateInstance(const BSTR pszProgId);
```

Description

This is CreateInstance, a member of class ITransactionContext.

57.1.11 ITransactionContextEx Interface

Class Hierarchy



Pascal

```
ITransactionContextEx = interface(IUnknown);
```

C++

```
__interface ITransactionContextEx : public IUnknown;
```

File

Mtx

Description

This is class Mtx.ITransactionContextEx.

57.1.11.1 ITransactionContextEx Members

The following tables list the members exposed by ITransactionContextEx.

Public Methods

	Name	Description
	Abort (see page 5878)	This is Abort, a member of class ITransactionContextEx.
	Commit (see page 5878)	This is Commit, a member of class ITransactionContextEx.
	CreateInstance (see page 5878)	This is CreateInstance, a member of class ITransactionContextEx.

57.1.11.2 ITransactionContextEx Methods

The methods of the ITransactionContextEx class are listed here.

Public Methods

	Name	Description
	Abort (see page 5878)	This is Abort, a member of class ITransactionContextEx.
	Commit (see page 5878)	This is Commit, a member of class ITransactionContextEx.
	CreateInstance (see page 5878)	This is CreateInstance, a member of class ITransactionContextEx.

57.1.11.2.1 ITransactionContextEx.Abort Method

Pascal

```
function Abort: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Abort();
```

Description

This is Abort, a member of class ITransactionContextEx.

57.1.11.2.2 ITransactionContextEx.Commit Method

Pascal

```
function Commit: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Commit();
```

Description

This is Commit, a member of class ITransactionContextEx.

57.1.11.2.3 ITransactionContextEx.CreateInstance Method

Pascal

```
function CreateInstance(const cid: TGUID; const rid: TGUID; out pv): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const TGUID cid, const TGUID rid,  pv);
```

Description

This is CreateInstance, a member of class ITransactionContextEx.

57.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateSharedPropertyGroup (see page 5879)	Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.
	CreateSharedPropertyGroupManager (see page 5880)	Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.
	CreateTransactionContext (see page 5880)	Starts an MTS transaction.
	CreateTransactionContextEx (see page 5881)	Instantiates a TransactionContextEx object, which marks the beginning of a transaction.
	GetObjectContext (see page 5882)	Fetches the IObjectContext (see page 5862) interface for the current thread.
	SafeRef (see page 5882)	Returns a reference to a transactional object that is safe to pass outside its context.

57.2.1 Mtx.CreateSharedPropertyGroup Function

Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.

Pascal

```
function CreateSharedPropertyGroup(const Name: WideString): ISharedPropertyGroup;
```

C++

```
ISharedPropertyGroup CreateSharedPropertyGroup(const BSTR Name);
```

File

Mtx

Description

CreateSharedPropertyGroup returns an interface to a new shared property group. If a property group of the specified name exists, it returns a reference to the existing shared property group. Use this function to create and access the shared properties of a shared property group.

CreateSharedPropertyGroup sets up a SharedPropertyGroup with the default values of LockSetGet (see page 5885) for locking (isolation) mode, and Process (see page 5889) for release mode. LockSetGet (see page 5885) locks a property during every get or set operation on a shared property, making each operation atomic. This ensures that two clients cannot read or write to the same property at the same time, but it does not prevent other clients from concurrently accessing other properties in the same group. The Process (see page 5889) release mode ensures that the property group is not destroyed until the process in which it was created has terminated. (Objects that hold references on a property group must still call Release on their references).

To override these defaults, call CreateSharedPropertyGroupManager (see page 5880) and use the resulting interface to create a property group.

ISharedPropertyGroup (see page 5871) has several methods that can be used to share state among multiple objects in a server process. See ISharedPropertyGroup (see page 5871) in the MTS documentation for a list of methods.

By using shared property groups, you eliminate name collisions because they establish unique name spaces for the shared

properties they contain. The Shared Property Manager also implements locks and semaphores to protect shared properties from simultaneous access, which could result in lost updates and could leave the properties in an unpredictable state.

Note: Only objects running in the same process can share properties. If you want instances of different components to share properties, you have to install the components in the same MTS package. Because there is a risk that administrators will move components from one package to another, it's safest to limit the use of a shared property group to instances of components that are defined in the same DLL.

See Also

CreateSharedPropertyGroupManager

57.2.2 Mtx.CreateSharedPropertyGroupManager Function

Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.

Pascal

```
function CreateSharedPropertyGroupManager: ISharedPropertyGroupManager;
```

C++

```
ISharedPropertyGroupManager CreateSharedPropertyGroupManager();
```

File

Mtx

Description

Call CreateSharedPropertyGroupManager before creating shared property groups if you are not using the CreateSharedPropertyGroup (see page 5879) function. The ISharedPropertyGroupManager (see page 5873) interface lets you create and access shared properties with more control that is provided by CreateSharedPropertyGroup (see page 5879).

By using shared property groups, you eliminate name collisions because they establish unique name spaces for the shared properties they contain. The Shared Property Manager also implements locks and semaphores to protect shared properties from simultaneous access, which could result in lost updates and could leave the properties in an unpredictable state.

For more information about the ISharedPropertyGroupManager (see page 5873) interface, see the MTS documentation.

See Also

CreateSharedPropertyGroup

57.2.3 Mtx.CreateTransactionContext Function

Starts an MTS transaction.

Pascal

```
function CreateTransactionContext: ITransactionContext;
```

C++

```
ITransactionContext CreateTransactionContext();
```

File

Mtx

Description

CreateTransactionContext allows client applications to control MTS transactions. CreateTransactionContext returns an interface to the new transaction context. Use this interface to enlist MTS server objects in the transaction, and to commit or roll back the transaction when done.

See Also

CreateTransactionContextEx

TMtsAutoObject

TMtsAutoObject

TMtsDataModule

TMtsDataModule

57.2.4 Mtx.CreateTransactionContextEx Function

Instantiates a TransactionContextEx object, which marks the beginning of a transaction.

Pascal

```
function CreateTransactionContextEx: ITransactionContextEx;
```

C++

```
ITransactionContextEx CreateTransactionContextEx();
```

File

Mtx

Description

MTS base clients use CreateTransactionEx to combine the work of one or more MTS objects into an atomic transaction and to commit or abort the transaction. Clients initiate a transaction by instantiating a CreateTransactionEx object. Clients end a transaction by calling Complete or Abort on the object.

Method	Description
Abort	Aborts the work of all MTS objects participating in the current transaction. The transaction is completed on return from this method.
Commit	Attempts to commit the work of all MTS objects participating in the current transaction. If any of the MTS objects participating in the transaction call SetAbort or DisableCommit, or if a system error occurs, the transaction is aborted. Otherwise, the transaction is committed. In either case, the transaction is completed on return from this method.
CreateInstance	Instantiates another MTS object. If the component that provides the object is configured to support or require a transaction, then the new object runs under the transaction of the TransactionContextEx object.

See Also

CreateTransactionContext

TMtsAutoObject

TMtsAutoObject

TMtsDataModule

TMtsDataModule

57.2.5 Mtx.GetObjectContext Function

Fetches the IObjectContext (see page 5862) interface for the current thread.

Pascal

```
function GetObjectContext: IObjectContext;
```

C++

```
IObjectContext GetObjectContext();
```

File

Mtx

Description

When a COM object is created as part of the MTS environment or of a COM+ package (that is, a transactional object), a corresponding context object is also created. This context object exists for the entire lifetime of its transactional object, across one or more reactivation cycles. The object context keeps track of the object during deactivation. Call GetObjectContext to obtain an IObjectContext (see page 5862) interface pointer for the context object associated with the current thread.

Pass a pointer that receives the IObjectContext (see page 5862) interface pointer as ppInstanceContext.

GetObjectContext returns S_OK if it returns a pointer to the context object. Otherwise, it returns an error code.

Note: C++Builder automatically fetches the IObjectContext (see page 5862) interface when a transactional object is activated. The interface pointer is stored as the m_spObjectContext member of the transactional object's class.

Warning: When writing a transactional object that uses VCL classes, use the Get_ObjectContext method of TMtsDll instead. ComSvcs.h defines GetObjectContext using a macro so that it calls COM+ or MTS, depending on the system you use to build the transactional object. This macro interferes with the GetObjectContext method of some VCL classes, and so it is undefined when you include the VCL headers.

See Also

TMtsDll

57.2.6 Mtx.SafeRef Function

Returns a reference to a transactional object that is safe to pass outside its context.

Pascal

```
function SafeRef(const rid: TGUID; Unk: IUnknown): Pointer;
```

C++

```
void * SafeRef(const TGUID rid, IUnknown Unk);
```

File

Mtx

Description

SafeRef encapsulates the SafeRef MTS function that passes in

a reference to the interface ID of the interface (rid) that the current object wants to pass to another object or client

and

a reference to an interface on the current object (pUnk).

SafeRef returns a pointer to the interface specified in the rid parameter that is safe to pass outside the current object's context.

When using COM+, it is always safe to pass a self-reference to a client or other object. Thus, when you call SafeRef in an application that is running under a version of Windows that implemnts COM+, SafeRef simply returns the pointer passed as pUnk. Thus, it is not necessary to call SafeRef for COM+ applications. It will, however, do no harm in an application that is used under both MTS and COM+.

Under MTS, and should always call SafeRef before passing a self-reference to a client or other object and then pass the reference returned by this call. Under MTS, it is not safe for an object to pass a self pointer (Delphi) or this pointer (C++), or a self-reference obtained through an internal call to QueryInterface, to a client or to any other object. Once such a reference is passed outside the object's context, it is no longer a valid reference.

Calling SafeRef on a reference that is already safe returns the safe reference unchanged, except that the reference count on the interface is incremented.

When a client calls QueryInterface on a reference that is safe, MTS automatically ensures that the reference returned to the client is also a safe reference.

An object that obtains a safe reference must release the safe reference when it is finished with it.

Under MTS, safe references have different pointer values than their unsafe counterparts. For example, self or this and the safe version of self or this do not have the same value. It is important to be aware of this when testing whether two pointers refer to the same object. Calling QueryInterface for IID_IUnknown on each of the pointers and comparing the value of the returned pointers may result in the wrong conclusion. It is possible that both pointers refer to the same object, but that one is a safe reference and the other is not. If both references are safe references, they can be compared in the usual way. This is only a consideration for MTS objects, because clients should never have access to unsafe references.

Note: When writing a transactional object that may be installed under either MTS or COM+, use the SafeRef method of TMtsDll instead. TMtsDll automatically checks the current environment and returns the appropriate reference.

See Also

TMtsAutoObject

TMTSDll

57.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CLASS_SharedPropertyGroupManager (see page 5884)	This is constant Mtx.CLASS_SharedPropertyGroupManager.
CLASS_TransactionContext (see page 5884)	This is constant Mtx.CLASS_TransactionContext.
CLASS_TransactionContextEx (see page 5885)	This is constant Mtx.CLASS_TransactionContextEx.
LockMethod (see page 5885)	This is constant Mtx.LockMethod.
LockSetGet (see page 5885)	This is constant Mtx.LockSetGet.
mtsErrCtxAborted (see page 5886)	This is constant Mtx.mtsErrCtxAborted.
mtsErrCtxAborting (see page 5886)	This is constant Mtx.mtsErrCtxAborting.
mtsErrCtxActivityTimeout (see page 5886)	This is constant Mtx.mtsErrCtxActivityTimeout.
mtsErrCtxNoContext (see page 5887)	This is constant Mtx.mtsErrCtxNoContext.
mtsErrCtxNoSecurity (see page 5887)	This is constant Mtx.mtsErrCtxNoSecurity.
mtsErrCtxNotRegistered (see page 5887)	This is constant Mtx.mtsErrCtxNotRegistered.
mtsErrCtxOldReference (see page 5888)	This is constant Mtx.mtsErrCtxOldReference.
mtsErrCtxRoleNotFound (see page 5888)	This is constant Mtx.mtsErrCtxRoleNotFound.
mtsErrCtxTMNotAvailable (see page 5888)	This is constant Mtx.mtsErrCtxTMNotAvailable.
mtsErrCtxWrongThread (see page 5889)	This is constant Mtx.mtsErrCtxWrongThread.
Process (see page 5889)	This is constant Mtx.Process.
Standard (see page 5889)	This is constant Mtx.Standard.

57.3.1 Mtx.CLASS_SharedPropertyGroupManager Constant

Pascal

```
CLASS_SharedPropertyGroupManager: TGUID = '{2A005C11-A5DE-11CF-9E66-00AA00A3F464}';
```

C++

```
TGUID CLASS_SharedPropertyGroupManager = '{2A005C11-A5DE-11CF-9E66-00AA00A3F464}';
```

File

Mtx

Description

This is constant Mtx.CLASS_SharedPropertyGroupManager.

57.3.2 Mtx.CLASS_TransactionContext Constant

Pascal

```
CLASS_TransactionContext: TGUID = '{7999FC25-D3C6-11CF-ACAB-00A024A55AEF}';
```

C++

```
TGUID CLASS_TransactionContext = '{7999FC25-D3C6-11CF-ACAB-00A024A55AEF}';
```

File

Mtx

Description

This is constant Mtx.CLASS_TransactionContext.

57.3.3 Mtx.CLASS_TransactionContextEx Constant

Pascal

```
CLASS_TransactionContextEx: TGUID = '{5CB66670-D3D4-11CF-ACAB-00A024A55AEF}';
```

C++

```
TGUID CLASS_TransactionContextEx = '{5CB66670-D3D4-11CF-ACAB-00A024A55AEF}';
```

File

Mtx

Description

This is constant Mtx.CLASS_TransactionContextEx.

57.3.4 Mtx.LockMethod Constant

Pascal

```
LockMethod = $00000001;
```

C++

```
LockMethod = 0x00000001;
```

File

Mtx

Description

This is constant Mtx.LockMethod.

57.3.5 Mtx.LockSetGet Constant

Pascal

```
LockSetGet = $00000000;
```

C++

```
LockSetGet = 0x00000000;
```

File

Mtx

Description

This is constant Mtx.LockSetGet.

57.3.6 Mtx.mtsErrCtxAborted Constant

Pascal

```
mtsErrCtxAborted = $8004E002;
```

C++

```
mtsErrCtxAborted = 0x8004E002;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxAborted.

57.3.7 Mtx.mtsErrCtxAborting Constant

Pascal

```
mtsErrCtxAborting = $8004E003;
```

C++

```
mtsErrCtxAborting = 0x8004E003;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxAborting.

57.3.8 Mtx.mtsErrCtxActivityTimeout Constant

Pascal

```
mtsErrCtxActivityTimeout = $8004E006;
```

C++

```
mtsErrCtxActivityTimeout = 0x8004E006;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxActivityTimeout.

57.3.9 Mtx.mtsErrCtxNoContext Constant

Pascal

```
mtsErrCtxNoContext = $8004E004;
```

C++

```
mtsErrCtxNoContext = 0x8004E004;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNoContext.

57.3.10 Mtx.mtsErrCtxNoSecurity Constant

Pascal

```
mtsErrCtxNoSecurity = $8004E00D;
```

C++

```
mtsErrCtxNoSecurity = 0x8004E00D;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNoSecurity.

57.3.11 Mtx.mtsErrCtxNotRegistered Constant

Pascal

```
mtsErrCtxNotRegistered = $8004E005;
```

C++

```
mtsErrCtxNotRegistered = 0x8004E005;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNotRegistered.

57.3.12 Mtx.mtsErrCtxOldReference Constant

Pascal

```
mtsErrCtxOldReference = $8004E007;
```

C++

```
mtsErrCtxOldReference = 0x8004E007;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxOldReference.

57.3.13 Mtx.mtsErrCtxRoleNotFound Constant

Pascal

```
mtsErrCtxRoleNotFound = $8004E00C;
```

C++

```
mtsErrCtxRoleNotFound = 0x8004E00C;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxRoleNotFound.

57.3.14 Mtx.mtsErrCtxTMNotAvailable Constant

Pascal

```
mtsErrCtxTMNotAvailable = $8004E00F;
```

C++

```
mtsErrCtxTMNotAvailable = 0x8004E00F;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxTMNotAvailable.

57.3.15 Mtx.mtsErrCtxWrongThread Constant

Pascal

```
mtsErrCtxWrongThread = $8004E00E;
```

C++

```
mtsErrCtxWrongThread = 0x8004E00E;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxWrongThread.

57.3.16 Mtx.Process Constant

Pascal

```
Process = $00000001;
```

C++

```
Process = 0x00000001;
```

File

Mtx

Description

This is constant Mtx.Process.

57.3.17 Mtx.Standard Constant

Pascal

```
Standard = $00000000;
```

C++

```
Standard = 0x00000000;
```

File

Mtx

Description

This is constant Mtx.Standard.

58 ObjAuto Namespace

This is namespace ObjAuto.

Topics

Name	Description
Interfaces (see page 5893)	The following table lists interfaces in this documentation.
Functions (see page 5895)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5898)	The following table lists structs, records, enums in this documentation.
Types (see page 5901)	The following table lists types in this documentation.
Constants (see page 5905)	The following table lists constants in this documentation.

Constants

Name	Description
paEAX (see page 5905)	This is constant ObjAuto.paEAX.
paECX (see page 5905)	This is constant ObjAuto.paECX.
paEDX (see page 5906)	This is constant ObjAuto.paEDX.
paStack (see page 5906)	This is constant ObjAuto.paStack.

Functions

	Name	Description
	CreateMethodPointer (see page 5895)	This is function ObjAuto.CreateMethodPointer.
	CreateMethodPointer (see page 5895)	This is function ObjAuto.CreateMethodPointer.
	GetInvokeInstance (see page 5896)	This is function ObjAuto.GetInvokeInstance.
	GetMethodInfo (see page 5896)	This is function ObjAuto.GetMethodInfo.
	GetMethods (see page 5896)	This is function ObjAuto.GetMethods.
	ObjectInvoke (see page 5897)	This is function ObjAuto.ObjectInvoke.
	ReleaseMethodPointer (see page 5897)	This is function ObjAuto.ReleaseMethodPointer.

Interfaces

	Name	Description
	IMethodHandler (see page 5893)	This is class ObjAuto.IMethodHandler.

Structs, Records, Enums

	Name	Description
	TCallingConvention (see page 5898)	This is record ObjAuto.TCallingConvention.
	TMethodInfoHeader (see page 5898)	This is record ObjAuto.TMethodInfoHeader.

	TParameters (see page 5899)	This is record ObjAuto.TParameters.
	TParamInfo (see page 5899)	This is record ObjAuto.TParamInfo.
	TReturnInfo (see page 5900)	This is record ObjAuto.TReturnInfo.

Types

Name	Description
PMethodInfoHeader (see page 5901)	This is type ObjAuto.PMethodInfoHeader.
PParameters (see page 5901)	This is type ObjAuto.PParameters.
PParamInfo (see page 5902)	This is type ObjAuto.PParamInfo.
PPointer (see page 5902)	This is type ObjAuto.PPointer.
PReturnInfo (see page 5902)	This is type ObjAuto.PReturnInfo.
PWord (see page 5903)	This is type ObjAuto.PWord.
TDynamicInvokeEvent (see page 5903)	This is type ObjAuto.TDynamicInvokeEvent.
TMethodInfoArray (see page 5903)	This is type ObjAuto.TMethodInfoArray.
TParamFlags (see page 5904)	This is type ObjAuto.TParamFlags.

58.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IMethodHandler (see page 5893)	This is class ObjAuto.IMethodHandler.

58.1.1 IMethodHandler Interface

Class Hierarchy

ObjAuto.IMethodHandler

Pascal

```
IMethodHandler = interface;
```

C++

```
__interface IMethodHandler;
```

File

ObjAuto

Description

This is class ObjAuto.IMethodHandler.

58.1.1.1 IMethodHandler Members

The following tables list the members exposed by IMethodHandler.

Public Methods

	Name	Description
	Execute (see page 5894)	This is Execute, a member of class IMethodHandler.
	InstanceToVariant (see page 5894)	This is InstanceToVariant, a member of class IMethodHandler.

58.1.1.2 IMethodHandler Methods

The methods of the IMethodHandler class are listed here.

Public Methods

	Name	Description
	Execute (see page 5894)	This is Execute, a member of class IMethodHandler.
	InstanceToVariant (see page 5894)	This is InstanceToVariant, a member of class IMethodHandler.

58.1.1.2.1 IMethodHandler.Execute Method

Pascal

```
function Execute(const Args: array of Variant): Variant;
```

C++

```
__fastcall Variant Execute(const array of Variant Args);
```

Description

This is Execute, a member of class IMethodHandler.

58.1.1.2.2 IMethodHandler.InstanceToVariant Method

Pascal

```
function InstanceToVariant(Instance: TObject): Variant;
```

C++

```
__fastcall Variant InstanceToVariant(TObject * Instance);
```

Description

This is InstanceToVariant, a member of class IMethodHandler.

58.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateMethodPointer (see page 5895)	This is function ObjAuto.CreateMethodPointer.
	CreateMethodPointer (see page 5895)	This is function ObjAuto.CreateMethodPointer.
	GetInvokeInstance (see page 5896)	This is function ObjAuto.GetInvokeInstance.
	GetMethodInfo (see page 5896)	This is function ObjAuto.GetMethodInfo.
	GetMethods (see page 5896)	This is function ObjAuto.GetMethods.
	ObjectInvoke (see page 5897)	This is function ObjAuto.ObjectInvoke.
	ReleaseMethodPointer (see page 5897)	This is function ObjAuto.ReleaseMethodPointer.

58.2.1 ObjAuto.CreateMethodPointer Function

Pascal

```
function CreateMethodPointer(const MethodHandler: IMethodHandler; TypeData: PTypeData):
TMethod; overload;
```

C++

```
TMethod CreateMethodPointer(const IMethodHandler MethodHandler, PTypeData TypeData);
```

File

ObjAuto

Description

This is function ObjAuto.CreateMethodPointer.

58.2.2 ObjAuto.CreateMethodPointer Function

Pascal

```
function CreateMethodPointer(const ADynamicInvokeEvent: TDynamicInvokeEvent; TypeData:
PTypeData): TMethod; overload;
```

C++

```
TMethod CreateMethodPointer(const TDynamicInvokeEvent ADynamicInvokeEvent, PTypeData TypeData);
```

File

ObjAuto

Description

This is function ObjAuto.CreateMethodPointer.

58.2.3 ObjAuto.GetInvokeInstance Function

Pascal

```
function GetInvokeInstance(MethodPointer: TMethod): TObject;
```

C++

```
TObject * GetInvokeInstance(TMethod MethodPointer);
```

File

ObjAuto

Description

This is function ObjAuto.GetInvokeInstance.

58.2.4 ObjAuto.GetMethodInfo Function

Pascal

```
function GetMethodInfo(Instance: TObject; const MethodName: ShortString): PMethodInfoHeader;
```

C++

```
PMethodInfoHeader GetMethodInfo(TObject * Instance, const ShortString MethodName);
```

File

ObjAuto

Description

This is function ObjAuto.GetMethodInfo.

58.2.5 ObjAuto.GetMethods Function

Pascal

```
function GetMethods(ClassType: TClass): TMethodInfoArray;
```

C++

```
TMethodInfoArray GetMethods(TClass ClassType);
```

File

ObjAuto

Description

This is function ObjAuto.GetMethods.

58.2.6 ObjAuto.ObjectInvoke Function

Pascal

```
function ObjectInvoke(Instance: TObject; MethodHeader: PMethodInfoHeader; const ParamIndexes:
array of Integer; const Params: array of Variant): Variant;
```

C++

```
Variant ObjectInvoke(TObject * Instance, PMethodInfoHeader MethodHeader, const array of
Integer ParamIndexes, const array of Variant Params);
```

File

ObjAuto

Description

This is function ObjAuto.ObjectInvoke.

58.2.7 ObjAuto.ReleaseMethodPointer Function

Pascal

```
procedure ReleaseMethodPointer(MethodPointer: TMethod);
```

C++

```
ReleaseMethodPointer(TMethod MethodPointer);
```

File

ObjAuto

Description

This is function ObjAuto.ReleaseMethodPointer.

58.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCallingConvention (see page 5898)	This is record ObjAuto.TCallingConvention.

Records

	Name	Description
	TMethodInfoHeader (see page 5898)	This is record ObjAuto.TMethodInfoHeader.
	TParameters (see page 5899)	This is record ObjAuto.TParameters.
	TParamInfo (see page 5899)	This is record ObjAuto.TParamInfo.
	TReturnInfo (see page 5900)	This is record ObjAuto.TReturnInfo.

58.3.1 ObjAuto.TCallingConvention Enumeration

Pascal

```
TCallingConvention = (
  ccRegister,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
);
```

C++

```
enum TCallingConvention {
  ccRegister,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
};
```

File

ObjAuto

Description

This is record ObjAuto.TCallingConvention.

58.3.2 ObjAuto.TMethodInfoHeader Record

Pascal

```
TMethodInfoHeader = packed record
  Len: Word;
  Addr: Pointer;
```

```
    Name: ShortString;
  end;
```

C++

```
  struct TMethodInfoHeader {
    Word Len;
    void * Addr;
    ShortString Name;
  };
```

File

ObjAuto

Description

This is record ObjAuto.TMethodInfoHeader.

58.3.3 ObjAuto.TParameters Record

Pascal

```
  TParameters = packed record
    Registers: array[paEDX..paECX] of Cardinal;
    Stack: array[0..1023] of Byte;
  end;
```

C++

```
  struct TParameters {
    array[paEDX..paECX] of Cardinal Registers;
    array[0..1023] of Byte Stack;
  };
```

File

ObjAuto

Description

This is record ObjAuto.TParameters.

58.3.4 ObjAuto.TParamInfo Record

Pascal

```
  TParamInfo = packed record
    Flags: TParamFlags;
    ParamType: ^PTypeInfo;
    Access: Word;
    Name: ShortString;
  end;
```

C++

```
  struct TParamInfo {
    TParamFlags Flags;
    ^PTypeInfo ParamType;
    Word Access;
    ShortString Name;
  };
```

File

ObjAuto

Description

This is record ObjAuto.TParamInfo.

58.3.5 ObjAuto.TReturnInfo Record

Pascal

```
TReturnInfo = packed record
  Version: Byte;
  CallingConvention: TCallingConvention;
  ReturnType: ^PTypeInfo;
  ParamSize: Word;
end;
```

C++

```
struct TReturnInfo {
  Byte Version;
  TCallingConvention CallingConvention;
  ^PTypeInfo ReturnType;
  Word ParamSize;
};
```

File

ObjAuto

Description

This is record ObjAuto.TReturnInfo.

58.4 Types

The following table lists types in this documentation.

Types

Name	Description
PMethodInfoHeader (see page 5901)	This is type ObjAuto.PMethodInfoHeader.
PParameters (see page 5901)	This is type ObjAuto.PParameters.
PParamInfo (see page 5902)	This is type ObjAuto.PParamInfo.
PPointer (see page 5902)	This is type ObjAuto.PPointer.
PReturnInfo (see page 5902)	This is type ObjAuto.PReturnInfo.
PWord (see page 5903)	This is type ObjAuto.PWord.
TDynamicInvokeEvent (see page 5903)	This is type ObjAuto.TDynamicInvokeEvent.
TMethodInfoArray (see page 5903)	This is type ObjAuto.TMethodInfoArray.
TParamFlags (see page 5904)	This is type ObjAuto.TParamFlags.

58.4.1 ObjAuto.PMethodInfoHeader Type

Pascal

```
PMethodInfoHeader = ^TMethodInfoHeader;
```

C++

```
^TMethodInfoHeader PMethodInfoHeader;
```

File

ObjAuto

Description

This is type ObjAuto.PMethodInfoHeader.

58.4.2 ObjAuto.PParameters Type

Pascal

```
PParameters = ^TParameters;
```

C++

```
^TParameters PParameters;
```

File

ObjAuto

Description

This is type ObjAuto.PParameters.

58.4.3 ObjAuto.PParamInfo Type

Pascal

```
PParamInfo = ^TParamInfo;
```

C++

```
^TParamInfo PParamInfo;
```

File

ObjAuto

Description

This is type ObjAuto.PParamInfo.

58.4.4 ObjAuto.PPointer Type

Pascal

```
PPointer = ^Pointer;
```

C++

```
^Pointer PPointer;
```

File

ObjAuto

Description

This is type ObjAuto.PPointer.

58.4.5 ObjAuto.PReturnInfo Type

Pascal

```
PReturnInfo = ^TReturnInfo;
```

C++

```
^TReturnInfo PReturnInfo;
```

File

ObjAuto

Description

This is type ObjAuto.PReturnInfo.

58.4.6 ObjAuto.PWord Type

Pascal

```
PWord = ^Word;
```

C++

```
^Word PWord;
```

File

ObjAuto

Description

This is type ObjAuto.PWord.

58.4.7 ObjAuto.TDynamicInvokeEvent Type

Pascal

```
TDynamicInvokeEvent = procedure (Params: PParameters; StackSize: Integer) of object;
```

C++

```
(Params: PParameters; StackSize: Integer) ( TDynamicInvokeEvent)();
```

File

ObjAuto

Description

This is type ObjAuto.TDynamicInvokeEvent.

58.4.8 ObjAuto.TMethodInfoArray Type

Pascal

```
TMethodInfoArray = array of PMethodInfoHeader;
```

C++

```
array of PMethodInfoHeader TMethodInfoArray;
```

File

ObjAuto

Description

This is type ObjAuto.TMethodInfoArray.

58.4.9 ObjAuto.TParamFlags Type

Pascal

```
TParamFlags = set of (pfVar, pfConst, pfArray, pfAddress, pfReference, pfOut, pfResult);
```

C++

```
(pfVar, pfConst, pfArray, pfAddress, pfReference, pfOut, pfResult) TParamFlags;
```

File

ObjAuto

Description

This is type ObjAuto.TParamFlags.

58.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
paEAX (see page 5905)	This is constant ObjAuto.paEAX.
paECX (see page 5905)	This is constant ObjAuto.paECX.
paEDX (see page 5906)	This is constant ObjAuto.paEDX.
paStack (see page 5906)	This is constant ObjAuto.paStack.

58.5.1 ObjAuto.paEAX Constant

Pascal

```
paEAX = Word(0);
```

C++

```
paEAX = Word(0);
```

File

ObjAuto

Description

This is constant ObjAuto.paEAX.

58.5.2 ObjAuto.paECX Constant

Pascal

```
paECX = Word(2);
```

C++

```
paECX = Word(2);
```

File

ObjAuto

Description

This is constant ObjAuto.paECX.

58.5.3 ObjAuto.paEDX Constant

Pascal

```
paEDX = Word(1);
```

C++

```
paEDX = Word(1);
```

File

ObjAuto

Description

This is constant ObjAuto.paEDX.

58.5.4 ObjAuto.paStack Constant

Pascal

```
paStack = Word(3);
```

C++

```
paStack = Word(3);
```

File

ObjAuto

Description

This is constant ObjAuto.paStack.

59 ObjBrkr Namespace

This is namespace ObjBrkr.

Topics

Name	Description
Classes (see page 5908)	The following table lists classes in this documentation.

Classes

	Name	Description
	EBrokerException (see page 5908)	EBrokerException is the exception raised by simple object brokers when they can't locate an available server.
	TServerCollection (see page 5908)	TServerCollection is a collection of TServerItem (see page 5912) objects.
	TServerItem (see page 5912)	TServerItem describes an application server for a simple object broker.
	TSimpleObjectBroker (see page 5916)	TSimpleObjectBroker maintains a list of available application servers for a connection component.

59.1 Classes

The following table lists classes in this documentation.

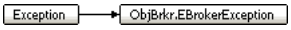
Classes

	Name	Description
	EBrokerException (see page 5908)	EBrokerException is the exception raised by simple object brokers when they can't locate an available server.
	TServerCollection (see page 5908)	TServerCollection is a collection of TServerItem (see page 5912) objects.
	TServerItem (see page 5912)	TServerItem describes an application server for a simple object broker.
	TSimpleObjectBroker (see page 5916)	TSimpleObjectBroker maintains a list of available application servers for a connection component.

59.1.1 EBrokerException Class

EBrokerException is the exception raised by simple object brokers when they can't locate an available server.

Class Hierarchy



Pascal

```
EBrokerException = class(Exception);
```

C++

```
class EBrokerException : public Exception;
```

File

ObjBrkr

Description

EBrokerException is raised when a simple object broker can't locate an available server.

See Also

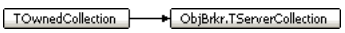
TSimpleObjectBroker (see page 5916)

TServerCollection (see page 5908)

59.1.2 TServerCollection Class

TServerCollection is a collection of TServerItem (see page 5912) objects.

Class Hierarchy



Pascal

```
TServerCollection = class(TOwnedCollection);
```

C++

```
class TServerCollection : public TOwnedCollection;
```

File

ObjBrkr

Description

TSimpleObjectBroker (see page 5916) uses TServerCollection to represent the set of available application servers to which a client application can connect. Use the properties and methods of TServerCollection to manage the set of servers. For example, you can add items as new server systems come on line, disable server items when the corresponding system is temporarily out of commission, and so on.

See Also

TServerItem (see page 5912)

TSimpleObjectBroker (see page 5916)

59.1.2.1 TServerCollection Members

The following tables list the members exposed by TServerCollection.

Public Methods

	Name	Description
	Create (see page 5909)	Creates and initializes a TServerCollection (see page 5908) object.
	FindServer (see page 5910)	Returns the server from the Items (see page 5911) property array that has a specified computer name.
	GetBalancedName (see page 5910)	Returns the ComputerName property of a randomly chosen server from the Items (see page 5911) property array.
	GetNextName (see page 5911)	This is GetNextName, a member of class TServerCollection.

Public Properties

	Name	Description
	Items (see page 5911)	Lists the TServerItem (see page 5912) objects in the collection.

59.1.2.2 TServerCollection.Create Constructor

Creates and initializes a TServerCollection (see page 5908) object.

Pascal

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

C++

```
__fastcall TServerCollection(TComponent * AOwner);
```

Description

Applications should not need to instantiate TServerCollection (see page 5908). The simple object broker that uses TServerCollection (see page 5908) to manage its list of available servers creates this object from its own constructor.

See Also

Servers (see page 5923)

59.1.2.3 TServerCollection Methods

The methods of the TServerCollection class are listed here.

Public Methods

	Name	Description
	FindServer (see page 5910)	Returns the server from the Items (see page 5911) property array that has a specified computer name.
	GetBalancedName (see page 5910)	Returns the ComputerName property of a randomly chosen server from the Items (see page 5911) property array.
	GetNextName (see page 5911)	This is GetNextName, a member of class TServerCollection.

59.1.2.3.1 TServerCollection.FindServer Method

Returns the server from the Items (see page 5911) property array that has a specified computer name.

Pascal

```
function FindServer(const ComputerName: string): TServerItem;
```

C++

```
__fastcall TServerItem FindServer(const UnicodeString ComputerName);
```

Description

Call FindServer to locate a server in the items property array given its ComputerName property. The ComputerName parameter specifies this computer name. Usually, this value is obtained by a previous call to GetBalancedName (see page 5910) or GetNextName (see page 5911).

Note: FindServer returns the first item it finds with the given ComputerName property. Unlike the GetBalancedName (see page 5910) and GetNextName (see page 5911) methods, it does not ignore servers that have failed or been disabled (HasFailed is true, or Enabled is false).

See Also

GetBalancedName (see page 5910)

getnextname (see page 5911)

ComputerName (see page 5914)

Items (see page 5911)

Enabled (see page 5915)

HasFailed (see page 5915)

59.1.2.3.2 TServerCollection.GetBalancedName Method

Returns the ComputerName property of a randomly chosen server from the Items (see page 5911) property array.

Pascal

```
function GetBalancedName: string;
```

C++

```
__fastcall UnicodeString GetBalancedName();
```

Description

Call GetBalancedName to get the name of an available server. Servers in the Items (see page 5911) property array are considered available when no connection attempts have failed (HasFailed is false) and their Enabled property has not been set to false. GetBalancedName selects an available TServerItem (see page 5912) at random from the Items (see page 5911) property array and returns its ComputerName property.

See Also

Items (see page 5911)

LoadBalanced (see page 5922)

ComputerName (see page 5914)

HasFailed (see page 5915)

Enabled (see page 5915)

getnextname (see page 5911)

59.1.2.3.3 TServerCollection.GetNextName Method

Pascal

```
function GetNextName: string;
```

C++

```
__fastcall UnicodeString GetNextName();
```

Description

This is GetNextName, a member of class TServerCollection.

59.1.2.4 TServerCollection Properties

The properties of the TServerCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 5911)	Lists the TServerItem (see page 5912) objects in the collection.

59.1.2.4.1 TServerCollection.Items Property

Lists the TServerItem (see page 5912) objects in the collection.

Pascal

```
property Items [Index: Integer]: TServerItem;
```

C++

```
__property TServerItem Items[int Index];
```

Description

Use Items to access individual TServerItem (see page 5912) objects maintained by TServerCollection (see page 5908).

Note: In Delphi, Items is the default property of TServerCollection (see page 5908), so the name of the Items property can be left off when using this property. Thus the line

```
TSimpleObjectBroker1.Servers.Items[I].Enabled := False;
```

can be written

```
TSimpleObjectBroker1.Servers[I].Enabled := False;
```

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Enabled (see page 5915)

GetBalancedName (see page 5910)

getnextname (see page 5911)

FindServer (see page 5910)

operator_[]

59.1.3 TServerItem Class

TServerItem describes an application server for a simple object broker.

Class Hierarchy

Pascal

```
TServerItem = class(TCollectionItem);
```

C++

```
class TServerItem : public TCollectionItem;
```

File

ObjBrkr

Description

Use TServerItem to describe an application server to which client applications can connect. TSimpleObjectBroker (see page 5916) maintains a list of these descriptions. Then, when a connection component needs to open a connection, it can use the information stored in a server item to identify an available application server.

See Also

TSimpleObjectBroker (see page 5916)

TRemoteDataModule

TDispatchConnection (see page 4796)

59.1.3.1 TServerItem Members

The following tables list the members exposed by TServerItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 5914)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Create (see page 5913)	Creates and initializes a TServerItem (see page 5912) instance.

Public Properties

	Name	Description
	HasFailed (see page 5915)	Indicates whether a connection attempt to the associated server machine has failed.

Published Properties

	Name	Description
	ComputerName (see page 5914)	Identifies the machine where the application server resides.
	Enabled (see page 5915)	Enables or disables the server item.
	Port (see page 5916)	Specifies the port number for the associated server application.

59.1.3.2 TServerItem.Create Constructor

Creates and initializes a TServerItem (see page 5912) instance.

Pascal

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

C++

```
virtual __fastcall TServerItem(TCollection AOwner);
```

Description

Do not call Create directly to add a new server item to a server collection. Instead, use the Add or Insert method of the TServerCollection (see page 5908) object.

The AOwner parameter specifies the server collection that manages the newly created server item.

After calling the inherited constructor, Create initializes the Enabled (see page 5915) property to true, the HasFailed (see page 5915) property to false, and the Port (see page 5916) property to 211.

See Also

Add

Insert

Port (see page 5916)

HasFailed (see page 5915)

Enabled (see page 5915)

59.1.3.3 TServerItem Methods

The methods of the TServerItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 5914)	Returns the name of the collection item as it appears in the collection editor.

59.1.3.3.1 TServerItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

59.1.3.4 TServerItem Properties

The properties of the TServerItem class are listed here.

Public Properties

	Name	Description
	HasFailed (see page 5915)	Indicates whether a connection attempt to the associated server machine has failed.

Published Properties

	Name	Description
	ComputerName (see page 5914)	Identifies the machine where the application server resides.
	Enabled (see page 5915)	Enables or disables the server item.
	Port (see page 5916)	Specifies the port number for the associated server application.

59.1.3.4.1 TServerItem.ComputerName Property

Identifies the machine where the application server resides.

Pascal

```
property ComputerName: string;
```

C++

```
__property UnicodeString ComputerName;
```

Description

Set ComputerName to a string that identifies the machine where the application server resides. The value of this string depends on the type of connection component that requests this value from its object broker.

If the connection component is a TDCOMConnection (see page 4790) object, ComputerName is the Windows computer name. (This value can be found in the Control panel under network settings.)

If the connection component is a TSocketConnection object, ComputerName is either the host name or IP address of the system running the application server. Socket connection components parse this string to determine which it is.

If the connection component is a TWebConnection or TSoapConnection object, ComputerName is the URL of the dispatcher (httpsrvr.dll or soapsrvr.dll (running on the system with the application server.

See Also

GetBalancedName (see page 5910)

getnextname (see page 5911)

Port (see page 5916)

computername (see page 4793)

Address

Host

ObjectBroker (see page 4806)

59.1.3.4.2 TServerItem.Enabled Property

Enables or disables the server item.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Set Enabled to indicate whether the server should made available in the simple object broker. When Enabled is true, the ComputerName (see page 5914) property can be supplied to connection components by the object broker's GetComputerForGUID or GetComputerForProgID methods. When Enabled is false, the server item is not returned.

Note: Even when the server item is disabled, it is still returned by TServerCollection (see page 5908)'s FindServer method.

See Also

GetComputerForGUID (see page 5919)

GetComputerForProgID (see page 5919)

HasFailed (see page 5915)

FindServer (see page 5910)

59.1.3.4.3 TServerItem.HasFailed Property

Indicates whether a connection attempt to the associated server machine has failed.

Pascal

```
property HasFailed: Boolean;
```

C++

```
__property Boolean HasFailed;
```

Description

When connection components can't successfully open a connection to an application server, they inform the object broker by calling SetConnectStatus. The simple object broker, in turn, records this fact by setting the HasFailed property to true.

When HasFailed is false, the ComputerName (see page 5914) property can be supplied to connection components by the object broker's GetComputerForGUID or GetComputerForProgID methods. When HasFailed is true, the server item is not returned.

Once a server has failed it is presumed to be down until the application explicitly resets the HasFailed property to false.

Do not use HasFailed to disable server items. Instead, use the Enabled (see page 5915) property to disable servers for any reason other than a failed connection.

Note: Even when HasFailed is true, the server item is still returned by TServerCollection (see page 5908)'s FindServer method.

See Also

SetConnectStatus (see page 5921)

Enabled (see page 5915)

GetComputerForGUID (see page 5919)

GetComputerForProgID (see page 5919)

FindServer (see page 5910)

59.1.3.4.4 TServerItem.Port Property

Specifies the port number for the associated server application.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Read Port to obtain a port number when using the server item to supply information for connecting using TCP/IP. This value is returned by the GetPortForComputer method of the simple object broker.

By default, the value of Port is 211, which is the default port number for the socket dispatcher application ScktSrvr.exe. If the server listens on a different port, change this property to indicate the listening port.

Note: Server items that supply a Port number should specify the server system's IP address or host name as the value of ComputerName (see page 5914).

See Also

ComputerName (see page 5914)

59.1.4 TSimpleObjectBroker Class

TSimpleObjectBroker maintains a list of available application servers for a connection component.

Class Hierarchy

Pascal

```
TSimpleObjectBroker = class(TCustomObjectBroker);
```

C++

```
class TSimpleObjectBroker : public TCustomObjectBroker;
```

File

ObjBrkr

Description

TSimpleObjectBroker maintains a list of servers, all of which run the application server for a multi-tiered application. When a connection component requests a server, the simple object broker passes the name of one of the available servers in the list. This process allows client applications to locate server systems dynamically at runtime. If there are redundant server systems, this allows client applications to continue functioning when a server goes down.

TSimpleObjectBroker can randomize the selection of servers from among the currently available servers in its list to provide a simple system of load balancing.

See Also

TDispatchConnection (see page 4796)

TServerCollection (see page 5908)

59.1.4.1 TSimpleObjectBroker Members

The following tables list the members exposed by TSimpleObjectBroker.

Public Methods

	Name	Description
	Create (see page 5918)	Creates and initializes a simple object broker.
	Destroy (see page 5918)	Destroys an instance of TSimpleObjectBroker (see page 5916).
	GetComputerForGUID (see page 5919)	Returns a string that identifies an available server machine.
	GetComputerForProgID (see page 5919)	Returns a string that identifies an available server machine.
	GetPortForComputer (see page 5920)	Returns a port number for connecting to a specified remote server.
	LoadFromStream (see page 5920)	Sets the value of the Servers (see page 5923) property from a stream.
	SaveToStream (see page 5921)	Saves the value of the Servers (see page 5923) property to a stream.
	SetConnectStatus (see page 5921)	Records whether a connection was established to a specified server.

Public Properties

	Name	Description
	ServerData (see page 5922)	Encapsulates the list of servers as an OleVariant.

Published Properties

	Name	Description
	LoadBalanced (see page 5922)	Specifies how the simple object broker picks a server from its list.
	Servers (see page 5923)	Maintains the list of servers that can be provided to connection components.

59.1.4.2 TSimpleObjectBroker.Create Constructor

Creates and initializes a simple object broker.

Pascal

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

C++

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

Description

Call Create to instantiate a simple object broker at runtime.

After calling the inherited constructor, Create initializes the LoadBalanced (see page 5922) property to false and creates the TServerCollection (see page 5908) that implements the Servers (see page 5923) property.

See Also

Create

Servers (see page 5923)

LoadBalanced (see page 5922)

59.1.4.3 TSimpleObjectBroker.Destroy Destructor

Destroys an instance of TSimpleObjectBroker (see page 5916).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleObjectBroker();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the simple object broker is not nil, and only then calls Destroy.

See Also

System::TObject::Free

59.1.4.4 TSimpleObjectBroker Methods

The methods of the TSimpleObjectBroker class are listed here.

Public Methods

	Name	Description
	GetComputerForGUID (see page 5919)	Returns a string that identifies an available server machine.
	GetComputerForProgID (see page 5919)	Returns a string that identifies an available server machine.
	GetPortForComputer (see page 5920)	Returns a port number for connecting to a specified remote server.
	LoadFromStream (see page 5920)	Sets the value of the Servers (see page 5923) property from a stream.
	SaveToStream (see page 5921)	Saves the value of the Servers (see page 5923) property to a stream.
	SetConnectStatus (see page 5921)	Records whether a connection was established to a specified server.

59.1.4.4.1 TSimpleObjectBroker.GetComputerForGUID Method

Returns a string that identifies an available server machine.

Pascal

```
function GetComputerForGUID(GUID: TGUID): string; override;
```

C++

```
virtual __fastcall UnicodeString GetComputerForGUID(TGUID GUID);
```

Description

Call GetComputerForGUID to obtain the name of an available server machine. The GUID parameter (which specifies the global unique identifier of the desired server interface) is ignored.

GetComputerForGUID returns the ComputerName property of an available server from the Servers (see page 5923) property. If the LoadBalanced (see page 5922) property is true, this server is chosen at random from available servers. If the LoadBalanced (see page 5922) property is false, this server is the first available server listed in the Servers (see page 5923) property. If no servers are available, GetComputerForGUID raises an EBrokerException (see page 5908) exception.

A server is considered available if its Enabled property is true and its HasFailed property is false. The broker automatically sets HasFailed to false when it is notified of a connection failure for that server.

See Also

GetComputerForProgID (see page 5919)

ComputerName (see page 5914)

LoadBalanced (see page 5922)

EBrokerException (see page 5908)

Enabled (see page 5915)

HasFailed (see page 5915)

SetConnectStatus (see page 5921)

59.1.4.4.2 TSimpleObjectBroker.GetComputerForProgID Method

Returns a string that identifies an available server machine.

Pascal

```
function GetComputerForProgID(const ProgID): string; override;
```

C++

```
virtual __fastcall UnicodeString GetComputerForProgID(const  ProgID);
```

Description

Call GetComputerForProgID to obtain the name of an available server machine. The ProgID parameter (which specifies the desired server interface) is ignored.

TSimpleObjectBroker (see page 5916) returns the ComputerName property of an available server from the Servers (see page 5923) property. If the LoadBalanced (see page 5922) property is true, this server is chosen at random from available servers. If the LoadBalanced (see page 5922) property is false, this server is the first available server listed in the Servers (see page 5923) property. If no servers are available, GetComputerForGUID (see page 5919) raises an EBrokerException (see page 5908) exception.

A server is considered available if its Enabled property is true and its HasFailed property is false. The broker automatically sets HasFailed to false when it is notified of a connection failure for that server.

See Also

GetComputerForGUID (see page 5919)

ComputerName (see page 5914)

LoadBalanced (see page 5922)

EBrokerException (see page 5908)

Enabled (see page 5915)

HasFailed (see page 5915)

SetConnectStatus (see page 5921)

59.1.4.4.3 TSimpleObjectBroker.GetPortForComputer Method

Returns a port number for connecting to a specified remote server.

Pascal

```
function GetPortForComputer(const ComputerName: string): Integer; override;
```

C++

```
virtual __fastcall int GetPortForComputer(const UnicodeString ComputerName);
```

Description

Socket connection components call GetPortForComputer to get a value for their Port property. The ComputerName parameter specifies a Host name or IP address that was returned by GetComputerForGUID (see page 5919) or GetComputerForProgID (see page 5919). This value is used to identify a server in the list maintained by the Servers (see page 5923) property.

There are no connection components that explicitly use a Port number except on the Windows product. GetPortForComputer is included in TCustomObjectBroker (see page 4787) to allow for the possibility of connection components that use an explicit Port number. The ComputerName parameter is a Host name or IP address that is returned by the GetComputerForGUID (see page 5919) or GetComputerForProgID (see page 5919) method. This value is used to identify a server in the list maintained by the Servers (see page 5923) property.

See Also

GetComputerForProgID (see page 5919)

GetComputerForGUID (see page 5919)

Port

ComputerName (see page 5914)

Port (see page 5916)

59.1.4.4.4 TSimpleObjectBroker.LoadFromStream Method

Sets the value of the Servers (see page 5923) property from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to load a list of available servers that has been previously saved using the SaveToStream (see page 5921) method.

See Also

SaveToStream (see page 5921)

ServerData (see page 5922)

59.1.4.4.5 TSimpleObjectBroker.SaveToStream Method

Saves the value of the Servers (see page 5923) property to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the list of available servers. The servers can be restored later by calling the LoadFromStream (see page 5920) method.

See Also

LoadFromStream (see page 5920)

ServerData (see page 5922)

59.1.4.4.6 TSimpleObjectBroker.SetConnectStatus Method

Records whether a connection was established to a specified server.

Pascal

```
procedure SetConnectStatus(ComputerName: string; Success: Boolean); override;
```

C++

```
virtual __fastcall SetConnectStatus(UnicodeString ComputerName, Boolean Success);
```

Description

After calling GetComputerForGUID (see page 5919) or GetComputerForProgID (see page 5919), connection components use the returned value to open a connection to the identified server. They then call SetConnectStatus to inform the broker of whether the connection was successfully opened.

SetConnectStatus sets the HasFailed property of the corresponding server to true if the connection was unsuccessful. It sets the HasFailed property to false if the connection was successful.

When the HasFailed property of a server is true, that server is ignored when the GetComputerForGUID (see page 5919) or GetComputerForProgID (see page 5919) method looks for available connections.

See Also

GetComputerForGUID (see page 5919)

GetComputerForProgID (see page 5919)

59.1.4.5 TSimpleObjectBroker Properties

The properties of the TSimpleObjectBroker class are listed here.

Public Properties

	Name	Description
	ServerData (see page 5922)	Encapsulates the list of servers as an OleVariant.

Published Properties

	Name	Description
	LoadBalanced (see page 5922)	Specifies how the simple object broker picks a server from its list.
	Servers (see page 5923)	Maintains the list of servers that can be provided to connection components.

59.1.4.5.1 TSimpleObjectBroker.LoadBalanced Property

Specifies how the simple object broker picks a server from its list.

Pascal

```
property LoadBalanced: Boolean;
```

C++

```
__property Boolean LoadBalanced;
```

Description

Set LoadBalanced to true to indicate that the simple object broker should supply computer names randomly from the available servers listed in the Servers (see page 5923) property. When LoadBalanced is false, GetComputerForGUID (see page 5919) and GetComputerForProgID (see page 5919) always return the first available server.

If the simple object broker is used by multiple connection components in the client application, it is preferable to set LoadBalanced to true in order to distribute the work among available servers.

When there is only a single connection component, there is no need to randomize the selection of servers, because only one connection will be used at a time. However, when deploying client applications, you may want to reorder the Servers (see page 5923) list in the simple object broker for each client, so that different clients give preference to different servers.

See Also

Servers (see page 5923)

GetComputerForGUID (see page 5919)

GetComputerForProgID (see page 5919)

59.1.4.5.2 TSimpleObjectBroker.ServerData Property

Encapsulates the list of servers as an OleVariant.

Pascal

```
property ServerData: OleVariant;
```

C++

```
__property OleVariant ServerData;
```

Description

Reading ServerData returns the value of the Servers (see page 5923) property stored in the internal component format. Writing ServerData sets the value of the Servers (see page 5923) property from an OleVariant that was saved previously.

Use the ServerData property to provide a mechanism for loading and saving the list of servers represented by the Servers (see page 5923) property. For example, An application can write the value of ServerData to an .ini file or the system registry. Later, it can read the value it saved as an OleVariant, and set ServerData to restore the Servers (see page 5923) property to the value that was saved.

Note: The implementation of ServerData uses the SaveToStream (see page 5921) method to encode the Servers (see page 5923) property as an OleVariant and the LoadFromStream (see page 5920) method to decode that OleVariant and restore the Servers (see page 5923) property.

See Also

Servers (see page 5923)

LoadFromStream (see page 5920)

SaveToStream (see page 5921)

59.1.4.5.3 TSimpleObjectBroker.Servers Property

Maintains the list of servers that can be provided to connection components.

Pascal

```
property Servers: TServerCollection;
```

C++

```
__property TServerCollection Servers;
```

Description

Use Servers to specify the list of servers that the simple object broker can return from the GetComputerForProgID (see page 5919) and GetComputerForGUID (see page 5919) methods.

Each item in the collection keeps track of whether the last connection attempt to that server was successful, and if not, the simple object broker ignores that server. In addition, individual servers in the collection can be disabled so that the simple object broker ignores them until they are re-enabled.

See Also

ServerData (see page 5922)

GetComputerForProgID (see page 5919)

GetComputerForGUID (see page 5919)

TServerItem (see page 5912)

60 ObjComAuto Namespace

This is namespace ObjComAuto.

Topics

Name	Description
Classes (see page 5926)	The following table lists classes in this documentation.
Functions (see page 5931)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5933)	The following table lists structs, records, enums in this documentation.
Types (see page 5935)	The following table lists types in this documentation.

Classes

	Name	Description
	TObjectDispatch (see page 5926)	This is class ObjComAuto.TObjectDispatch.

Functions

	Name	Description
	CreateMethodPointer (see page 5931)	This is function ObjComAuto.CreateMethodPointer.
	CreateMethodPointer (see page 5931)	This is function ObjComAuto.CreateMethodPointer.
	ReleaseMethodPointer (see page 5932)	This is function ObjComAuto.ReleaseMethodPointer.

Structs, Records, Enums

	Name	Description
	TDispatchInfo (see page 5933)	This is record ObjComAuto.TDispatchInfo.
	TDispatchKind (see page 5933)	This is record ObjComAuto.TDispatchKind.

Types

Name	Description
TDispatchInfos (see page 5935)	This is type ObjComAuto.TDispatchInfos.

60.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TObjectDispatch (see page 5926)	This is class ObjComAuto.TObjectDispatch.

60.1.1 TObjectDispatch Class

Class Hierarchy

Pascal

```
TObjectDispatch = class(TInterfacedObject, IDispatch);
```

C++

```
class TObjectDispatch : public TInterfacedObject, public IDispatch;
```

File

ObjComAuto

Description

This is class ObjComAuto.TObjectDispatch.

60.1.1.1 TObjectDispatch Members

The following tables list the members exposed by TObjectDispatch.

Protected Methods

	Name	Description
V	GetMethodInfo (see page 5928)	This is GetMethodInfo, a member of class TObjectDispatch.
V	GetObjectDispatch (see page 5928)	This is GetObjectDispatch, a member of class TObjectDispatch.
V	GetPropInfo (see page 5929)	This is GetPropInfo, a member of class TObjectDispatch.

Protected Properties

	Name	Description
R	Instance (see page 5930)	This is Instance, a member of class TObjectDispatch.

Public Methods

	Name	Description
	Create (see page 5927)	This is Create, a member of class TObjectDispatch.
V	Destroy (see page 5927)	Disposes of an object instance.
V	GetIDsOfNames (see page 5928)	This is GetIDsOfNames, a member of class TObjectDispatch.
	GetTypeInfo (see page 5929)	This is GetTypeInfo, a member of class TObjectDispatch.

	GetTypeInfoCount (see page 5929)	This is GetTypeInfoCount, a member of class TObjectDispatch.
	Invoke (see page 5929)	This is Invoke, a member of class TObjectDispatch.

60.1.1.2 TObjectDispatch.Create Constructor

Pascal

```
constructor Create(Instance: TObject; Owned: Boolean = True);
```

C++

```
__fastcall TObjectDispatch(TObject * Instance, Boolean Owned = True);
```

Description

This is Create, a member of class TObjectDispatch.

60.1.1.3 TObjectDispatch.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TObjectDispatch();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

60.1.1.4 TObjectDispatch Methods

The methods of the TObjectDispatch class are listed here.

Protected Methods

	Name	Description
	GetMethodInfo (see page 5928)	This is GetMethodInfo, a member of class TObjectDispatch.
	GetObjectDispatch (see page 5928)	This is GetObjectDispatch, a member of class TObjectDispatch.
	GetPropInfo (see page 5929)	This is GetPropInfo, a member of class TObjectDispatch.

Public Methods

	Name	Description
	GetIDsOfNames (see page 5928)	This is GetIDsOfNames, a member of class TObjectDispatch.
	GetTypeInfo (see page 5929)	This is GetTypeInfo, a member of class TObjectDispatch.
	GetTypeInfoCount (see page 5929)	This is GetTypeInfoCount, a member of class TObjectDispatch.
	Invoke (see page 5929)	This is Invoke, a member of class TObjectDispatch.

60.1.1.4.1 TObjectDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HRESULT; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int
NameCount, int LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TObjectDispatch.

60.1.1.4.2 TObjectDispatch.GetMethodInfo Method

Pascal

```
function GetMethodInfo(const AName: ShortString; var AInstance: TObject): PMethodInfoHeader;
virtual;
```

C++

```
virtual __fastcall PMethodInfoHeader GetMethodInfo(const ShortString AName, TObject *
AInstance);
```

Description

This is GetMethodInfo, a member of class TObjectDispatch.

60.1.1.4.3 TObjectDispatch.GetObjectDispatch Method

Pascal

```
function GetObjectDispatch(Obj: TObject): TObjectDispatch; virtual;
```

C++

```
virtual __fastcall TObjectDispatch GetObjectDispatch(TObject * Obj);
```

Description

This is GetObjectDispatch, a member of class TObjectDispatch.

60.1.1.4.4 TObjectDispatch.GetPropInfo Method

Pascal

```
function GetPropInfo(const AName: string; var AInstance: TObject; var CompIndex: Integer):
PPropInfo; virtual;
```

C++

```
virtual __fastcall PPropInfo GetPropInfo(const UnicodeString AName, TObject * AInstance, int
CompIndex);
```

Description

This is GetPropInfo, a member of class TObjectDispatch.

60.1.1.4.5 TObjectDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TObjectDispatch.

60.1.1.4.6 TObjectDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TObjectDispatch.

60.1.1.4.7 TObjectDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HRESULT; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word
Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TObjectDispatch.

60.1.1.5 TObjectDispatch Properties

The properties of the TObjectDispatch class are listed here.

Protected Properties

	Name	Description
	Instance (see page 5930)	This is Instance, a member of class TObjectDispatch.

60.1.1.5.1 TObjectDispatch.Instance Property

Pascal

```
property Instance: TObject;
```

C++

```
__property TObject * Instance;
```

Description

This is Instance, a member of class TObjectDispatch.

60.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateMethodPointer (see page 5931)	This is function ObjComAuto.CreateMethodPointer.
	CreateMethodPointer (see page 5931)	This is function ObjComAuto.CreateMethodPointer.
	ReleaseMethodPointer (see page 5932)	This is function ObjComAuto.ReleaseMethodPointer.

60.2.1 ObjComAuto.CreateMethodPointer Function

Pascal

```
function CreateMethodPointer(const Instance: IDispatch; DispID: Integer; TypeInfo: PTypeInfo):
TMethod; overload;
```

C++

```
TMethod CreateMethodPointer(const IDispatch Instance, int DispID, PTypeInfo TypeInfo);
```

File

ObjComAuto

Description

This is function ObjComAuto.CreateMethodPointer.

60.2.2 ObjComAuto.CreateMethodPointer Function

Pascal

```
function CreateMethodPointer(const Instance: IDispatch; const MethodName: string; TypeInfo:
PTypeInfo): TMethod; overload;
```

C++

```
TMethod CreateMethodPointer(const IDispatch Instance, const UnicodeString MethodName,
PTypeInfo TypeInfo);
```

File

ObjComAuto

Description

This is function ObjComAuto.CreateMethodPointer.

60.2.3 ObjComAuto.ReleaseMethodPointer Function

Pascal

```
procedure ReleaseMethodPointer(MethodPointer: TMethod);
```

C++

```
ReleaseMethodPointer(TMethod MethodPointer);
```

File

ObjComAuto

Description

This is function ObjComAuto.ReleaseMethodPointer.

60.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDispatchKind (see page 5933)	This is record ObjComAuto.TDispatchKind.

Records

	Name	Description
	TDispatchInfo (see page 5933)	This is record ObjComAuto.TDispatchInfo.

60.3.1 ObjComAuto.TDispatchInfo Record

Pascal

```
TDispatchInfo = record
  Instance: TObject;
  Kind: TDispatchKind;
  case TDispatchKind of
    dkMethod: (MethodInfo: PMethodInfoHeader;);
    dkProperty: (PropInfo: PPropInfo;);
    dkSubComponent: (Index: Integer;);
end;
```

C++

```
struct TDispatchInfo {
  TObject * Instance;
  TDispatchKind Kind;
  PMethodInfoHeader MethodInfo;
  PPropInfo PropInfo;
  int Index;
};
```

File

ObjComAuto

Description

This is record ObjComAuto.TDispatchInfo.

60.3.2 ObjComAuto.TDispatchKind Enumeration

Pascal

```
TDispatchKind = (
  dkMethod,
  dkProperty,
  dkSubComponent
);
```

C++

```
enum TDispatchKind {
  dkMethod,
  dkProperty,
  dkSubComponent
};
```

File

ObjComAuto

Description

This is record ObjComAuto.TDispatchKind.

60.4 Types

The following table lists types in this documentation.

Types

Name	Description
TDispatchInfos (see page 5935)	This is type ObjComAuto.TDispatchInfos.

60.4.1 ObjComAuto.TDispatchInfos Type

Pascal

```
TDispatchInfos = array of TDispatchInfo;
```

C++

```
array of TDispatchInfo TDispatchInfos;
```

File

ObjComAuto

Description

This is type ObjComAuto.TDispatchInfos.

61 OleAuto Namespace

This is namespace OleAuto.

Topics

Name	Description
Classes (see page 5939)	The following table lists classes in this documentation.
Functions (see page 5955)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5961)	The following table lists structs, records, enums in this documentation.
Types (see page 5963)	The following table lists types in this documentation.
Variables (see page 5964)	The following table lists variables in this documentation.
Constants (see page 5965)	The following table lists constants in this documentation.

Classes

	Name	Description
	EOleError (see page 5939)	This is class OleAuto.EOleError.
	EOleException (see page 5939)	This is class OleAuto.EOleException.
	EOleSysError (see page 5941)	This is class OleAuto.EOleSysError.
	TAutoDispatch (see page 5942)	This is class OleAuto.TAutoDispatch.
	TAutomation (see page 5946)	This is class OleAuto.TAutomation.
	TAutoObject (see page 5949)	This is class OleAuto.TAutoObject.
	TRegistryClass (see page 5953)	This is class OleAuto.TRegistryClass.

Constants

Name	Description
MaxDispArgs (see page 5965)	This is constant OleAuto.MaxDispArgs.

Functions

	Name	Description
	ClassIDToProgID (see page 5955)	This is function OleAuto.ClassIDToProgID.
	ClassIDToString (see page 5955)	This is function OleAuto.ClassIDToString.
	CreateOleObject (see page 5956)	This is function OleAuto.CreateOleObject.
	DispInvoke (see page 5956)	This is function OleAuto.DispInvoke.
	DispInvokeError (see page 5956)	This is function OleAuto.DispInvokeError.
	DllCanUnloadNow (see page 5957)	This is function OleAuto.DllCanUnloadNow.
	DllGetClassObject (see page 5957)	This is function OleAuto.DllGetClassObject.
	DllRegisterServer (see page 5957)	This is function OleAuto.DllRegisterServer.

	DllUnregisterServer (see page 5958)	This is function OleAuto.DllUnregisterServer.
	GetActiveOleObject (see page 5958)	This is function OleAuto.GetActiveOleObject.
	OleCheck (see page 5958)	This is function OleAuto.OleCheck.
	OleError (see page 5959)	This is function OleAuto.OleError.
	ProgIDToClassID (see page 5959)	This is function OleAuto.ProgIDToClassID.
	StringToClassID (see page 5959)	This is function OleAuto.StringToClassID.
	VarFromInterface (see page 5960)	This is function OleAuto.VarFromInterface.
	VarToAutoObject (see page 5960)	This is function OleAuto.VarToAutoObject.
	VarToInterface (see page 5960)	This is function OleAuto.VarToInterface.

Structs, Records, Enums

	Name	Description
	TAutoClassInfo (see page 5961)	This is record OleAuto.TAutoClassInfo.
	TAutoClassInstancing (see page 5961)	This is record OleAuto.TAutoClassInstancing.
	TStartMode (see page 5962)	This is record OleAuto.TStartMode.

Types

Name	Description
TAutoClass (see page 5963)	This is type OleAuto.TAutoClass.
TLastReleaseEvent (see page 5963)	This is type OleAuto.TLastReleaseEvent.

Variables

Name	Description
Automation (see page 5964)	This is variable OleAuto.Automation.

61.1 Classes

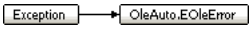
The following table lists classes in this documentation.

Classes

	Name	Description
	EOleError (see page 5939)	This is class OleAuto.EOleError.
	EOleException (see page 5939)	This is class OleAuto.EOleException.
	EOleSysError (see page 5941)	This is class OleAuto.EOleSysError.
	TAutoDispatch (see page 5942)	This is class OleAuto.TAutoDispatch.
	TAutomation (see page 5946)	This is class OleAuto.TAutomation.
	TAutoObject (see page 5949)	This is class OleAuto.TAutoObject.
	TRegistryClass (see page 5953)	This is class OleAuto.TRegistryClass.

61.1.1 EOleError Class

Class Hierarchy



Pascal

```
EOleError = class(Exception);
```

C++

```
class EOleError : public Exception;
```

File

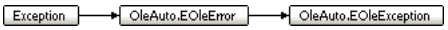
OleAuto

Description

This is class OleAuto.EOleError.

61.1.2 EOleException Class

Class Hierarchy



Pascal

```
EOleException = class(EOleError);
```

C++

```
class EOleException : public EOleError;
```

File

OleAuto

Description

This is class OleAuto.EOleException.

61.1.2.1 EOleException Members

The following tables list the members exposed by EOleException.

Public Methods

	Name	Description
	Create (see page 5940)	This is Create, a member of class EOleException.

Public Properties

	Name	Description
R	ErrorCode (see page 5940)	This is ErrorCode, a member of class EOleException.
R	HelpFile (see page 5941)	This is HelpFile, a member of class EOleException.
R	Source (see page 5941)	This is Source, a member of class EOleException.

61.1.2.2 EOleException.Create Constructor

Pascal

```
constructor Create(const ExcepInfo: TExcepInfo);
```

C++

```
__fastcall EOleException(const TExcepInfo ExcepInfo);
```

Description

This is Create, a member of class EOleException.

61.1.2.3 EOleException Properties

The properties of the EOleException class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 5940)	This is ErrorCode, a member of class EOleException.
R	HelpFile (see page 5941)	This is HelpFile, a member of class EOleException.
R	Source (see page 5941)	This is Source, a member of class EOleException.

61.1.2.3.1 EOleException.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class EOleException.

61.1.2.3.2 EOleException.HelpFile Property

Pascal

```
property HelpFile: string;
```

C++

```
__property UnicodeString HelpFile;
```

Description

This is HelpFile, a member of class EOleException.

61.1.2.3.3 EOleException.Source Property

Pascal

```
property Source: string;
```

C++

```
__property UnicodeString Source;
```

Description

This is Source, a member of class EOleException.

61.1.3 EOleSysError Class

Class Hierarchy

Pascal

```
EOleSysError = class(EOleError);
```

C++

```
class EOleSysError : public EOleError;
```

File

OleAuto

Description

This is class OleAuto.EOleSysError.

61.1.3.1 EOleSysError Members

The following tables list the members exposed by EOleSysError.

Public Methods

	Name	Description
	Create (see page 5942)	This is Create, a member of class EOleSysError.

Public Properties

	Name	Description
	ErrorCode (see page 5942)	This is ErrorCode, a member of class EOleSysError.

61.1.3.2 EOleSysError.Create Constructor

Pascal

```
constructor Create(ErrorCode: Integer);
```

C++

```
__fastcall EOleSysError(int ErrorCode);
```

Description

This is Create, a member of class EOleSysError.

61.1.3.3 EOleSysError Properties

The properties of the EOleSysError class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 5942)	This is ErrorCode, a member of class EOleSysError.

61.1.3.3.1 EOleSysError.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class EOleSysError.

61.1.4 TAutoDispatch Class

Class Hierarchy

Pascal

```
TAutoDispatch = class(IDispatch);
```

C++

```
class TAutoDispatch : public IDispatch;
```

File

OleAuto

Description

This is class OleAuto.TAutoDispatch.

61.1.4.1 TAutoDispatch Members

The following tables list the members exposed by TAutoDispatch.

Public Methods

	Name	Description
	AddRef (see page 5944)	This is AddRef, a member of class TAutoDispatch.
	Create (see page 5943)	This is Create, a member of class TAutoDispatch.
	GetAutoObject (see page 5944)	This is GetAutoObject, a member of class TAutoDispatch.
	GetIDsOfNames (see page 5944)	This is GetIDsOfNames, a member of class TAutoDispatch.
	GetTypeInfo (see page 5944)	This is GetTypeInfo, a member of class TAutoDispatch.
	GetTypeInfoCount (see page 5944)	This is GetTypeInfoCount, a member of class TAutoDispatch.
	Invoke (see page 5945)	This is Invoke, a member of class TAutoDispatch.
	QueryInterface (see page 5945)	This is QueryInterface, a member of class TAutoDispatch.
	Release (see page 5945)	This is Release, a member of class TAutoDispatch.

Public Properties

	Name	Description
	AutoObject (see page 5946)	This is AutoObject, a member of class TAutoDispatch.

61.1.4.2 TAutoDispatch.Create Constructor

Pascal

```
constructor Create(AutoObject: TAutoObject);
```

C++

```
__fastcall TAutoDispatch(TAutoObject AutoObject);
```

Description

This is Create, a member of class TAutoDispatch.

61.1.4.3 TAutoDispatch Methods

The methods of the TAutoDispatch class are listed here.

Public Methods

	Name	Description
	AddRef (see page 5944)	This is AddRef, a member of class TAutoDispatch.
	GetAutoObject (see page 5944)	This is GetAutoObject, a member of class TAutoDispatch.
	GetIDsOfNames (see page 5944)	This is GetIDsOfNames, a member of class TAutoDispatch.
	GetTypeInfo (see page 5944)	This is GetTypeInfo, a member of class TAutoDispatch.
	GetTypeInfoCount (see page 5944)	This is GetTypeInfoCount, a member of class TAutoDispatch.
	Invoke (see page 5945)	This is Invoke, a member of class TAutoDispatch.
	QueryInterface (see page 5945)	This is QueryInterface, a member of class TAutoDispatch.
	Release (see page 5945)	This is Release, a member of class TAutoDispatch.

61.1.4.3.1 TAutoDispatch.AddRef Method

Pascal

```
function AddRef: Longint; override;
```

C++

```
virtual __fastcall Longint AddRef();
```

Description

This is AddRef, a member of class TAutoDispatch.

61.1.4.3.2 TAutoDispatch.GetAutoObject Method

Pascal

```
function GetAutoObject: TAutoObject; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall TAutoObject GetAutoObject();
```

Description

This is GetAutoObject, a member of class TAutoDispatch.

61.1.4.3.3 TAutoDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const iid: TIID; rgszNames: POleStrList; cNames: Integer; lcid: TLCID;
rgdispid: PDispIDList): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetIDsOfNames(const TIID iid, POleStrList rgszNames, int cNames,
TLCID lcid, PDispIDList rgdispid);
```

Description

This is GetIDsOfNames, a member of class TAutoDispatch.

61.1.4.3.4 TAutoDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(itinfo: Integer; lcid: TLCID; var tinfo: ITypeInfo): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetTypeInfo(int itinfo, TLCID lcid, ITypeInfo tinfo);
```

Description

This is GetTypeInfo, a member of class TAutoDispatch.

61.1.4.3.5 TAutoDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(var ctinfo: Integer): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetTypeInfoCount(int ctinfo);
```

Description

This is GetTypeInfoCount, a member of class TAutoDispatch.

61.1.4.3.6 TAutoDispatch.Invoke Method

Pascal

```
function Invoke(dispIDMember: TDispID; const iid: TIID; lcid: TLCID; flags: Word; var
dispParams: TDispParams; varResult: PVariant; excepInfo: PExcepInfo; argErr: PInteger):
HResult; override;
```

C++

```
virtual __fastcall HRESULT Invoke(TDispID dispIDMember, const TIID iid, TLCID lcid, Word
flags, TDispParams dispParams, PVariant varResult, PExcepInfo excepInfo, PInteger argErr);
```

Description

This is Invoke, a member of class TAutoDispatch.

61.1.4.3.7 TAutoDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const iid: TIID; var obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TIID iid,  obj);
```

Description

This is QueryInterface, a member of class TAutoDispatch.

61.1.4.3.8 TAutoDispatch.Release Method

Pascal

```
function Release: Longint; override;
```

C++

```
virtual __fastcall Longint Release();
```

Description

This is Release, a member of class TAutoDispatch.

61.1.4.4 TAutoDispatch Properties

The properties of the TAutoDispatch class are listed here.

Public Properties

	Name	Description
	AutoObject (see page 5946)	This is AutoObject, a member of class TAutoDispatch.

61.1.4.4.1 TAutoDispatch.AutoObject Property

Pascal

```
property AutoObject: TAutoObject;
```

C++

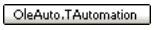
```
__property TAutoObject AutoObject;
```

Description

This is AutoObject, a member of class TAutoDispatch.

61.1.5 TAutomation Class

Class Hierarchy



Pascal

```
TAutomation = class;
```

C++

```
class TAutomation;
```

File

OleAuto

Description

This is class OleAuto.TAutomation.

61.1.5.1 TAutomation Members

The following tables list the members exposed by TAutomation.

Public Events

	Name	Description
	OnLastRelease (see page 5949)	This is OnLastRelease, a member of class TAutomation.

Public Methods

	Name	Description
	Create (see page 5947)	This is Create, a member of class TAutomation.
V	Destroy (see page 5947)	This is Destroy, a member of class TAutomation.
	RegisterClass (see page 5947)	This is RegisterClass, a member of class TAutomation.
	UpdateRegistry (see page 5947)	This is UpdateRegistry, a member of class TAutomation.

Public Properties

	Name	Description
R	AutoObjectCount (see page 5948)	This is AutoObjectCount, a member of class TAutomation.
	IsInprocServer (see page 5948)	This is IsInprocServer, a member of class TAutomation.
R	StartMode (see page 5948)	This is StartMode, a member of class TAutomation.

61.1.5.2 TAutomation.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TAutomation();
```

Description

This is Create, a member of class TAutomation.

61.1.5.3 TAutomation.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAutomation();
```

Description

This is Destroy, a member of class TAutomation.

61.1.5.4 TAutomation Methods

The methods of the TAutomation class are listed here.

Public Methods

	Name	Description
	RegisterClass (see page 5947)	This is RegisterClass, a member of class TAutomation.
	UpdateRegistry (see page 5947)	This is UpdateRegistry, a member of class TAutomation.

61.1.5.4.1 TAutomation.RegisterClass Method

Pascal

```
procedure RegisterClass(const AutoClassInfo: TAutoClassInfo);
```

C++

```
__fastcall RegisterClass(const TAutoClassInfo AutoClassInfo);
```

Description

This is RegisterClass, a member of class TAutomation.

61.1.5.4.2 TAutomation.UpdateRegistry Method

Pascal

```
procedure UpdateRegistry(Register: Boolean);
```

C++

```
__fastcall UpdateRegistry(Boolean Register);
```

Description

This is UpdateRegistry, a member of class TAutomation.

61.1.5.5 TAutomation Properties

The properties of the TAutomation class are listed here.

Public Properties

	Name	Description
	AutoObjectCount (see page 5948)	This is AutoObjectCount, a member of class TAutomation.
	IsInprocServer (see page 5948)	This is IsInprocServer, a member of class TAutomation.
	StartMode (see page 5948)	This is StartMode, a member of class TAutomation.

61.1.5.5.1 TAutomation.AutoObjectCount Property

Pascal

```
property AutoObjectCount: Integer;
```

C++

```
__property int AutoObjectCount;
```

Description

This is AutoObjectCount, a member of class TAutomation.

61.1.5.5.2 TAutomation.IsInprocServer Property

Pascal

```
property IsInprocServer: Boolean;
```

C++

```
__property Boolean IsInprocServer;
```

Description

This is IsInprocServer, a member of class TAutomation.

61.1.5.5.3 TAutomation.StartMode Property

Pascal

```
property StartMode: TStartMode;
```

C++

```
__property TStartMode StartMode;
```

Description

This is StartMode, a member of class TAutomation.

61.1.5.6 TAutomation Events

The events of the TAutomation class are listed here.

Public Events

	Name	Description
	OnLastRelease (see page 5949)	This is OnLastRelease, a member of class TAutomation.

61.1.5.6.1 TAutomation.OnLastRelease Event

Pascal

```
property OnLastRelease: TLastReleaseEvent;
```

C++

```
__property TLastReleaseEvent OnLastRelease;
```

Description

This is OnLastRelease, a member of class TAutomation.

61.1.6 TAutoObject Class

Class Hierarchy

Pascal

```
TAutoObject = class(TObject);
```

C++

```
class TAutoObject : public TObject;
```

File

OleAuto

Description

This is class OleAuto.TAutoObject.

61.1.6.1 TAutoObject Members

The following tables list the members exposed by TAutoObject.

Protected Methods

	Name	Description
	CreateAutoDispatch (see page 5951)	This is CreateAutoDispatch, a member of class TAutoObject.
	GetExceptionInfo (see page 5951)	This is GetExceptionInfo, a member of class TAutoObject.

Public Methods

	Name	Description
	AddRef (see page 5951)	This is AddRef, a member of class TAutoObject.
	Create (see page 5950)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 5950)	Disposes of an object instance.
	Release (see page 5952)	This is Release, a member of class TAutoObject.

Public Properties

	Name	Description
	AutoDispatch (see page 5952)	This is AutoDispatch, a member of class TAutoObject.
	OleObject (see page 5952)	This is OleObject, a member of class TAutoObject.
	RefCount (see page 5952)	This is RefCount, a member of class TAutoObject.

61.1.6.2 TAutoObject.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TAutoObject();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy

System::TObject::Free

61.1.6.3 TAutoObject.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAutoObject();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the

object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

61.1.6.4 TAutoObject Methods

The methods of the TAutoObject class are listed here.

Protected Methods

	Name	Description
	CreateAutoDispatch (see page 5951)	This is CreateAutoDispatch, a member of class TAutoObject.
	GetExceptionInfo (see page 5951)	This is GetExceptionInfo, a member of class TAutoObject.

Public Methods

	Name	Description
	AddRef (see page 5951)	This is AddRef, a member of class TAutoObject.
	Release (see page 5952)	This is Release, a member of class TAutoObject.

61.1.6.4.1 TAutoObject.AddRef Method

Pascal

```
function AddRef: Integer;
```

C++

```
__fastcall int AddRef();
```

Description

This is AddRef, a member of class TAutoObject.

61.1.6.4.2 TAutoObject.CreateAutoDispatch Method

Pascal

```
function CreateAutoDispatch: TAutoDispatch; virtual;
```

C++

```
virtual __fastcall TAutoDispatch CreateAutoDispatch();
```

Description

This is CreateAutoDispatch, a member of class TAutoObject.

61.1.6.4.3 TAutoObject.GetExceptionInfo Method

Pascal

```
procedure GetExceptionInfo(ExceptObject: TObject; var ExcepInfo: TExcepInfo); virtual;
```

C++

```
virtual __fastcall GetExceptionInfo(TObject * ExceptObject, TExcepInfo ExcepInfo);
```

Description

This is GetExceptionInfo, a member of class TAutoObject.

61.1.6.4.4 TAutoObject.Release Method

Pascal

```
function Release: Integer;
```

C++

```
__fastcall int Release();
```

Description

This is Release, a member of class TAutoObject.

61.1.6.5 TAutoObject Properties

The properties of the TAutoObject class are listed here.

Public Properties

	Name	Description
	AutoDispatch (see page 5952)	This is AutoDispatch, a member of class TAutoObject.
	OleObject (see page 5952)	This is OleObject, a member of class TAutoObject.
	RefCount (see page 5952)	This is RefCount, a member of class TAutoObject.

61.1.6.5.1 TAutoObject.AutoDispatch Property

Pascal

```
property AutoDispatch: TAutoDispatch;
```

C++

```
__property TAutoDispatch AutoDispatch;
```

Description

This is AutoDispatch, a member of class TAutoObject.

61.1.6.5.2 TAutoObject.OleObject Property

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

This is OleObject, a member of class TAutoObject.

61.1.6.5.3 TAutoObject.RefCount Property

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

This is RefCount, a member of class TAutoObject.

61.1.7 TRegistryClass Class

Class Hierarchy

Pascal

```
TRegistryClass = class;
```

C++

```
class TRegistryClass;
```

File

OleAuto

Description

This is class OleAuto.TRegistryClass.

61.1.7.1 TRegistryClass Members

The following tables list the members exposed by TRegistryClass.

Public Methods

	Name	Description
	Create (see page 5953)	This is Create, a member of class TRegistryClass.
	Destroy (see page 5953)	This is Destroy, a member of class TRegistryClass.
	UpdateRegistry (see page 5954)	This is UpdateRegistry, a member of class TRegistryClass.

61.1.7.2 TRegistryClass.Create Constructor

Pascal

```
constructor Create(const AutoClassInfo: TAutoClassInfo);
```

C++

```
__fastcall TRegistryClass(const TAutoClassInfo AutoClassInfo);
```

Description

This is Create, a member of class TRegistryClass.

61.1.7.3 TRegistryClass.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistryClass();
```

Description

This is Destroy, a member of class TRegistryClass.

61.1.7.4 TRegistryClass Methods

The methods of the TRegistryClass class are listed here.

Public Methods

	Name	Description
	UpdateRegistry (see page 5954)	This is UpdateRegistry, a member of class TRegistryClass.

61.1.7.4.1 TRegistryClass.UpdateRegistry Method

Pascal

```
procedure UpdateRegistry(Register: Boolean);
```

C++

```
__fastcall UpdateRegistry(Boolean Register);
```

Description

This is UpdateRegistry, a member of class TRegistryClass.

61.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ClassIDToProgID (see page 5955)	This is function OleAuto.ClassIDToProgID.
	ClassIDToString (see page 5955)	This is function OleAuto.ClassIDToString.
	CreateOleObject (see page 5956)	This is function OleAuto.CreateOleObject.
	DispInvoke (see page 5956)	This is function OleAuto.DispInvoke.
	DispInvokeError (see page 5956)	This is function OleAuto.DispInvokeError.
	DllCanUnloadNow (see page 5957)	This is function OleAuto.DllCanUnloadNow.
	DllGetClassObject (see page 5957)	This is function OleAuto.DllGetClassObject.
	DllRegisterServer (see page 5957)	This is function OleAuto.DllRegisterServer.
	DllUnregisterServer (see page 5958)	This is function OleAuto.DllUnregisterServer.
	GetActiveOleObject (see page 5958)	This is function OleAuto.GetActiveOleObject.
	OleCheck (see page 5958)	This is function OleAuto.OleCheck.
	OleError (see page 5959)	This is function OleAuto.OleError.
	ProgIDToClassID (see page 5959)	This is function OleAuto.ProgIDToClassID.
	StringToClassID (see page 5959)	This is function OleAuto.StringToClassID.
	VarFromInterface (see page 5960)	This is function OleAuto.VarFromInterface.
	VarToAutoObject (see page 5960)	This is function OleAuto.VarToAutoObject.
	VarToInterface (see page 5960)	This is function OleAuto.VarToInterface.

61.2.1 OleAuto.ClassIDToProgID Function

Pascal

```
function ClassIDToProgID(const ClassID: TCLSID): string;
```

C++

```
UnicodeString ClassIDToProgID(const TCLSID ClassID);
```

File

OleAuto

Description

This is function OleAuto.ClassIDToProgID.

61.2.2 OleAuto.ClassIDToString Function

Pascal

```
function ClassIDToString(const ClassID: TCLSID): string;
```

C++

```
UnicodeString ClassIDToString(const TCLSID ClassID);
```

File

OleAuto

Description

This is function OleAuto.ClassIDToString.

61.2.3 OleAuto.CreateOleObject Function

Pascal

```
function CreateOleObject(const ClassName: string): Variant;
```

C++

```
Variant CreateOleObject(const UnicodeString ClassName);
```

File

OleAuto

Description

This is function OleAuto.CreateOleObject.

61.2.4 OleAuto.DispInvoke Function

Pascal

```
procedure DispInvoke(Dispatch: IDispatch; CallDesc: PCallDesc; DispIDs: PDispIDList; Params:
Pointer; Result: PVariant);
```

C++

```
DispInvoke(IDispatch Dispatch, PCallDesc CallDesc, PDispIDList DispIDs, void * Params,
PVariant Result);
```

File

OleAuto

Description

This is function OleAuto.DispInvoke.

61.2.5 OleAuto.DispInvokeError Function

Pascal

```
procedure DispInvokeError(Status: HResult; const ExcepInfo: TExcepInfo);
```

C++

```
DispInvokeError(HRESULT Status, const TExcepInfo ExcepInfo);
```

File

OleAuto

Description

This is function OleAuto.DispInvokeError.

61.2.6 OleAuto.DllCanUnloadNow Function

Pascal

```
function DllCanUnloadNow: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllCanUnloadNow();
```

File

OleAuto

Description

This is function OleAuto.DllCanUnloadNow.

61.2.7 OleAuto.DllGetClassObject Function

Pascal

```
function DllGetClassObject(const CLSID: TCLSID; const IID: TIID; var Obj): HResult; stdcall;
```

C++

```
__stdcall HRESULT DllGetClassObject(const TCLSID CLSID, const TIID IID,  Obj);
```

File

OleAuto

Description

This is function OleAuto.DllGetClassObject.

61.2.8 OleAuto.DllRegisterServer Function

Pascal

```
function DllRegisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllRegisterServer();
```

File

OleAuto

Description

This is function OleAuto.DllRegisterServer.

61.2.9 OleAuto.DllUnregisterServer Function

Pascal

```
function DllUnregisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllUnregisterServer();
```

File

OleAuto

Description

This is function OleAuto.DllUnregisterServer.

61.2.10 OleAuto.GetActiveOleObject Function

Pascal

```
function GetActiveOleObject(const ClassName: string): Variant;
```

C++

```
Variant GetActiveOleObject(const UnicodeString ClassName);
```

File

OleAuto

Description

This is function OleAuto.GetActiveOleObject.

61.2.11 OleAuto.OleCheck Function

Pascal

```
procedure OleCheck(Result: HResult);
```

C++

```
OleCheck(HRESULT Result);
```

File

OleAuto

Description

This is function OleAuto.OleCheck.

61.2.12 OleAuto.OleError Function

Pascal

```
procedure OleError(ErrorCode: HResult);
```

C++

```
OleError(HRESULT ErrorCode);
```

File

OleAuto

Description

This is function OleAuto.OleError.

61.2.13 OleAuto.ProgIDToClassID Function

Pascal

```
function ProgIDToClassID(const ProgID: string): TCLSID;
```

C++

```
TCLSID ProgIDToClassID(const UnicodeString ProgID);
```

File

OleAuto

Description

This is function OleAuto.ProgIDToClassID.

61.2.14 OleAuto.StringToClassID Function

Pascal

```
function StringToClassID(const S: string): TCLSID;
```

C++

```
TCLSID StringToClassID(const UnicodeString S);
```

File

OleAuto

Description

This is function OleAuto.StringToClassID.

61.2.15 OleAuto.VarFromInterface Function

Pascal

```
function VarFromInterface(Unknown: IUnknown): Variant;
```

C++

```
Variant VarFromInterface(IUnknown Unknown);
```

File

OleAuto

Description

This is function OleAuto.VarFromInterface.

61.2.16 OleAuto.VarToAutoObject Function

Pascal

```
function VarToAutoObject(const V: Variant): TAutoObject;
```

C++

```
TAutoObject VarToAutoObject(const Variant V);
```

File

OleAuto

Description

This is function OleAuto.VarToAutoObject.

61.2.17 OleAuto.VarToInterface Function

Pascal

```
function VarToInterface(const V: Variant): IDispatch;
```

C++

```
IDispatch VarToInterface(const Variant V);
```

File

OleAuto

Description

This is function OleAuto.VarToInterface.

61.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoClassInstancing (see page 5961)	This is record OleAuto.TAutoClassInstancing.
	TStartMode (see page 5962)	This is record OleAuto.TStartMode.

Records

	Name	Description
	TAutoClassInfo (see page 5961)	This is record OleAuto.TAutoClassInfo.

61.3.1 OleAuto.TAutoClassInfo Record

Pascal

```
TAutoClassInfo = record
  AutoClass: TAutoClass;
  ProgID: string;
  ClassID: string;
  Description: string;
  Instancing: TAutoClassInstancing;
end;
```

C++

```
struct TAutoClassInfo {
  TAutoClass AutoClass;
  UnicodeString ProgID;
  UnicodeString ClassID;
  UnicodeString Description;
  TAutoClassInstancing Instancing;
};
```

File

OleAuto

Description

This is record OleAuto.TAutoClassInfo.

61.3.2 OleAuto.TAutoClassInstancing Enumeration

Pascal

```
TAutoClassInstancing = (
  acInternal,
  acSingleInstance,
  acMultiInstance
);
```

C++

```
enum TAutoClassInstancing {
  acInternal,
  acSingleInstance,
  acMultiInstance
};
```

File

OleAuto

Description

This is record OleAuto.TAutoClassInstancing.

61.3.3 OleAuto.TStartMode Enumeration

Pascal

```
TStartMode = (
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
);
```

C++

```
enum TStartMode {
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
};
```

File

OleAuto

Description

This is record OleAuto.TStartMode.

61.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAutoClass (see page 5963)	This is type OleAuto.TAutoClass.
TLastReleaseEvent (see page 5963)	This is type OleAuto.TLastReleaseEvent.

61.4.1 OleAuto.TAutoClass Type

Pascal

```
TAutoClass = class of TAutoObject;
```

C++

```
TAutoObject TAutoClass;
```

File

OleAuto

Description

This is type OleAuto.TAutoClass.

61.4.2 OleAuto.TLastReleaseEvent Type

Pascal

```
TLastReleaseEvent = procedure (var Shutdown: Boolean) of object;
```

C++

```
(var Shutdown: Boolean) ( TLastReleaseEvent)();
```

File

OleAuto

Description

This is type OleAuto.TLastReleaseEvent.

61.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Automation (see page 5964)	This is variable OleAuto.Automation.

61.5.1 OleAuto.Automation Variable

Pascal

```
Automation: TAutomation;
```

C++

```
TAutomation Automation;
```

File

OleAuto

Description

This is variable OleAuto.Automation.

61.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxDispArgs (see page 5965)	This is constant OleAuto.MaxDispArgs.

61.6.1 OleAuto.MaxDispArgs Constant

Pascal

```
MaxDispArgs = 32;
```

C++

```
MaxDispArgs = 32;
```

File

OleAuto

Description

This is constant OleAuto.MaxDispArgs.

62 OleCtnrs Namespace

This is namespace OleCtnrs.

Topics

Name	Description
Classes (see page 5969)	The following table lists classes in this documentation.
Interfaces (see page 6053)	The following table lists interfaces in this documentation.
Functions (see page 6055)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6056)	The following table lists structs, records, enums in this documentation.
Types (see page 6060)	The following table lists types in this documentation.
Constants (see page 6061)	The following table lists constants in this documentation.

Classes

	Name	Description
	TOleContainer (see page 5969)	TOleContainer embeds or links OLE objects.
	TOleForm (see page 6050)	TOleForm is an object that provides OLE support for a TForm (see page 2241) object.

Constants

Name	Description
ovDiscardUndoState (see page 6061)	This is constant OleCtnrs.ovDiscardUndoState.
ovHide (see page 6061)	This is constant OleCtnrs.ovHide.
ovInPlaceActivate (see page 6062)	This is constant OleCtnrs.ovInPlaceActivate.
ovOpen (see page 6062)	This is constant OleCtnrs.ovOpen.
ovPrimary (see page 6062)	This is constant OleCtnrs.ovPrimary.
ovShow (see page 6063)	This is constant OleCtnrs.ovShow.
ovUIActivate (see page 6063)	This is constant OleCtnrs.ovUIActivate.

Functions

	Name	Description
	DestroyMetaPict (see page 6055)	Frees a metafile resource given its handle.

Interfaces

	Name	Description
	IVCLFrameForm (see page 6053)	This is class OleCtnrs.IVCLFrameForm.

Structs, Records, Enums

	Name	Description
	TAutoActivate (see page 6056)	This is record OleCtnrs.TAutoActivate.
	TCreateInfo (see page 6056)	This is record OleCtnrs.TCreateInfo.
	TCreateType (see page 6057)	This is record OleCtnrs.TCreateType.
	TObjectState (see page 6057)	This is record OleCtnrs.TObjectState.
	TSizeMode (see page 6058)	This is record OleCtnrs.TSizeMode.
	TVerbInfo (see page 6058)	This is record OleCtnrs.TVerbInfo.

Types

Name	Description
TObjectMoveEvent (see page 6060)	This is type OleCtnrs.TObjectMoveEvent.

62.1 Classes

The following table lists classes in this documentation.

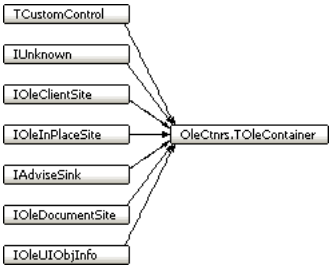
Classes

	Name	Description
	TOleContainer (see page 5969)	TOleContainer embeds or links OLE objects.
	TOleForm (see page 6050)	TOleForm is an object that provides OLE support for a TForm (see page 2241) object.

62.1.1 TOleContainer Class

TOleContainer embeds or links OLE objects.

Class Hierarchy



Pascal

```
TOleContainer = class(TCustomControl, IUnknown, IOleClientSite, IOleInPlaceSite, IAdviseSink,
IOleDocumentSite, IOleUIObjInfo);
```

C++

```
class TOleContainer : public TCustomControl, public IUnknown, public IOleClientSite, public
IOleInPlaceSite, public IAdviseSink, public IOleDocumentSite, public IOleUIObjInfo;
```

File

OleCtnrs

Description

Use TOleContainer to handle many of the complexities of OLE 2.0. TOleContainer lets the user choose an OLE object to insert by simply calling the InsertObjectDialog (see page 5982) method. TOleContainer can create either an embedded OLE object or a linked OLE object.

TOleContainer automatically handles menu merging -- the process of combining the container form's menu with that of an in-place activated OLE object's server application. The menu items' GroupIndex property settings control how menus are merged. Those main menu items with GroupIndex values of 0, 2, and 4 remain; TOleContainer merges the server application's menus and replaces the main menu items with GroupIndex values of 1, 3, and 5 (if they exist).

OLE objects that are activated in-place add their server's toolbars directly into the container application's window. Normally, any panels used for toolbars are replaced by the OLE object's server's toolbars. Prevent this by setting a panel's Locked property to true.

When using TOleContainer in an SDI application (the main form's FormStyle property is fsNormal rather than fsMDIForm), place the TOleContainer component inside a panel whose Align (see page 5991) property is set to alClient. TOleContainer replaces an SDI form's toolbars as described above, and using a panel automatically adjusts the amount of space available to the OLE container.

See Also

TForm (see page 2241)

TMainMenu (see page 4828)

TMenuItem (see page 4869)

TOleControl (see page 6074)

TPanel (see page 1326)

Creating Clients for Servers That Do Not Have a Type Library

Active Documents

Building COM Applications

62.1.1.1 TOleContainer Members

The following tables list the members exposed by TOleContainer.

Protected Methods

	Name	Description
	_AddRef (see page 5974)	Called when an application uses a component interface.
	_Release (see page 5975)	Called when an application releases a component interface.
	Changed (see page 5975)	Sends a CM_CHANGED message to the control.
	CreateParams (see page 5979)	Initializes a window-creation parameter data structure.
	DblClick (see page 5980)	OnDblClick event dispatcher.
	DefineProperties (see page 5980)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DoEnter (see page 5981)	Respond to receiving input focus.
	GetPopupMenu (see page 5982)	Returns the pop-up menu associated with the control.
	KeyDown (see page 5983)	Respond to key press events.
	MouseDown (see page 5984)	OnMouseDown event dispatcher.
	Paint (see page 5985)	Renders the image of a custom control.
	QueryInterface (see page 5987)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	ChangeIconDialog (see page 5976)	Displays the Change Icon OLE dialog box.
	Close (see page 5976)	Deactivates the OLE object.
	Copy (see page 5976)	Copies the OLE object to the Windows clipboard.
	Create (see page 5972)	Creates and initializes an instance of TOleContainer (see page 5969).
	CreateLinkToFile (see page 5977)	Creates a linked OLE object from the contents of a file.
	CreateObject (see page 5977)	Creates an embedded OLE object given its class name (also known as programmatic identifier).
	CreateObjectFromFile (see page 5978)	Creates an embedded OLE object from the contents of a file.
	CreateObjectFromInfo (see page 5979)	Creates an OLE object matching the specified TCreateInfo (see page 6056) record.
	Destroy (see page 5973)	Destroys an instance of TOleContainer (see page 5969).
	DestroyObject (see page 5981)	Destroys the container's OLE object.
	DoVerb (see page 5981)	Requests that the OLE object perform some action.

	GetIconMetaPict (see page 5982)	Returns the global memory handle of a metafile containing the OLE object's icon.
	InsertObjectDialog (see page 5982)	Displays the Insert Object OLE dialog box.
	LoadFromFile (see page 5984)	Loads an OLE object from the specified file.
	LoadFromStream (see page 5984)	Loads an OLE object from the specified stream.
	ObjectPropertiesDialog (see page 5985)	Displays the OLE Object Properties property sheet.
	Paste (see page 5986)	Pastes the contents of the Windows clipboard as an embedded object.
	PasteSpecialDialog (see page 5986)	Displays the Paste (see page 5986) Special OLE dialog box.
	Run (see page 5987)	Runs the OLE object's server application, but doesn't open or activate the OLE object itself.
	SaveAsDocument (see page 5987)	Saves the OLE object to the specified file in native OLE Document format.
	SaveToFile (see page 5989)	Saves the OLE object to the specified file.
	SaveToStream (see page 5989)	Saves the OLE object to the specified stream.
	UpdateObject (see page 5990)	Refreshes the OLE object.
	UpdateVerbs (see page 5990)	Refreshes the list of verbs the OLE object can respond to.

Public Properties

	Name	Description
R	CanPaste (see page 5997)	Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.
R	Linked (see page 5005)	Indicates whether the OLE object is linked or embedded.
	Modified (see page 6006)	Indicates whether the OLE object has been modified.
R	NewInserted (see page 6006)	Indicates whether the OLE object was created rather than pasted in or created as a link to a file.
R	ObjectVerbs (see page 6006)	Lists the verbs that the OLE object supports.
R	OleClassName (see page 6007)	Specifies the class name of the OLE object.
R	OleObject (see page 6008)	Represents the OLE object hosted by the OLE container.
R	OleObjectInterface (see page 6008)	Provides access to the IOleObject interface for the OLE object.
R	PrimaryVerb (see page 6015)	Specifies the index of the primary (default) verb for the OLE object.
R	SourceDoc (see page 6017)	Specifies the name of the source document for a linked OLE object.
R	State (see page 6017)	Describes the state of the OLE object.
R	StorageInterface (see page 6018)	Provides access to the IStorage interface for the OLE object.

Published Events

	Name	Description
	OnActivate (see page 6021)	Occurs immediately after the OLE object is activated.
	OnDeactivate (see page 6022)	Occurs immediately after the OLE object is deactivated.
	OnDragDrop (see page 6022)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 6025)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 6028)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6029)	Occurs when a control receives the input focus.
	OnExit (see page 6032)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 6035)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6038)	Occurs when a key is pressed.
	OnKeyUp (see page 6039)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 6041)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 6042)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 6044)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 6047)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnObjectMove (see page 6049)	Occurs when the OLE container is notified that the OLE object has moved or resized.
	OnResize (see page 6049)	Occurs immediately after the OLE container window is resized.

	OnStartDrag (see page 6049)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 5991)	Determines how the control aligns within its container (parent control).
	AllowActiveDoc (see page 5995)	Specifies whether the OLE container supports the IOleDocumentSite interface.
	AllowInPlace (see page 5995)	Determines whether the OLE container allows in-place activation.
	Anchors (see page 5996)	Specifies how the control is anchored to its parent.
	AutoActivate (see page 5996)	Determines how the object in an OLE container is activated.
	AutoVerbMenu (see page 5997)	Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.
	BorderStyle (see page 5997)	Determines whether a single line is drawn around the OLE container.
	Caption (see page 5998)	Specifies a text string that identifies the control to the user.
	Color (see page 6000)	Specifies the background color of the control.
	Constraints (see page 6001)	Specifies the size constraints for the control.
	CopyOnSave (see page 6002)	Determines whether the SaveToFile (see page 5989) and SaveToStream (see page 5989) methods first create a copy of the OLE object.
	Ctl3D (see page 6002)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 6003)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 6003)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 6005)	Controls whether the control responds to mouse, keyboard, and timer events.
	Iconic (see page 6005)	Determines whether the OLE object is displayed as an icon.
	OldStreamFormat (see page 6007)	Indicates whether the OLE object is loaded and saved using the VCL format.
	ParentColor (see page 6008)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6009)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 6009)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 6010)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 6015)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SizeMode (see page 6017)	Determines how the OLE object is sized in the OLE container.
	TabOrder (see page 6018)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6019)	Determines if the user can tab to a control.
	Visible (see page 6019)	Determines whether the component appears onscreen.

62.1.1.2 TOleContainer.Create Constructor

Creates and initializes an instance of TOleContainer (see page 5969).

Pascal

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

C++

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

Description

Call Create to instantiate a TOleContainer (see page 5969) object at runtime. OLE containers placed on forms at design time are created automatically.

Pass a component to the constructor to supply the OLE container with an Owner that is responsible for freeing it.

After calling the inherited constructor, Create

Sets ControlStyle to [csClickEvents, csSetCaption, csOpaque, csDoubleClicks].

Sets TabStop (see page 6019) to true.

Sets ParentColor (see page 6008) to false and BorderStyle (see page 5997) to bsSingle.

Sets AllowActiveDoc (see page 5995) and AllowInPlace (see page 5995) to true.

Sets AutoActivate (see page 5996) to aaDoubleClick and AutoVerbMenu (see page 5997) to true.

Sets CopyOnSave (see page 6002) to true.

See Also

AllowActiveDoc (see page 5995)

AllowInPlace (see page 5995)

AutoActivate (see page 5996)

AutoVerbMenu (see page 5997)

BorderStyle (see page 5997)

ControlStyle

CopyOnSave (see page 6002)

ParentColor

TabStop

62.1.1.3 TOleContainer.Destroy Destructor

Destroys an instance of TOleContainer (see page 5969).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleContainer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TOleContainer (see page 5969) reference is not nil, and only then calls Destroy.Destroy frees the OLE object before calling the inherited destructor.

See Also

DestroyObject (see page 5981)

System::TObject::Free

62.1.1.4 TOleContainer Methods

The methods of the TOleContainer class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5974)	Called when an application uses a component interface.
	_Release (see page 5975)	Called when an application releases a component interface.
	Changed (see page 5975)	Sends a CM_CHANGED message to the control.
V	CreateParams (see page 5979)	Initializes a window-creation parameter data structure.

	DblClick (see page 5980)	OnDblClick event dispatcher.
	DefineProperties (see page 5980)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DoEnter (see page 5981)	Respond to receiving input focus.
	GetPopupMenu (see page 5982)	Returns the pop-up menu associated with the control.
	KeyDown (see page 5983)	Respond to key press events.
	MouseDown (see page 5984)	OnMouseDown event dispatcher.
	Paint (see page 5985)	Renders the image of a custom control.
	QueryInterface (see page 5987)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	ChangeIconDialog (see page 5976)	Displays the Change Icon OLE dialog box.
	Close (see page 5976)	Deactivates the OLE object.
	Copy (see page 5976)	Copies the OLE object to the Windows clipboard.
	CreateLinkToFile (see page 5977)	Creates a linked OLE object from the contents of a file.
	CreateObject (see page 5977)	Creates an embedded OLE object given its class name (also known as programmatic identifier).
	CreateObjectFromFile (see page 5978)	Creates an embedded OLE object from the contents of a file.
	CreateObjectFromInfo (see page 5979)	Creates an OLE object matching the specified TCreateInfo (see page 6056) record.
	DestroyObject (see page 5981)	Destroys the container's OLE object.
	DoVerb (see page 5981)	Requests that the OLE object perform some action.
	GetIconMetaPict (see page 5982)	Returns the global memory handle of a metafile containing the OLE object's icon.
	InsertObjectDialog (see page 5982)	Displays the Insert Object OLE dialog box.
	LoadFromFile (see page 5984)	Loads an OLE object from the specified file.
	LoadFromStream (see page 5984)	Loads an OLE object from the specified stream.
	ObjectPropertiesDialog (see page 5985)	Displays the OLE Object Properties property sheet.
	Paste (see page 5986)	Pastes the contents of the Windows clipboard as an embedded object.
	PasteSpecialDialog (see page 5986)	Displays the Paste (see page 5986) Special OLE dialog box.
	Run (see page 5987)	Runs the OLE object's server application, but doesn't open or activate the OLE object itself.
	SaveAsDocument (see page 5987)	Saves the OLE object to the specified file in native OLE Document format.
	SaveToFile (see page 5989)	Saves the OLE object to the specified file.
	SaveToStream (see page 5989)	Saves the OLE object to the specified stream.
	UpdateObject (see page 5990)	Refreshes the OLE object.
	UpdateVerbs (see page 5990)	Refreshes the list of verbs the OLE object can respond to.

62.1.1.4.1 TOleContainer._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke

QueryInterface

_Release

ComObject

_AddRef

62.1.1.4.2 TOleContainer._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke

QueryInterface

_AddRef

ComObject

62.1.1.4.3 TOleContainer.Changed Method

Sends a CM_CHANGED message to the control.

Pascal

```
procedure Changed; dynamic;
```

C++

```
__fastcall Changed();
```

Description

Call Changed after changing the properties of the control in a way that affects the properties of a parent object. Changed sends a CM_CHANGED message back to the control. If the control is a windowed control, this message is bubbled up to the parent so that it can make any necessary adjustments.

See Also

Parent

62.1.1.4.4 TOleContainer.ChangeIconDialog Method

Displays the Change Icon OLE dialog box.

Pascal

```
function ChangeIconDialog: Boolean;
```

C++

```
__fastcall Boolean ChangeIconDialog();
```

Description

Call ChangeIconDialog to let the user choose a different icon (from .ICO, .DLL, and .EXE files) to use when displaying the OLE object as an icon. An OLE object must already be loaded before calling ChangeIconDialog. ChangeIconDialog returns true if the dialog box was successfully displayed and the user chose the OK button. ChangeIconDialog returns false otherwise.

See Also

Iconic (see page 6005)

ObjectPropertiesDialog (see page 5985)

62.1.1.4.5 TOleContainer.Close Method

Deactivates the OLE object.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to deactivate the OLE object and terminate its server application, without removing it from the container. Any changes the user made to the OLE object are automatically saved. After calling Close, the container's State (see page 6017) property is osLoaded.

Note: If Close is called when no OLE object is loaded, TOleContainer (see page 5969) raises an exception.

See Also

OnDeactivate (see page 6022)

State (see page 6017)

62.1.1.4.6 TOleContainer.Copy Method

Copies the OLE object to the Windows clipboard.

Pascal

```
procedure Copy;
```

C++

```
__fastcall Copy();
```

Description

Call Copy to write the OLE object to the clipboard. If there is no OLE object in the container, Copy raises an exception.

See Also

Paste (see page 5986)

PasteSpecialDialog (see page 5986)

SaveToFile (see page 5989)

SaveToStream (see page 5989)

62.1.1.4.7 TOleContainer.CreateLinkToFile Method

Creates a linked OLE object from the contents of a file.

Pascal

```
procedure CreateLinkToFile(const FileName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateLinkToFile(const UnicodeString FileName, Boolean Iconic);
```

Description

Call CreateLinkToFile to create a linked OLE object from the file specified by the FileName parameter. Iconic (see page 6005) specifies whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there is already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateObject (see page 5977)

CreateObjectFromFile (see page 5978)

CreateObjectFromInfo (see page 5979)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.8 TOleContainer.CreateObject Method

Creates an embedded OLE object given its class name (also known as programmatic identifier).

Pascal

```
procedure CreateObject(const OleClassName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateObject(const UnicodeString OleClassName, Boolean Iconic);
```

Description

Call CreateObject to create an OLE object using the class name (also known as programmatic identifier) specified by the OleClassName (see page 6007) parameter. The Iconic (see page 6005) parameter indicates whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateLinkToFile (see page 5977)

CreateObjectFromFile (see page 5978)

CreateObjectFromInfo (see page 5979)

OleClassName (see page 6007)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.9 TOleContainer.CreateObjectFromFile Method

Creates an embedded OLE object from the contents of a file.

Pascal

```
procedure CreateObjectFromFile(const FileName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateObjectFromFile(const UnicodeString FileName, Boolean Iconic);
```

Description

Call CreateObjectFromFile to create an OLE object from the file specified by the FileName parameter. The Iconic (see page 6005) parameter indicates whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

Delphi Examples:

```
{
This example uses an OLE container and a main menu on a form.
When the user clicks the Open or Save options in the main menu,
an open/save dialog is displayed and by default,
the user is required to enter the name of a Word document.
Files with other extensions supported by the OLE container
can also be accessed. An additional test is made to assure that
the file to open exists and that no file overwriting is
going to take place.
}
procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenDialog1.Filter := 'Documents (*.doc)|*.DOC|Any file (*.*)|*.*';
  SaveDialog1.Filter := 'Documents (*.doc)|*.DOC|Any file (*.*)|*.*';
end;

procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    if FileExists(OpenDialog1.FileName) then
      OleContainer1.CreateObjectFromFile(OpenDialog1.FileName, false)
    else
      raise Exception.Create('File not found.');
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
    if FileExists(SaveDialog1.FileName) then
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      OleContainer1.SaveAsDocument(SaveDialog1.FileName);
end;
```

See Also

CreateLinkToFile (see page 5977)

CreateObject (see page 5977)

CreateObjectFromInfo (see page 5979)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.10 TOleContainer.CreateObjectFromInfo Method

Creates an OLE object matching the specified TCreateInfo (see page 6056) record.

Pascal

```
procedure CreateObjectFromInfo(const CreateInfo: TCreateInfo);
```

C++

```
__fastcall CreateObjectFromInfo(const TCreateInfo CreateInfo);
```

Description

Call CreateObjectFromInfo to create a linked or embedded OLE object based on the specifications in the CreateInfo parameter. CreateInfo specifies whether the object should be created from a file, from a data transfer object, or from a class identifier, and provides the necessary supporting information. CreateInfo also indicates whether the OLE object should be displayed as an Icon, and allows the icon to be specified. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

Note: The CreateLinkToFile (see page 5977), CreateObject (see page 5977), and CreateObjectFromFile (see page 5978) methods are one-line calls for the most common types of OLE objects. They can be used instead of filling out the TCreateInfo (see page 6056) record.

See Also

CreateLinkToFile (see page 5977)

CreateObject (see page 5977)

CreateObjectFromFile (see page 5978)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.11 TOleContainer.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

62.1.1.4.12 TOleContainer.DblClick Method

OnDblClick event dispatcher.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click

OnDblClick

62.1.1.4.13 TOleContainer.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty

DefineProperty

DefineProperties

IsControl

62.1.1.4.14 TOleContainer.DestroyObject Method

Destroys the container's OLE object.

Pascal

```
procedure DestroyObject;
```

C++

```
__fastcall DestroyObject();
```

Description

Call DestroyObject to destroy the OLE object, discarding any unsaved changes the user made to it.

See Also

Close (see page 5976)

OleObject (see page 6008)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.15 TOleContainer.DoEnter Method

Respond to receiving input focus.

Pascal

```
procedure DoEnter; override;
```

C++

```
virtual __fastcall DoEnter();
```

Description

DoEnter is called automatically when the control receives the input focus. As implemented in TWinControl, DoEnter calls the OnEnter event handler, if defined.

Descendant classes that override DoEnter should always call the inherited method.

See Also

DoExit

OnEnter

62.1.1.4.16 TOleContainer.DoVerb Method

Requests that the OLE object perform some action.

Pascal

```
procedure DoVerb(Verb: Integer);
```

C++

```
__fastcall DoVerb(int Verb);
```

Description

Call DoVerb to request that the OLE object perform one of the actions listed by the ObjectVerbs (see page 6006) property.

OLE defines several verbs, such as ovShow (see page 6063) (to display the OLE object) and ovPrimary (see page 6062) (the default action, usually to activate the OLE object). OLE objects can also define their own custom verbs. Use the ObjectVerbs (see page 6006) property to get a list of those custom verbs.

See Also

ObjectVerbs (see page 6006)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.4.17 TOleContainer.GetIconMetaPict Method

Returns the global memory handle of a metafile containing the OLE object's icon.

Pascal

```
function GetIconMetaPict: HGlobal;
```

C++

```
__fastcall HGlobal GetIconMetaPict();
```

Description

Call GetIconMetaPict to obtain a metafile handle for drawing the OLE object's icon on another canvas. Use the global DestroyMetaPict (see page 6055) procedure to destroy the handle. If the OLE container is empty, GetIconMetaPict raises an exception.

See Also

DestroyMetaPict (see page 6055)

TMetafile (see page 2780)

TMetafileCanvas (see page 2794)

62.1.1.4.18 TOleContainer.GetPopupMenu Method

Returns the pop-up menu associated with the control.

Pascal

```
function GetPopupMenu: TPopupMenu; override;
```

C++

```
virtual __fastcall TPopupMenu GetPopupMenu();
```

Description

Call GetPopupMenu to find the popup menu associated with the control. Override GetPopupMenu to change which popup menu is associated with the control or to specify different popup menus depending on the state of the control.

GetPopupMenu returns the value of the Popup property.

See Also

PopupMenu

62.1.1.4.19 TOleContainer.InsertObjectDialog Method

Displays the Insert Object OLE dialog box.

Pascal

```
function InsertObjectDialog: Boolean;
```

C++

```
__fastcall Boolean InsertObjectDialog();
```

Description

Call InsertObjectDialog to let the user create an OLE object. The OLE object can be embedded or linked. InsertObjectDialog returns true if the dialog box was successfully displayed and the user chose the OK button. InsertObjectDialog returns false otherwise. If the user chose OK and there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateObjectFromInfo (see page 5979)

ObjectPropertiesDialog (see page 5985)

PasteSpecialDialog (see page 5986)

62.1.1.4.20 TOleContainer.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

62.1.1.4.21 TOleContainer.LoadFromFile Method

Loads an OLE object from the specified file.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString FileName);
```

Description

Call LoadFromFile to load an OLE object that was saved using the SaveToFile (see page 5989) method. If OldStreamFormat (see page 6007) is true, LoadFromFile loads files saved by an TOleContainer (see page 5969) object as well as files saved using the current format; if OldStreamFormat (see page 6007) is false, LoadFromFile will not load files saved by the library. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

LoadFromStream (see page 5984)

OldStreamFormat (see page 6007)

SaveToFile (see page 5989)

62.1.1.4.22 TOleContainer.LoadFromStream Method

Loads an OLE object from the specified stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to load an OLE object from a stream. If OldStreamFormat (see page 6007) is true, LoadFromStream loads OLE objects saved by a TOleContainer (see page 5969) object as well as OLE objects saved using the current format; if OldStreamFormat (see page 6007) is false, LoadFromStream will not load OLE objects saved by the library. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

LoadFromFile (see page 5984)

OldStreamFormat (see page 6007)

SaveToStream (see page 5989)

TStream

62.1.1.4.23 TOleContainer.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

62.1.1.4.24 TOleContainer.ObjectPropertiesDialog Method

Displays the OLE Object Properties property sheet.

Pascal

```
function ObjectPropertiesDialog: Boolean;
```

C++

```
__fastcall Boolean ObjectPropertiesDialog();
```

Description

Call ObjectPropertiesDialog to let the user see and edit the properties of the OLE object. Any changes the user makes in the property sheet are automatically made to the OLE object. ObjectPropertiesDialog returns true if the property sheet was successfully displayed and the user chose the OK button. It returns false otherwise.

If the OLE container is empty, ObjectPropertiesDialog raises an exception.

See Also

DoVerb (see page 5981)

62.1.1.4.25 TOleContainer.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT

messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

62.1.1.4.26 TOleContainer.Paste Method

Pastes the contents of the Windows clipboard as an embedded object.

Pascal

```
procedure Paste;
```

C++

```
__fastcall Paste();
```

Description

Call Paste to embed the object on the clipboard in the OLE container. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CanPaste (see page 5997)

CreateObjectFromInfo (see page 5979)

PasteSpecialDialog (see page 5986)

62.1.1.4.27 TOleContainer.PasteSpecialDialog Method

Displays the Paste (see page 5986) Special OLE dialog box.

Pascal

```
function PasteSpecialDialog: Boolean;
```

C++

```
__fastcall Boolean PasteSpecialDialog();
```

Description

Call PasteSpecialDialog to let the user control how the contents of the Windows clipboard are pasted into the container. The Paste (see page 5986) Special dialog box lets the user select the format of the data, whether it should be embedded or linked, and whether the OLE object should be displayed as an icon (and if so, to choose a different icon). PasteSpecialDialog returns true if the dialog box was successfully displayed and the user chose the OK button. It returns false otherwise. If the user chose the OK button when there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CanPaste (see page 5997)

InsertObjectDialog (see page 5982)

Paste (see page 5986)

62.1.1.4.28 TOleContainer.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the component supports the interface specified by IID, and if so, returns a reference to that interface as the Obj parameter. If the component does not support the interface, the Obj parameter returns nil (Delphi) or NULL (C++).

Note: For components that act as COM object wrapper, QueryInterface calls the QueryInterface method of the internal COM object.

See Also

_AddRef

_Release

QueryInterface

QueryInterface

Invoke

ComObject

62.1.1.4.29 TOleContainer.Run Method

Runs the OLE object's server application, but doesn't open or activate the OLE object itself.

Pascal

```
procedure Run;
```

C++

```
__fastcall Run();
```

Description

Call Run to ensure that the server application is running. Once the server application is running, opening and activating the OLE object happens much more quickly. An OLE object must already be loaded in the container before calling Run, Otherwise, Run raises an exception.

See Also

State (see page 6017)

62.1.1.4.30 TOleContainer.SaveAsDocument Method

Saves the OLE object to the specified file in native OLE Document format.

Pascal

```
procedure SaveAsDocument(const FileName: string);
```

C++

```
__fastcall SaveAsDocument(const UnicodeString FileName);
```

Description

Call SaveAsDocument to save the OLE object in its native OLE Document format. An OLE object must already be loaded in the container before calling SaveAsDocument. Otherwise, SaveAsDocument raises an exception.

Note: Saving an OLE object to a file or a stream limits the ability of another application, such as Word, to open the file, even if it is saved with the correct extension. Saving to a file or a stream loses information necessary to open such a file. Use SaveAsDocument when needing to save files in a native OLE Document format.

Delphi Examples:

```
{
This example uses an OLE container and a main menu on a form.
When the user clicks the Open or Save options in the main menu,
an open/save dialog is displayed and by default,
the user is required to enter the name of a Word document.
Files with other extensions supported by the OLE container
can also be accessed. An additional test is made to assure that
the file to open exists and that no file overwriting is
going to take place.
}
procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenDialog1.Filter := 'Documents (*.doc)|*.DOC|Any file (*.*)|*.*';
  SaveDialog1.Filter := 'Documents (*.doc)|*.DOC|Any file (*.*)|*.*';
end;

procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    if FileExists(OpenDialog1.FileName) then
      OleContainer1.CreateObjectFromFile(OpenDialog1.FileName, false)
    else
      raise Exception.Create('File not found.');
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
    if FileExists(SaveDialog1.FileName) then
      raise Exception.Create('File already exists. Cannot overwrite.')
    else
      OleContainer1.SaveAsDocument(SaveDialog1.FileName);
end;
```

See Also

SaveToFile (see page 5989)

SaveToStream (see page 5989)

62.1.1.4.31 TOleContainer.SaveToFile Method

Saves the OLE object to the specified file.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString FileName);
```

Description

Call SaveToFile to save the OLE object. If CopyOnSave (see page 6002) is true, SaveToFile first creates a temporary OLE object that compresses redundant data, saving space in the file that is saved. If OldStreamFormat (see page 6007) is true, SaveToFile writes a file that can be read by a TOleContainer (see page 5969) component.

An OLE object must already be loaded in the container before calling SaveToFile. Otherwise, SaveToFile raises an exception.

See Also

Copy (see page 5976)

CopyOnSave (see page 6002)

LoadFromFile (see page 5984)

OldStreamFormat (see page 6007)

SaveToStream (see page 5989)

62.1.1.4.32 TOleContainer.SaveToStream Method

Saves the OLE object to the specified stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the OLE object. If CopyOnSave (see page 6002) is true, SaveToStream first creates a temporary OLE object that compresses redundant data, reducing the information that must be saved. If OldStreamFormat (see page 6007) is true, SaveToStream writes the object in a format that can be read by a TOleContainer (see page 5969) component.

An OLE object must already be loaded in the container before calling SaveToStream. Otherwise, SaveToStream raises an exception.

See Also

Copy (see page 5976)

CopyOnSave (see page 6002)

LoadFromStream (see page 5984)

OldStreamFormat (see page 6007)

SaveToFile (see page 5989)

62.1.1.4.33 TOleContainer.UpdateObject Method

Refreshes the OLE object.

Pascal

```
procedure UpdateObject;
```

C++

```
__fastcall UpdateObject();
```

Description

Linked (see page 6005) OLE objects and embedded objects that contain linked OLE objects get out of date when the source of the link is updated. UpdateObject rereads the source to ensure that the OLE object has current data. If there is no OLE object loaded in the container, UpdateObject has no effect.

See Also

Linked (see page 6005)

UpdateVerbs (see page 5990)

62.1.1.4.34 TOleContainer.UpdateVerbs Method

Refreshes the list of verbs the OLE object can respond to.

Pascal

```
procedure UpdateVerbs;
```

C++

```
__fastcall UpdateVerbs();
```

Description

TOleContainer (see page 5969) automatically calls UpdateVerbs when the ObjectVerbs (see page 6006) property is first read, but some OLE objects change their verbs when they perform some actions. The Media Clip OLE object, for example, changes its Play verb to Stop while running. Call UpdateVerbs to ensure that the list of verbs is appropriate to the state of the OLE object.

An OLE object must already be loaded in the container before calling UpdateVerbs. Otherwise, UpdateVerbs raises an exception.

See Also

AutoVerbMenu (see page 5997)

ObjectVerbs (see page 6006)

62.1.1.5 TOleContainer Properties

The properties of the TOleContainer class are listed here.

Public Properties

	Name	Description
R	CanPaste (see page 5997)	Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.
R	Linked (see page 6005)	Indicates whether the OLE object is linked or embedded.
	Modified (see page 6006)	Indicates whether the OLE object has been modified.

	NewInserted (see page 6006)	Indicates whether the OLE object was created rather than pasted in or created as a link to a file.
	ObjectVerbs (see page 6006)	Lists the verbs that the OLE object supports.
	OleClassName (see page 6007)	Specifies the class name of the OLE object.
	OleObject (see page 6008)	Represents the OLE object hosted by the OLE container.
	OleObjectInterface (see page 6008)	Provides access to the IOleObject interface for the OLE object.
	PrimaryVerb (see page 6015)	Specifies the index of the primary (default) verb for the OLE object.
	SourceDoc (see page 6017)	Specifies the name of the source document for a linked OLE object.
	State (see page 6017)	Describes the state of the OLE object.
	StorageInterface (see page 6018)	Provides access to the IStorage interface for the OLE object.

Published Properties

	Name	Description
	Align (see page 5991)	Determines how the control aligns within its container (parent control).
	AllowActiveDoc (see page 5995)	Specifies whether the OLE container supports the IOleDocumentSite interface.
	AllowInPlace (see page 5995)	Determines whether the OLE container allows in-place activation.
	Anchors (see page 5996)	Specifies how the control is anchored to its parent.
	AutoActivate (see page 5996)	Determines how the object in an OLE container is activated.
	AutoVerbMenu (see page 5997)	Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.
	BorderStyle (see page 5997)	Determines whether a single line is drawn around the OLE container.
	Caption (see page 5998)	Specifies a text string that identifies the control to the user.
	Color (see page 6000)	Specifies the background color of the control.
	Constraints (see page 6001)	Specifies the size constraints for the control.
	CopyOnSave (see page 6002)	Determines whether the SaveToFile (see page 5989) and SaveToStream (see page 5989) methods first create a copy of the OLE object.
	Ctl3D (see page 6002)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 6003)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 6003)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 6005)	Controls whether the control responds to mouse, keyboard, and timer events.
	Iconic (see page 6005)	Determines whether the OLE object is displayed as an icon.
	OldStreamFormat (see page 6007)	Indicates whether the OLE object is loaded and saved using the VCL format.
	ParentColor (see page 6008)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6009)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 6009)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 6010)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 6015)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SizeMode (see page 6017)	Determines how the OLE object is sized in the OLE container.
	TabOrder (see page 6018)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6019)	Determines if the user can tab to a control.
	Visible (see page 6019)	Determines whether the component appears onscreen.

62.1.1.5.1 TOleContainer.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
```

```
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

62.1.1.5.2 TOleContainer.AllowActiveDoc Property

Specifies whether the OLE container supports the IOleDocumentSite interface.

Pascal

```
property AllowActiveDoc: Boolean;
```

C++

```
__property Boolean AllowActiveDoc;
```

Description

Set AllowActiveDoc to true to specify that the OLE container can house an ActiveDoc (DocObject). Set AllowActiveDoc to false to limit the OLE container to hosting OLE controls that use the more usual interfaces such as IOleClientSite. ActiveDoc-enabled controls have a slightly different border. Set AllowActiveDoc to false to maintain a consistent appearance in forms that do not need to support the IOleDocumentSite interface.

See Also

AllowInPlace (see page 5995)

62.1.1.5.3 TOleContainer.AllowInPlace Property

Determines whether the OLE container allows in-place activation.

Pascal

```
property AllowInPlace: Boolean;
```

C++

```
__property Boolean AllowInPlace;
```

Description

Set AllowInPlace to specify what happens when the user activates the OLE object. If AllowInPlace is true and Iconic (see page 6005) is false, the OLE object is activated in place. If AllowInPlace is false, the OLE object is activated in a separate window. The default value is true.

See Also

AllowActiveDoc (see page 5995)

AutoActivate (see page 5996)

Iconic (see page 6005)

62.1.1.5.4 TOleContainer.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

62.1.1.5.5 TOleContainer.AutoActivate Property

Determines how the object in an OLE container is activated.

Pascal

```
property AutoActivate: TAutoActivate;
```

C++

```
__property TAutoActivate AutoActivate;
```

Description

Set AutoActivate to specify what actions activate the OLE object. AutoActivate takes one of the values in the following table:

Values	Meaning
aaManual	The OLE object must be activated programmatically by calling DoVerb (see page 5981) (ovShow (see page 6063)).
aaGetFocus	The OLE object is activated when the OLE container gets focus (by clicking on it with the mouse or pressing Tab to move the focus to it).
aaDoubleClick	(Default) The OLE object is activated by double-clicking it or pressing Enter while the container has the focus.

See Also

AllowActiveDoc (see page 5995)

DoVerb (see page 5981)

62.1.1.5.6 TOleContainer.AutoVerbMenu Property

Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.

Pascal

```
property AutoVerbMenu: Boolean;
```

C++

```
__property Boolean AutoVerbMenu;
```

Description

Use AutoVerbMenu to specify whether the OLE object can replace the OLE container's popup menu. If AutoVerbMenu is true (the default), the OLE object replaces any existing PopupMenu (see page 6010) property. If AutoVerbMenu is false, no popup menu is automatically created.

See Also

DoVerb (see page 5981)

ObjectVerbs (see page 6006)

PopupMenu

62.1.1.5.7 TOleContainer.BorderStyle Property

Determines whether a single line is drawn around the OLE container.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether the OLE container should be framed by a single line. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

See Also

Ctl3D

62.1.1.5.8 TOleContainer.CanPaste Property

Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.

Pascal

```
property CanPaste: Boolean;
```

C++

```
__property Boolean CanPaste;
```

Description

Read CanPaste before trying to paste or paste link an OLE object from the clipboard.

See Also

Paste (see page 5986)

PasteSpecialDialog (see page 5986)

62.1.1.5.9 TOleContainer.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
```

```
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

62.1.1.5.10 TOleContainer.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

62.1.1.5.11 TOleContainer.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

62.1.1.5.12 TOleContainer.CopyOnSave Property

Determines whether the SaveToFile (see page 5989) and SaveToStream (see page 5989) methods first create a copy of the OLE object.

Pascal

```
property CopyOnSave: Boolean;
```

C++

```
__property Boolean CopyOnSave;
```

Description

When CopyOnSave is true, the SaveToFile (see page 5989) and SaveToStream (see page 5989) methods create a temporary OLE object, compressing redundant data to save space. This compressed copy is then saved to the file or stream. When CopyOnSave is false, the OLE object is saved directly to the file or stream without using a temporary copy to compress redundant information.

Set CopyOnSave to false when the container holds very large embedded objects and there might not be enough memory to make the temporary copy. The default value is true.

See Also

OleObject (see page 6008)

SaveToFile (see page 5989)

SaveToStream (see page 5989)

62.1.1.5.13 TOleContainer.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

62.1.1.5.14 TOleContainer.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

62.1.1.5.15 TOleContainer.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
```

```
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

62.1.1.5.16 TOleContainer.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

62.1.1.5.17 TOleContainer.Iconic Property

Determines whether the OLE object is displayed as an icon.

Pascal

```
property Iconic: Boolean;
```

C++

```
__property Boolean Iconic;
```

Description

Set Iconic to specify whether the OLE container displays the icon for the server application instead of the image of the object. If Iconic is true, the OLE object is displayed as an icon; if false, it's displayed as it would be in the server application.

See Also

GetIconMetaPict (see page 5982)

62.1.1.5.18 TOleContainer.Linked Property

Indicates whether the OLE object is linked or embedded.

Pascal

```
property Linked: Boolean;
```

C++

```
__property Boolean Linked;
```

Description

Read Linked to determine whether the OLE object is linked to the OLE server. If Linked is true, the OLE object is linked; if false, it's embedded. An OLE object must already be loaded in the container before accessing the Linked property.

See Also

UpdateObject (see page 5990)

62.1.1.5.19 TOleContainer.Modified Property

Indicates whether the OLE object has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use Modified to determine when the OLE object has been modified (including being deleted or replaced by another OLE object). To keep track of incremental changes, set Modified to false and later check it to see if the OLE object has been modified after since Modified was set to false. If there is no OLE object loaded into the container, Modified returns false.

See Also

OleObject (see page 6008)

62.1.1.5.20 TOleContainer.NewInserted Property

Indicates whether the OLE object was created rather than pasted in or created as a link to a file.

Pascal

```
property NewInserted: Boolean;
```

C++

```
__property Boolean NewInserted;
```

Description

Read NewInserted to determine whether the OLE object was newly created via a call to InsertObjectDialog (see page 5982). If there is no OLE object loaded into the container, NewInserted returns false. If NewInserted is true after a call to InsertObjectDialog (see page 5982), call DoVerb (see page 5981)(ovShow (see page 6063)) to let the user edit the object.

See Also

DoVerb (see page 5981)

InsertObjectDialog (see page 5982)

OleObject (see page 6008)

62.1.1.5.21 TOleContainer.ObjectVerbs Property

Lists the verbs that the OLE object supports.

Pascal

```
property ObjectVerbs: TStrings;
```

C++

```
__property TStrings ObjectVerbs;
```

Description

Read ObjectVerbs to determine the verbs that can be used with the OLE object. The verb names can have embedded ampersand (&) characters to indicate shortcut keys. An OLE object must already be loaded in the container before accessing the ObjectVerbs property.

See Also

AutoVerbMenu (see page 5997)

DoVerb (see page 5981)

PrimaryVerb (see page 6015)

TStrings

62.1.1.5.22 TOleContainer.OldStreamFormat Property

Indicates whether the OLE object is loaded and saved using the VCL format.

Pascal

```
property OldStreamFormat: Boolean;
```

C++

```
__property Boolean OldStreamFormat;
```

Description

Set OldStreamFormat to true to read OLE objects that were saved to file using VCL or to save OLE objects that will be used by VCL. If OldStreamFormat is false (the default), the LoadFromFile (see page 5984), LoadFromStream (see page 5984), SaveToFile (see page 5989), and SaveToStream (see page 5989) methods use the newer format that contains extra header information. The extra header information is used to validate a file or stream. This property is provided for backward compatibility only.

See Also

LoadFromFile (see page 5984)

LoadFromStream (see page 5984)

SaveToFile (see page 5989)

SaveToStream (see page 5989)

62.1.1.5.23 TOleContainer.OleClassName Property

Specifies the class name of the OLE object.

Pascal

```
property OleClassName: string;
```

C++

```
__property UnicodeString OleClassName;
```

Description

Read OleClassName to obtain the class name (programmatic identifier) for the OLE object. The class name is a string describing the class of OLE object that is registered in the Windows registry. An OLE object must already be loaded in the container before accessing the OleClassName property.

See Also

OleObject (see page 6008)

62.1.1.5.24 TOleContainer.OleObject Property

Represents the OLE object hosted by the OLE container.

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

Read OleObject to access the OLE object hosted by the OLE container as a variant. To interact with the OLE object using the IOleObject interface, use the OleObjectInterface (see page 6008) property instead. If the OLE object is not loaded in the container before reading OleObject, an EOleError exception is raised.

See Also

EOleError

OleObjectInterface (see page 6008)

62.1.1.5.25 TOleContainer.OleObjectInterface Property

Provides access to the IOleObject interface for the OLE object.

Pascal

```
property OleObjectInterface: IOleObject;
```

C++

```
__property IOleObject OleObjectInterface;
```

Description

Use OleObjectInterface for low-level access to the OLE API. If there is no OLE object loaded into the container, OleObjectInterface returns nil (Delphi) or NULL (C++).

See Also

OleObject (see page 6008)

Creating Clients for Servers That Do Not Have a Type Library

62.1.1.5.26 TOleContainer.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its

own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

62.1.1.5.27 TOleContainer.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

62.1.1.5.28 TOleContainer.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

62.1.1.5.29 TOleContainer.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
```

```
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
```

```
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

62.1.1.5.30 TOleContainer.PrimaryVerb Property

Specifies the index of the primary (default) verb for the OLE object.

Pascal

```
property PrimaryVerb: Integer;
```

C++

```
__property int PrimaryVerb;
```

Description

Use PrimaryVerb to determine which verb in the ObjectVerbs (see page 6006) list specifies the default action for the OLE object. In a call to the DoVerb (see page 5981) method, DoVerb (see page 5981)(ovPrimary (see page 6062)) is the same as DoVerb (see page 5981)(PrimaryVerb). An OLE object must already be loaded in the container before accessing the PrimaryVerb property.

See Also

DoVerb (see page 5981)

ObjectVerbs (see page 6006)

62.1.1.5.31 TOleContainer.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

62.1.1.5.32 TOleContainer.SizeMode Property

Determines how the OLE object is sized in the OLE container.

Pascal

```
property SizeMode: TSizeMode;
```

C++

```
__property TSizeMode SizeMode;
```

Description

Set SizeMode to specify how the OLE object is fit into the client area of the OLE container. The following table gives the possible values :

Value	Meaning
smClip	(Default)- Displays the OLE object at its normal size, clipping any parts that don't fit within the container.
smCenter	Displays the OLE object at its normal size, centering it within the container.
smScale	Scales or shrinks the view of the OLE object to fit within the container, by scaling width and height proportionally.
smStretch	Scales or shrinks the view of the OLE object to fill the OLE container, without regard to preserving the proportions of the OLE object.
smAutoSize	Displays the OLE object at its normal size and automatically resizes the container to fit the size of the OLE object.

See Also

AllowInPlace (see page 5995)

ClientRect

62.1.1.5.33 TOleContainer.SourceDoc Property

Specifies the name of the source document for a linked OLE object.

Pascal

```
property SourceDoc: string;
```

C++

```
__property UnicodeString SourceDoc;
```

Description

Read SourceDoc to obtain the name of the OLE object when it is linked. An OLE object must already be loaded in the container before accessing the SourceDoc property. If the OLE object isn't linked, SourceDoc returns an empty string.

See Also

Linked (see page 6005)

OleClassName (see page 6007)

62.1.1.5.34 TOleContainer.State Property

Describes the state of the OLE object.

Pascal

```
property State: TObjectState;
```

C++

```
__property TObjectState State;
```

Description

Read State to determine the state of the OLE object and its server application. State has one of the following values:

Values	Meaning
osEmpty	There is no OLE object in the container.
osLoaded	There is an OLE object in the container, but its server application isn't currently running.
osRunning	The OLE object's server is running.
osOpen	The OLE object is open in a separate window.
osInPlaceActive	The OLE object is activated in place, but hasn't yet merged its menus or toolbars. This is an intermediate state; the State property will shift to osUIActive as soon as the menus and toolbars have been merged.
osUIActive	The OLE object is activated in place and menus and toolbars have been merged

See Also

AllowInPlace (see page 5995)

Iconic (see page 6005)

62.1.1.5.35 TOleContainer.StorageInterface Property

Provides access to the IStorage interface for the OLE object.

Pascal

```
property StorageInterface: IStorage;
```

C++

```
__property IStorage StorageInterface;
```

Description

Use StorageInterface for low-level access to the OLE API. If there is no OLE object loaded into the container, StorageInterface returns nil (Delphi) or NULL (C++).

See Also

OleObjectInterface (see page 6008)

62.1.1.5.36 TOleContainer.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

62.1.1.5.37 TOleContainer.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

62.1.1.5.38 TOleContainer.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

62.1.1.6 TOleContainer Events

The events of the TOleContainer class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6021)	Occurs immediately after the OLE object is activated.
	OnDeactivate (see page 6022)	Occurs immediately after the OLE object is deactivated.
	OnDragDrop (see page 6022)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 6025)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 6028)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6029)	Occurs when a control receives the input focus.
	OnExit (see page 6032)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 6035)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6038)	Occurs when a key is pressed.
	OnKeyUp (see page 6039)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 6041)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 6042)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 6044)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 6047)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnObjectMove (see page 6049)	Occurs when the OLE container is notified that the OLE object has moved or resized.
	OnResize (see page 6049)	Occurs immediately after the OLE container window is resized.
	OnStartDrag (see page 6049)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

62.1.1.6.1 TOleContainer.OnActivate Event

Occurs immediately after the OLE object is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to take specific action when the OLE object becomes active.

See Also

AutoActivate (see page 5996)

DoVerb (see page 5981)

OnDeactivate (see page 6022)

State (see page 6017)

62.1.1.6.2 TOleContainer.OnDeactivate Event

Occurs immediately after the OLE object is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to respond when the OLE object is deactivated.

See Also

Close (see page 5976)

OnActivate (see page 6021)

State (see page 6017)

62.1.1.6.3 TOleContainer.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
```

```
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

62.1.1.6.4 TOleContainer.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

62.1.1.6.5 TOleContainer.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

62.1.1.6.6 TOleContainer.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
```

```
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

62.1.1.6.7 TOleContainer.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

62.1.1.6.8 TOleContainer.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

62.1.1.6.9 TOleContainer.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

62.1.1.6.10 TOleContainer.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

62.1.1.6.11 TOleContainer.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

62.1.1.6.12 TOleContainer.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

62.1.1.6.13 TOleContainer.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

62.1.1.6.14 TOleContainer.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

62.1.1.6.15 TOleContainer.OnObjectMove Event

Occurs when the OLE container is notified that the OLE object has moved or resized.

Pascal

```
property OnObjectMove: TObjectMoveEvent;
```

C++

```
__property TObjectMoveEvent OnObjectMove;
```

Description

Write an OnObjectMove event handler to respond when the OLE object changes size or position. The Bounds parameter specifies the boundaries of the in-place object in the client coordinates of OLE container. OnObjectMove occurs before the OLE object's image is updated to reflect the change.

See Also

OnResize (see page 6049)

62.1.1.6.16 TOleContainer.OnResize Event

Occurs immediately after the OLE container window is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Write an OnResize event handler to respond to changes in the size of the OLE container window. If SizeMode (see page 6017) is smAutoSize, the container window changes size whenever the OLE object resizes.

See Also

OnObjectMove (see page 6049)

SizeMode (see page 6017)

62.1.1.6.17 TOleContainer.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse

button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

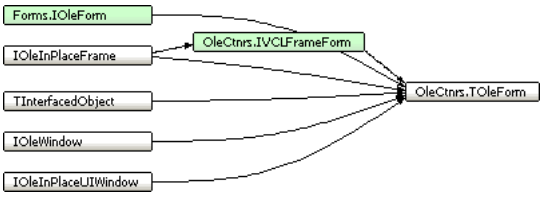
OnDragOver

OnDragDrop

62.1.2 TOleForm Class

TOleForm is an object that provides OLE support for a TForm (see page 2241) object.

Class Hierarchy



Pascal

```
TOleForm = class(TInterfacedObject, IOleForm, IOleWindow, IOleInPlaceUIWindow,
IOleInPlaceFrame, IVCLFrameForm);
```

C++

```
class TOleForm : public TInterfacedObject, public IOleForm, public IOleWindow, public
IOleInPlaceUIWindow, public IOleInPlaceFrame, public IVCLFrameForm;
```

File

OleCtnrs

Description

TOleForm implements the IOleForm (see page 2572), IOleWindow, IOleInPlaceUIWindow, and IOleInPlaceFrame interfaces, as well as an interface for internal use. The implementation of the TOleForm object is used internally by the Visual Component Library.

62.1.2.1 TOleForm Members

The following tables list the members exposed by TOleForm.

Public Methods

	Name	Description
	AddContainer (see page 6054)	This is AddContainer, a member of class IVCLFrameForm.
	ClearBorderSpace (see page 6054)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Create (see page 6051)	Create creates a new instance of the TOleForm (see page 6050) object.
	Destroy (see page 6051)	Destroy frees the memory associated with this instance of the TOleForm (see page 6050) object.
	Form (see page 6054)	This is Form, a member of class IVCLFrameForm.
	OnDestroy (see page 2572)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 2573)	This is OnResize, a member of class IOleForm.
	RemoveContainer (see page 6054)	This is RemoveContainer, a member of class IVCLFrameForm.

62.1.2.2 TOleForm.Create Constructor

Create creates a new instance of the TOleForm (see page 6050) object.

Pascal

```
constructor Create(Form: TCustomForm);
```

C++

```
__fastcall TOleForm(TCustomForm Form);
```

Description

Create is used to create a new instance of the TOleForm (see page 6050) object. Form (see page 6054) is the form upon which to base the new TOleForm (see page 6050) object.

See Also

Destroy (see page 6051)

62.1.2.3 TOleForm.Destroy Destructor

Destroy frees the memory associated with this instance of the TOleForm (see page 6050) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleForm();
```

Description

Do not call Destroy directly. Instead, call Free, which calls Destroy if this instance of the TOleForm (see page 6050) object is not nil.

Destroy frees private objects allocated and used by this instance of the TOleForm (see page 6050) object.

See Also

Create (see page 6051)

62.2 Interfaces

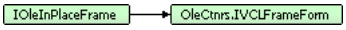
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IVCLFrameForm (see page 6053)	This is class OleCtnrs.IVCLFrameForm.

62.2.1 IVCLFrameForm Interface

Class Hierarchy



Pascal

```
IVCLFrameForm = interface(IOleInPlaceFrame);
```

C++

```
__interface IVCLFrameForm : public IOleInPlaceFrame;
```

File

OleCtnrs

Description

This is class OleCtnrs.IVCLFrameForm.

62.2.1.1 IVCLFrameForm Members

The following tables list the members exposed by IVCLFrameForm.

Public Methods

	Name	Description
	AddContainer (see page 6054)	This is AddContainer, a member of class IVCLFrameForm.
	ClearBorderSpace (see page 6054)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Form (see page 6054)	This is Form, a member of class IVCLFrameForm.
	RemoveContainer (see page 6054)	This is RemoveContainer, a member of class IVCLFrameForm.

62.2.1.2 IVCLFrameForm Methods

The methods of the IVCLFrameForm class are listed here.

Public Methods

	Name	Description
	AddContainer (see page 6054)	This is AddContainer, a member of class IVCLFrameForm.
	ClearBorderSpace (see page 6054)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Form (see page 6054)	This is Form, a member of class IVCLFrameForm.

	RemoveContainer (see page 6054)	This is RemoveContainer, a member of class IVCLFrameForm.

62.2.1.2.1 IVCLFrameForm.AddContainer Method

Pascal

```
procedure AddContainer(Instance: TOleContainer);
```

C++

```
__fastcall AddContainer(TOleContainer Instance);
```

Description

This is AddContainer, a member of class IVCLFrameForm.

62.2.1.2.2 IVCLFrameForm.ClearBorderSpace Method

Pascal

```
procedure ClearBorderSpace;
```

C++

```
__fastcall ClearBorderSpace();
```

Description

This is ClearBorderSpace, a member of class IVCLFrameForm.

62.2.1.2.3 IVCLFrameForm.Form Method

Pascal

```
function Form: TCustomForm;
```

C++

```
__fastcall TCustomForm Form();
```

Description

This is Form, a member of class IVCLFrameForm.

62.2.1.2.4 IVCLFrameForm.RemoveContainer Method

Pascal

```
procedure RemoveContainer(Instance: TOleContainer);
```

C++

```
__fastcall RemoveContainer(TOleContainer Instance);
```

Description

This is RemoveContainer, a member of class IVCLFrameForm.

62.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DestroyMetaPict (see page 6055)	Frees a metafile resource given its handle.

62.3.1 OleCtnrs.DestroyMetaPict Function

Frees a metafile resource given its handle.

Pascal

```
procedure DestroyMetaPict(MetaPict: HGlobal);
```

C++

```
DestroyMetaPict(HGlobal MetaPict);
```

File

OleCtnrs

Description

Call DestroyMetaPict to free a metafile obtained using the GetIconMetaPict method of TOleContainer (see page 5969).

You do not need to use DestroyMetaPict to free metafiles represented using TMetafile (see page 2780). These are freed automatically when the TMetafile (see page 2780) instance is destroyed. However, if you assign a new value to the Handle property of TMetafile (see page 2780), thereby replacing its old metafile, you must free the old metafile by calling DestroyMetaPict.

See Also

TOleContainer (see page 5969)

TMetafile

TMetafile

62.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoActivate (see page 6056)	This is record OleCtnrs.TAutoActivate.
	TCreateType (see page 6057)	This is record OleCtnrs.TCreateType.
	TObjectState (see page 6057)	This is record OleCtnrs.TObjectState.
	TSizeMode (see page 6058)	This is record OleCtnrs.TSizeMode.

Records

	Name	Description
	TCreateInfo (see page 6056)	This is record OleCtnrs.TCreateInfo.
	TVerbInfo (see page 6058)	This is record OleCtnrs.TVerbInfo.

62.4.1 OleCtnrs.TAutoActivate Enumeration

Pascal

```
TAutoActivate = (
  aaManual,
  aaGetFocus,
  aaDoubleClick
);
```

C++

```
enum TAutoActivate {
  aaManual,
  aaGetFocus,
  aaDoubleClick
};
```

File

OleCtnrs

Description

This is record OleCtnrs.TAutoActivate.

62.4.2 OleCtnrs.TCreateInfo Record

Pascal

```
TCreateInfo = record
  CreateType: TCreateType;
  ShowAsIcon: Boolean;
  IconMetaPict: HGlobal;
  ClassID: TCLSID;
  FileName: WideString;
```

```
    DataObject: IDataObject;
  end;
```

C++

```
  struct TCreateInfo {
    TCreateType CreateType;
    Boolean ShowAsIcon;
    HGlobal IconMetaPict;
    TCLSID ClassID;
    BSTR FileName;
    IDataObject DataObject;
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TCreateInfo.

62.4.3 OleCtnrs.TCreateType Enumeration

Pascal

```
  TCreateType = (
    ctNewObject,
    ctFromFile,
    ctLinkToFile,
    ctFromData,
    ctLinkFromData
  );
```

C++

```
  enum TCreateType {
    ctNewObject,
    ctFromFile,
    ctLinkToFile,
    ctFromData,
    ctLinkFromData
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TCreateType.

62.4.4 OleCtnrs.TObjectState Enumeration

Pascal

```
  TObjectState = (
    osEmpty,
    osLoaded,
    osRunning,
    osOpen,
    osInPlaceActive,
```

```
    osUIActive
  );
```

C++

```
  enum TObjectState {
    osEmpty,
    osLoaded,
    osRunning,
    osOpen,
    osInPlaceActive,
    osUIActive
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TObjectState.

62.4.5 OleCtnrs.TSizeMode Enumeration

Pascal

```
  TSizeMode = (
    smClip,
    smCenter,
    smScale,
    smStretch,
    smAutoSize
  );
```

C++

```
  enum TSizeMode {
    smClip,
    smCenter,
    smScale,
    smStretch,
    smAutoSize
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TSizeMode.

62.4.6 OleCtnrs.TVerbInfo Record

Pascal

```
  TVerbInfo = packed record
    Verb: Smallint;
    Flags: Word;
  end;
```

C++

```
struct TVerbInfo {
  Smallint Verb;
  Word Flags;
};
```

File

OleCtnrs

Description

This is record OleCtnrs.TVerbInfo.

62.5 Types

The following table lists types in this documentation.

Types

Name	Description
TObjectMoveEvent (see page 6060)	This is type OleCtnrs.TObjectMoveEvent.

62.5.1 OleCtnrs.TObjectMoveEvent Type

Pascal

```
TObjectMoveEvent = procedure (OleContainer: TOleContainer; const Bounds: TRect) of object;
```

C++

```
(OleContainer: TOleContainer; const Bounds: TRect) ( TObjectMoveEvent)();
```

File

OleCtnrs

Description

This is type OleCtnrs.TObjectMoveEvent.

62.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ovDiscardUndoState (see page 6061)	This is constant OleCtnrs.ovDiscardUndoState.
ovHide (see page 6061)	This is constant OleCtnrs.ovHide.
ovInPlaceActivate (see page 6062)	This is constant OleCtnrs.ovInPlaceActivate.
ovOpen (see page 6062)	This is constant OleCtnrs.ovOpen.
ovPrimary (see page 6062)	This is constant OleCtnrs.ovPrimary.
ovShow (see page 6063)	This is constant OleCtnrs.ovShow.
ovUIActivate (see page 6063)	This is constant OleCtnrs.ovUIActivate.

62.6.1 OleCtnrs.ovDiscardUndoState Constant

Pascal

```
ovDiscardUndoState = -6;
```

C++

```
ovDiscardUndoState = -6;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovDiscardUndoState.

62.6.2 OleCtnrs.ovHide Constant

Pascal

```
ovHide = -3;
```

C++

```
ovHide = -3;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovHide.

62.6.3 OleCtnrs.ovInPlaceActivate Constant

Pascal

```
ovInPlaceActivate = -5;
```

C++

```
ovInPlaceActivate = -5;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovInPlaceActivate.

62.6.4 OleCtnrs.ovOpen Constant

Pascal

```
ovOpen = -2;
```

C++

```
ovOpen = -2;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovOpen.

62.6.5 OleCtnrs.ovPrimary Constant

Pascal

```
ovPrimary = -65536;
```

C++

```
ovPrimary = -65536;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovPrimary.

62.6.6 OleCtnrs.ovShow Constant

Pascal

```
ovShow = -1;
```

C++

```
ovShow = -1;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovShow.

62.6.7 OleCtnrs.ovUIActivate Constant

Pascal

```
ovUIActivate = -4;
```

C++

```
ovUIActivate = -4;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovUIActivate.

63 OleCtrls Namespace

This is namespace OleCtrls.

Topics

Name	Description
Classes (see page 6067)	The following table lists classes in this documentation.
Functions (see page 6122)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6124)	The following table lists structs, records, enums in this documentation.
Types (see page 6127)	The following table lists types in this documentation.

Classes

	Name	Description
	EOleCtrlError (see page 6067)	EOleCtrlError is the exception class for ActiveX controls.
	TEnumPropDesc (see page 6067)	TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.
	TEventDispatch (see page 6070)	This is class OleCtrls.TEventDispatch.
	TOleControl (see page 6074)	TOleControl is the base class from which ActiveX controls are derived.

Functions

	Name	Description
	EmptyParam (see page 6122)	This is function OleCtrls.EmptyParam.
	FontToOleFont (see page 6122)	Returns a Variant containing an IFontDispatch interface that represents a TFont (see page 2730) object.
	OleFontToFont (see page 6123)	Fills in a TFont (see page 2730) structure to represent an IFontDispatch.

Structs, Records, Enums

	Name	Description
	TControlData (see page 6124)	This is record OleCtrls.TControlData.
	TControlData2 (see page 6125)	This is record OleCtrls.TControlData2.
	TEnumValue (see page 6125)	This is record OleCtrls.TEnumValue.

Types

Name	Description
PControlData (see page 6127)	This is type OleCtrls.PControlData.
PControlData2 (see page 6127)	This is type OleCtrls.PControlData2.
PEnumValueList (see page 6128)	This is type OleCtrls.PEnumValueList.
TEnumValueList (see page 6128)	This is type OleCtrls.TEnumValueList.

TOleEnum (see page 6128)	This is type OleCtrls.TOleEnum.
TServiceQuery (see page 6129)	This is type OleCtrls.TServiceQuery.

63.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOleCtrlError (see page 6067)	EOleCtrlError is the exception class for ActiveX controls.
	TEnumPropDesc (see page 6067)	TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.
	TEventDispatch (see page 6070)	This is class OleCtrls.TEventDispatch.
	TOleControl (see page 6074)	TOleControl is the base class from which ActiveX controls are derived.

63.1.1 EOleCtrlError Class

EOleCtrlError is the exception class for ActiveX controls.

Class Hierarchy

Pascal

```
EOleCtrlError = class(Exception);
```

C++

```
class EOleCtrlError : public Exception;
```

File

OleCtrls

Description

EOleCtrlError is raised when an application cannot link to an ActiveX control.

See Also

EOleError

63.1.2 TEnumPropDesc Class

TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.

Class Hierarchy

Pascal

```
TEnumPropDesc = class;
```

C++

```
class TEnumPropDesc;
```

File

OleCtrls

Description

Use TEnumPropDesc to interpret the strings used by an OLE control object to represent the values of one of its properties. TEnumPropDesc can provide a list of possible value strings to a callback function, or map between the string names of property values and the values themselves. TEnumPropDesc objects are made available through the GetEnumPropDesc method of an OLE control object.

See Also

TOleControl (see page 6074)

63.1.2.1 TEnumPropDesc Members

The following tables list the members exposed by TEnumPropDesc.

Public Methods

	Name	Description
	Create (see page 6068)	Creates an instance of TEnumPropDesc (see page 6067).
	Destroy (see page 6069)	Destroys an instance of TEnumPropDesc (see page 6067).
	GetStrings (see page 6069)	Calls the callback Proc for each property value string.
	StringToValue (see page 6070)	Maps a string representation of the property value to the integer value.
	ValueToString (see page 6070)	Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

63.1.2.2 TEnumPropDesc.Create Constructor

Creates an instance of TEnumPropDesc (see page 6067).

Pascal

```
constructor Create(DispID: Integer; ValueCount: Integer; const TypeInfo: ITypeInfo);
```

C++

```
__fastcall TEnumPropDesc(int DispID, int ValueCount, const ITypeInfo TypeInfo);
```

Description

Create is called by an OLE control object to create an instance of TEnumPropDesc (see page 6067) to represent the values of one of its properties. Applications should not need to create instances of TEnumPropDesc (see page 6067). When working with OLE properties, get the TEnumPropDesc (see page 6067) objects from the OLE control object.

DispID identifies the property of the OLE object, ValueCount is the number of distinct values the property can take TypeInfo is an ITypeInfo object that the constructor can use to capture the values and their strings.

See Also

GetEnumPropDesc (see page 6088)

63.1.2.3 TEnumPropDesc.Destroy Destructor

Destroys an instance of TEnumPropDesc (see page 6067).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEnumPropDesc();
```

Description

Do not call Destroy directly in an application. The TOleControl (see page 6074) object that owns the TEnumPropDesc (see page 6067) object will destroy it automatically.

Destroy frees the internal list used to map value strings to property values.

See Also

TOleControl (see page 6074)

System::TObject::Free

63.1.2.4 TEnumPropDesc Methods

The methods of the TEnumPropDesc class are listed here.

Public Methods

	Name	Description
	GetStrings (see page 6069)	Calls the callback Proc for each property value string.
	StringToValue (see page 6070)	Maps a string representation of the property value to the integer value.
	ValueToString (see page 6070)	Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

63.1.2.4.1 TEnumPropDesc.GetStrings Method

Calls the callback Proc for each property value string.

Pascal

```
procedure GetStrings(Proc: TGetStrProc);
```

C++

```
__fastcall GetStrings(TGetStrProc Proc);
```

Description

Use GetStrings to execute code for every property value string represented in the TEnumPropDesc (see page 6067) object. The callback Proc is called for every value, with the S parameter set to a string that gives the integer value followed by the string representation of that value.

For example, consider a TEnumPropDesc (see page 6067) object that represents a color property, with values equal to TColor (see page 2878) values. The callback Proc would be called for every built-in color name with strings of the form '255 - clRed (see page 2916)'.

See Also

StringToValue (see page 6070)

ValueToString (see page 6070)

63.1.2.4.2 TEnumPropDesc.StringToValue Method

Maps a string representation of the property value to the integer value.

Pascal

```
function StringToValue(const S: string): Integer;
```

C++

```
__fastcall int StringToValue(const UnicodeString S);
```

Description

Use StringToValue to map the string representation of a property value used by an OLE control object to an integer value that is a more accurate representation of the property value. If the property value is not an integer, the integer associated with the string can be a pointer to the value.

See Also

ValueToString (see page 6070)

63.1.2.4.3 TEnumPropDesc.ValueToString Method

Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

Pascal

```
function ValueToString(V: Integer): string;
```

C++

```
__fastcall UnicodeString ValueToString(int V);
```

Description

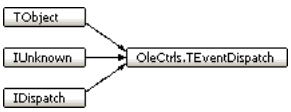
Use ValueToString to map the integer value of a property value to the string representation used by an OLE control object. If the property value is not an integer, the integer associated with the string can be a pointer to the value.

See Also

StringToValue (see page 6070)

63.1.3 TEventDispatch Class

Class Hierarchy



Pascal

```
TEventDispatch = class(TObject, IUnknown, IDispatch);
```

C++

```
class TEventDispatch : public TObject, public IUnknown, public IDispatch;
```

File

OleCtrls

Description

This is class OleCtrls.TEventDispatch.

63.1.3.1 TEventDispatch Members

The following tables list the members exposed by TEventDispatch.

Protected Methods

	Name	Description
	_AddRef (see page 6072)	This is _AddRef, a member of class TEventDispatch.
	_Release (see page 6072)	This is _Release, a member of class TEventDispatch.
	GetIDsOfNames (see page 6072)	This is GetIDsOfNames, a member of class TEventDispatch.
	GetTypeInfo (see page 6072)	This is GetTypeInfo, a member of class TEventDispatch.
	GetTypeInfoCount (see page 6073)	This is GetTypeInfoCount, a member of class TEventDispatch.
V	Invoke (see page 6073)	This is Invoke, a member of class TEventDispatch.
	QueryInterface (see page 6073)	This is QueryInterface, a member of class TEventDispatch.

Protected Properties

	Name	Description
R	Control (see page 6073)	This is Control, a member of class TEventDispatch.

Public Methods

	Name	Description
	Create (see page 6071)	This is Create, a member of class TEventDispatch.

63.1.3.2 TEventDispatch.Create Constructor

Pascal

```
constructor Create(Control: TOleControl);
```

C++

```
__fastcall TEventDispatch(TOleControl Control);
```

Description

This is Create, a member of class TEventDispatch.

63.1.3.3 TEventDispatch Methods

The methods of the TEventDispatch class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6072)	This is _AddRef, a member of class TEventDispatch.
	_Release (see page 6072)	This is _Release, a member of class TEventDispatch.
	GetIDsOfNames (see page 6072)	This is GetIDsOfNames, a member of class TEventDispatch.
	GetTypeInfo (see page 6072)	This is GetTypeInfo, a member of class TEventDispatch.
	GetTypeInfoCount (see page 6073)	This is GetTypeInfoCount, a member of class TEventDispatch.

	Invoke (see page 6073)	This is Invoke, a member of class TEventDispatch.
	QueryInterface (see page 6073)	This is QueryInterface, a member of class TEventDispatch.

63.1.3.3.1 TEventDispatch._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TEventDispatch.

63.1.3.3.2 TEventDispatch._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TEventDispatch.

63.1.3.3.3 TEventDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TEventDispatch.

63.1.3.3.4 TEventDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TEventDispatch.

63.1.3.3.5 TEventDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TEventDispatch.

63.1.3.3.6 TEventDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word
Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TEventDispatch.

63.1.3.3.7 TEventDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TEventDispatch.

63.1.3.4 TEventDispatch Properties

The properties of the TEventDispatch class are listed here.

Protected Properties

	Name	Description
	Control (see page 6073)	This is Control, a member of class TEventDispatch.

63.1.3.4.1 TEventDispatch.Control Property

Pascal

```
property Control: TOleControl;
```

C++

```
__property TOleControl Control;
```

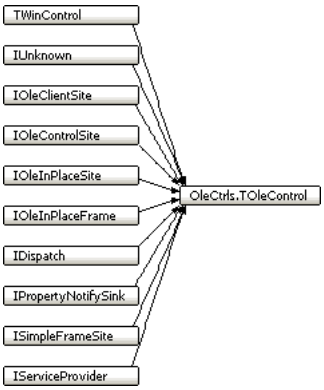
Description

This is Control, a member of class TEventDispatch.

63.1.4 TOleControl Class

TOleControl is the base class from which ActiveX controls are derived.

Class Hierarchy



Pascal

```
TOleControl = class(TWinControl, IUnknown, IOleClientSite, IOleControlSite, IOleInPlaceSite,
IOleInPlaceFrame, IDispatch, IPropertyNotifySink, ISimpleFrameSite, IServiceProvider);
```

C++

```
class TOleControl : public TWinControl, public IUnknown, public IOleClientSite, public
IOleControlSite, public IOleInPlaceSite, public IOleInPlaceFrame, public IDispatch, public
IPropertyNotifySink, public ISimpleFrameSite, public IServiceProvider;
```

File

OleCtrls

Description

TOleControl is derived from TWinControl and handles the interactions with COM necessary for using ActiveX controls. Most application do not need to explicitly derive a component from this class.

See Also

TOleContainer (see page 5969)

Code Generated When You Import Type Library Information

Using Component Wrappers

63.1.4.1 TOleControl Members

The following tables list the members exposed by TOleControl.

Protected Methods

	Name	Description
	_AddRef (see page 6081)	Called when an application uses a component interface.
	_Release (see page 6081)	Called when an application releases a component interface.
	_SetColorProp (see page 6082)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 6082)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 6082)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 6083)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 6083)	This is _SetTPictureProp, a member of class TOleControl.
	CanInPlaceActivate (see page 6083)	This is CanInPlaceActivate, a member of class TOleControl.
	ContextSensitiveHelp (see page 6083)	This is ContextSensitiveHelp, a member of class TOleControl.
V	CreateWnd (see page 6084)	Creates the underlying window.
	DeactivateAndUndo (see page 6084)	This is DeactivateAndUndo, a member of class TOleControl.
V	DefineProperties (see page 6085)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DestroyWindowHandle (see page 6085)	Destroys the window created in the CreateWindowHandle method.
	DiscardUndoState (see page 6086)	This is DiscardUndoState, a member of class TOleControl.
	EnableModeless (see page 6086)	This is EnableModeless, a member of class TOleControl.
	GetBorder (see page 6087)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 6087)	This is GetByteProp, a member of class TOleControl.
	GetColorProp (see page 6087)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 6087)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 6087)	This is GetContainer, a member of class TOleControl.
	GetCurrencyProp (see page 6088)	This is GetCurrencyProp, a member of class TOleControl.
	GetDoubleProp (see page 6088)	This is GetDoubleProp, a member of class TOleControl.
	GetExtendedControl (see page 6088)	This is GetExtendedControl, a member of class TOleControl.
	GetIDispatchProp (see page 6089)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 6089)	This is GetIDsOfNames, a member of class TOleControl.
	GetIntegerProp (see page 6089)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 6090)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 6090)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 6090)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 6091)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 6091)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 6091)	This is GetOleVariantProp, a member of class TOleControl.
	GetProperty (see page 6092)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 6092)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 6092)	This is GetSingleProp, a member of class TOleControl.
	GetSmallintProp (see page 6093)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 6093)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 6093)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 6093)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 6094)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 6094)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTPictureProp (see page 6094)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 6094)	This is GetTypeInfo, a member of class TOleControl.
	GetTypeInfoCount (see page 6094)	This is GetTypeInfoCount, a member of class TOleControl.
	GetVariantProp (see page 6095)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 6095)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 6095)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 6095)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 6096)	This is GetWordProp, a member of class TOleControl.
A	InitControlData (see page 6096)	This is InitControlData, a member of class TOleControl.
V	InitControlInterface (see page 6096)	This is InitControlInterface, a member of class TOleControl.

	InsertMenus (see page 6096)	This is InsertMenus, a member of class TOleControl.
	Invoke (see page 6096)	This is Invoke, a member of class TOleControl.
V	InvokeEvent (see page 6097)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 6097)	This is InvokeMethod, a member of class TOleControl.
	LockInPlaceActive (see page 6098)	This is LockInPlaceActive, a member of class TOleControl.
	OleControlSite_TranslateAccelerator (see page 6098)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 6098)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 6098)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 6099)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 6099)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 6099)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 6099)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 6100)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 6100)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 6100)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 6100)	This is OnRequestEdit, a member of class TOleControl.
	OnShowWindow (see page 6100)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 6101)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 6101)	This is OnUIDeactivate, a member of class TOleControl.
V	PaletteChanged (see page 6101)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PictureChanged (see page 6102)	This is PictureChanged, a member of class TOleControl.
	PostMessageFilter (see page 6102)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 6102)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 6103)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 6103)	This is QueryService, a member of class TOleControl.
	RemoveMenus (see page 6103)	This is RemoveMenus, a member of class TOleControl.
	RequestBorderSpace (see page 6104)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 6104)	This is RequestNewObjectLayout, a member of class TOleControl.
	SaveObject (see page 6104)	This is SaveObject, a member of class TOleControl.
	Scroll (see page 6104)	This is Scroll, a member of class TOleControl.
	SetActiveObject (see page 6104)	This is SetActiveObject, a member of class TOleControl.
	SetBorderSpace (see page 6105)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 6105)	This is SetByteProp, a member of class TOleControl.
	SetColorProp (see page 6105)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 6106)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 6106)	This is SetCurrencyProp, a member of class TOleControl.
	SetDoubleProp (see page 6106)	This is SetDoubleProp, a member of class TOleControl.
	SetIDispatchProp (see page 6106)	This is SetIDispatchProp, a member of class TOleControl.
	SetIntegerProp (see page 6107)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 6107)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 6107)	This is SetMenu, a member of class TOleControl.
V	SetName (see page 6107)	Sets the name of the control.
	SetOleBoolProp (see page 6108)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 6108)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 6108)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 6108)	This is SetOleVariantProp, a member of class TOleControl.
V	SetParent (see page 6108)	Sets the parent of the control.
	SetProperty (see page 6109)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 6110)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 6110)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 6110)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 6110)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 6111)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 6111)	This is SetTColorProp, a member of class TOleControl.

	SetTDateTimeProp (see page 6111)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 6111)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 6111)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 6112)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 6112)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 6112)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 6112)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 6113)	This is SetWordProp, a member of class TOleControl.
	ShowObject (see page 6113)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 6113)	This is ShowPropertyFrame, a member of class TOleControl.
V	StandardEvent (see page 6113)	This is StandardEvent, a member of class TOleControl.
V	SuppressException (see page 6114)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 6114)	This is TransformCoords, a member of class TOleControl.
V	WndProc (see page 6114)	Provides specific message responses for the control.

Protected Properties

	Name	Description
	ControlData (see page 6119)	This is ControlData, a member of class TOleControl.

Public Events

	Name	Description
	ServiceQuery (see page 6121)	This is ServiceQuery, a member of class TOleControl.

Public Methods

	Name	Description
	BrowseProperties (see page 6083)	Notifies the ActiveX control to display its Properties dialog box.
V	Create (see page 6077)	Creates a new instance of TOleControl (see page 6074).
V	DefaultHandler (see page 6084)	Handles all otherwise-unhandled messages.
V	Destroy (see page 6078)	Frees the memory associated with this instance of TOleControl (see page 6074).
	DoObjectVerb (see page 6086)	Notifies the ActiveX control to perform one of its actions.
	GetEnumPropDesc (see page 6088)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 6089)	Retrieves the Help context of a member of the ActiveX control.
	GetObjectVerbs (see page 6090)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetPropDisplayString (see page 6091)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 6092)	Returns the display strings for an interface for the ActiveX control.
	IsCustomProperty (see page 6097)	Indicates whether an interface for the ActiveX control is a custom property.
	IsPropPageProperty (see page 6097)	Indicates whether an interface for the OCX control is a property page property.
V	SetBounds (see page 6105)	Sets the boundaries of the ActiveX control.
	SetPropDisplayString (see page 6109)	Sets the display string for an interface for the ActiveX control.
	ShowAboutBox (see page 6113)	Notifies the ActiveX control to display its About box.

Public Properties

	Name	Description
	Anchors (see page 6119)	Specifies how the control is anchored to its parent.
R	DefaultDispatch (see page 6120)	The default dispatch interface.
R	OleObject (see page 6120)	The ActiveX control associated with the TOleControl (see page 6074) object.
R	PerPropBrowsing (see page 6120)	Provides access to information about the property page properties of the OleControl.
	TabStop (see page 6121)	Determines if the user can tab to a control.

63.1.4.2 TOleControl.Create Constructor

Creates a new instance of TOleControl (see page 6074).

Pascal

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

C++

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

Description

Use Create to create a new instance of the TOleControl (see page 6074) object. Create calls the inherited Create method and creates and initializes internal objects.

See Also

Destroy (see page 6078)

Create

63.1.4.3 TOleControl.Destroy Destructor

Frees the memory associated with this instance of TOleControl (see page 6074).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleControl();
```

Description

Do not call Destroy. Instead, call Free, which calls Destroy if this instance of TOleControl (see page 6074) is not nil.

See Also

Create (see page 6077)

System::TObject::Free

63.1.4.4 TOleControl Methods

The methods of the TOleControl class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6081)	Called when an application uses a component interface.
	_Release (see page 6081)	Called when an application releases a component interface.
	_SetColorProp (see page 6082)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 6082)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 6082)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 6083)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 6083)	This is _SetTPictureProp, a member of class TOleControl.
	CanInPlaceActivate (see page 6083)	This is CanInPlaceActivate, a member of class TOleControl.
	ContextSensitiveHelp (see page 6083)	This is ContextSensitiveHelp, a member of class TOleControl.
V	CreateWnd (see page 6084)	Creates the underlying window.
	DeactivateAndUndo (see page 6084)	This is DeactivateAndUndo, a member of class TOleControl.
V	DefineProperties (see page 6085)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DestroyWindowHandle (see page 6085)	Destroys the window created in the CreateWindowHandle method.

	DiscardUndoState (see page 6086)	This is DiscardUndoState, a member of class TOleControl.
	EnableModeless (see page 6086)	This is EnableModeless, a member of class TOleControl.
	GetBorder (see page 6087)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 6087)	This is GetByteProp, a member of class TOleControl.
	GetColorProp (see page 6087)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 6087)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 6087)	This is GetContainer, a member of class TOleControl.
	GetCurrencyProp (see page 6088)	This is GetCurrencyProp, a member of class TOleControl.
	GetDoubleProp (see page 6088)	This is GetDoubleProp, a member of class TOleControl.
	GetExtendedControl (see page 6088)	This is GetExtendedControl, a member of class TOleControl.
	GetIDispatchProp (see page 6089)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 6089)	This is GetIDsOfNames, a member of class TOleControl.
	GetIntegerProp (see page 6089)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 6090)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 6090)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 6090)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 6091)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 6091)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 6091)	This is GetOleVariantProp, a member of class TOleControl.
	GetProperty (see page 6092)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 6092)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 6092)	This is GetSingleProp, a member of class TOleControl.
	GetSmallintProp (see page 6093)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 6093)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 6093)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 6093)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 6094)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 6094)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTPictureProp (see page 6094)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 6094)	This is GetTypeInfo, a member of class TOleControl.
	GetTypeInfoCount (see page 6094)	This is GetTypeInfoCount, a member of class TOleControl.
	GetVariantProp (see page 6095)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 6095)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 6095)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 6095)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 6096)	This is GetWordProp, a member of class TOleControl.
	InitControlData (see page 6096)	This is InitControlData, a member of class TOleControl.
	InitControlInterface (see page 6096)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 6096)	This is InsertMenus, a member of class TOleControl.
	Invoke (see page 6096)	This is Invoke, a member of class TOleControl.
	InvokeEvent (see page 6097)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 6097)	This is InvokeMethod, a member of class TOleControl.
	LockInPlaceActive (see page 6098)	This is LockInPlaceActive, a member of class TOleControl.
	OleControlSite_TranslateAccelerator (see page 6098)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 6098)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 6098)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 6099)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
	OnChanged (see page 6099)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 6099)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 6099)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 6100)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 6100)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 6100)	This is OnPosRectChange, a member of class TOleControl.
	OnRequestEdit (see page 6100)	This is OnRequestEdit, a member of class TOleControl.

	OnShowWindow (see page 6100)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 6101)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 6101)	This is OnUIDeactivate, a member of class TOleControl.
V	PaletteChanged (see page 6101)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PictureChanged (see page 6102)	This is PictureChanged, a member of class TOleControl.
	PostMessageFilter (see page 6102)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 6102)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 6103)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 6103)	This is QueryService, a member of class TOleControl.
	RemoveMenus (see page 6103)	This is RemoveMenus, a member of class TOleControl.
	RequestBorderSpace (see page 6104)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 6104)	This is RequestNewObjectLayout, a member of class TOleControl.
	SaveObject (see page 6104)	This is SaveObject, a member of class TOleControl.
	Scroll (see page 6104)	This is Scroll, a member of class TOleControl.
	SetActiveObject (see page 6104)	This is SetActiveObject, a member of class TOleControl.
	SetBorderSpace (see page 6105)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 6105)	This is SetByteProp, a member of class TOleControl.
	SetColorProp (see page 6105)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 6106)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 6106)	This is SetCurrencyProp, a member of class TOleControl.
	SetDoubleProp (see page 6106)	This is SetDoubleProp, a member of class TOleControl.
	SetIDispatchProp (see page 6106)	This is SetIDispatchProp, a member of class TOleControl.
	SetIntegerProp (see page 6107)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 6107)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 6107)	This is SetMenu, a member of class TOleControl.
V	SetName (see page 6107)	Sets the name of the control.
	SetOleBoolProp (see page 6108)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 6108)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 6108)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 6108)	This is SetOleVariantProp, a member of class TOleControl.
V	SetParent (see page 6108)	Sets the parent of the control.
	SetProperty (see page 6109)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 6110)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 6110)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 6110)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 6110)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 6111)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 6111)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 6111)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 6111)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 6111)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 6112)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 6112)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 6112)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 6112)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 6113)	This is SetWordProp, a member of class TOleControl.
	ShowObject (see page 6113)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 6113)	This is ShowPropertyFrame, a member of class TOleControl.
V	StandardEvent (see page 6113)	This is StandardEvent, a member of class TOleControl.
V	SuppressException (see page 6114)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 6114)	This is TransformCoords, a member of class TOleControl.
V	WndProc (see page 6114)	Provides specific message responses for the control.

Public Methods

	Name	Description
	BrowseProperties (see page 6083)	Notifies the ActiveX control to display its Properties dialog box.
V	DefaultHandler (see page 6084)	Handles all otherwise-unhandled messages.
	DoObjectVerb (see page 6086)	Notifies the ActiveX control to perform one of its actions.
	GetEnumPropDesc (see page 6088)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 6089)	Retrieves the Help context of a member of the ActiveX control.
	GetObjectVerbs (see page 6090)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetPropDisplayString (see page 6091)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 6092)	Returns the display strings for an interface for the ActiveX control.
	IsCustomProperty (see page 6097)	Indicates whether an interface for the ActiveX control is a custom property.
	IsPropPageProperty (see page 6097)	Indicates whether an interface for the OCX control is a property page property.
V	SetBounds (see page 6105)	Sets the boundaries of the ActiveX control.
	SetPropDisplayString (see page 6109)	Sets the display string for an interface for the ActiveX control.
	ShowAboutBox (see page 6113)	Notifies the ActiveX control to display its About box.

63.1.4.4.1 TOleControl._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke

QueryInterface

_Release

ComObject

_AddRef

63.1.4.4.2 TOleControl._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke

QueryInterface

_AddRef

ComObject

63.1.4.4.3 TOleControl._SetColorProp Method

Pascal

```
procedure _SetColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall _SetColorProp(int Index, TColor Value);
```

Description

This is _SetColorProp, a member of class TOleControl.

63.1.4.4.4 TOleControl._SetTColorProp Method

Pascal

```
procedure _SetTColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall _SetTColorProp(int Index, TColor Value);
```

Description

This is _SetTColorProp, a member of class TOleControl.

63.1.4.4.5 TOleControl._SetTFontProp Method

Pascal

```
procedure _SetTFontProp(Index: Integer; Value: TFont);
```

C++

```
__fastcall _SetTFontProp(int Index, TFont Value);
```

Description

This is _SetTFontProp, a member of class TOleControl.

63.1.4.4.6 TOleControl._SetTOleEnumProp Method

Pascal

```
procedure _SetTOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall _SetTOleEnumProp(int Index, TOleEnum Value);
```

Description

This is _SetTOleEnumProp, a member of class TOleControl.

63.1.4.4.7 TOleControl._SetTPictureProp Method

Pascal

```
procedure _SetTPictureProp(Index: Integer; Value: TPicture);
```

C++

```
__fastcall _SetTPictureProp(int Index, TPicture Value);
```

Description

This is _SetTPictureProp, a member of class TOleControl.

63.1.4.4.8 TOleControl.BrowseProperties Method

Notifies the ActiveX control to display its Properties dialog box.

Pascal

```
procedure BrowseProperties;
```

C++

```
__fastcall BrowseProperties();
```

Description

Use BrowseProperties to notify the ActiveX control to display its Properties dialog box. The Properties dialog box contains property pages for the ActiveX control.

63.1.4.4.9 TOleControl.CanInPlaceActivate Method

Pascal

```
function CanInPlaceActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CanInPlaceActivate();
```

Description

This is CanInPlaceActivate, a member of class TOleControl.

63.1.4.4.10 TOleControl.ContextSensitiveHelp Method

Pascal

```
function ContextSensitiveHelp(fEnterMode: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ContextSensitiveHelp(BOOL fEnterMode);
```

Description

This is ContextSensitiveHelp, a member of class TOleControl.

63.1.4.4.11 TOleControl.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

63.1.4.4.12 TOleControl.DeactivateAndUndo Method

Pascal

```
function DeactivateAndUndo: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DeactivateAndUndo();
```

Description

This is DeactivateAndUndo, a member of class TOleControl.

63.1.4.4.13 TOleControl.DefaultHandler Method

Handles all otherwise-unhandled messages.

Pascal

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

C++

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

Description

DefaultHandler handles all messages not already handled in a control's WndProc (see page 6114) or message handling methods. The inherited method should always be called for unhandled messages.

See Also

DefaultHandler

63.1.4.4.14 TOleControl.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty

DefineProperty

DefineProperties

IsControl

63.1.4.4.15 TOleControl.DestroyWindowHandle Method

Destroys the window created in the CreateWindowHandle method.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

Call DestroyWindowHandle to dispose of the window for the control.

TWinControl's Destroy method calls DestroyWindowHandle to destroy any window associated with a windowed control before destroying the object. The DestroyWnd method also calls DestroyWindowHandle.

See Also

CreateWindowHandle

Destroy

DestroyWnd

63.1.4.4.16 TOleControl.DiscardUndoState Method

Pascal

```
function DiscardUndoState: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DiscardUndoState();
```

Description

This is DiscardUndoState, a member of class TOleControl.

63.1.4.4.17 TOleControl.DoObjectVerb Method

Notifies the ActiveX control to perform one of its actions.

Pascal

```
procedure DoObjectVerb(Verb: Integer);
```

C++

```
__fastcall DoObjectVerb(int Verb);
```

Description

Use DoObjectVerb to notify the ActiveX control to perform one of its actions. Verb is an integer that corresponds to the action. Most application-defined verbs are small positive integers.

Verb Name	Verb
OLEIVERB_PRIMARY	0
OLEIVERB_SHOW	–1
OLEIVERB_OPEN	–2
OLEIVERB_HIDE	–3
OLEIVERB_UIACTIVATE	–4
OLEIVERB_INPLACEACTIVATE	–5
OLEIVERB_DISCARDUNDOSTATE	–6
OLEIVERB_PROPERTIES	–7

For more information about verbs, search the OLE.HLP file or the latest Microsoft Developer Network CD.

63.1.4.4.18 TOleControl.EnableModeless Method

Pascal

```
function EnableModeless(fEnable: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnableModeless(BOOL fEnable);
```

Description

This is EnableModeless, a member of class TOleControl.

63.1.4.4.19 TOleControl.GetBorder Method

Pascal

```
function GetBorder(out rectBorder: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetBorder(TRect rectBorder);
```

Description

This is GetBorder, a member of class TOleControl.

63.1.4.4.20 TOleControl.GetByteProp Method

Pascal

```
function GetByteProp(Index: Integer): Byte;
```

C++

```
__fastcall Byte GetByteProp(int Index);
```

Description

This is GetByteProp, a member of class TOleControl.

63.1.4.4.21 TOleControl.GetColorProp Method

Pascal

```
function GetColorProp(Index: Integer): TColor;
```

C++

```
__fastcall TColor GetColorProp(int Index);
```

Description

This is GetColorProp, a member of class TOleControl.

63.1.4.4.22 TOleControl.GetCompProp Method

Pascal

```
function GetCompProp(Index: Integer): Comp;
```

C++

```
__fastcall Comp GetCompProp(int Index);
```

Description

This is GetCompProp, a member of class TOleControl.

63.1.4.4.23 TOleControl.GetContainer Method

Pascal

```
function GetContainer(out container: IOleContainer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetContainer(IOleContainer container);
```

Description

This is GetContainer, a member of class TOleControl.

63.1.4.4.24 TOleControl.GetCurrencyProp Method

Pascal

```
function GetCurrencyProp(Index: Integer): Currency;
```

C++

```
__fastcall Currency GetCurrencyProp(int Index);
```

Description

This is GetCurrencyProp, a member of class TOleControl.

63.1.4.4.25 TOleControl.GetDoubleProp Method

Pascal

```
function GetDoubleProp(Index: Integer): Double;
```

C++

```
__fastcall Double GetDoubleProp(int Index);
```

Description

This is GetDoubleProp, a member of class TOleControl.

63.1.4.4.26 TOleControl.GetEnumPropDesc Method

Returns descriptions of the ActiveX control's properties.

Pascal

```
function GetEnumPropDesc(DispID: Integer): TEnumPropDesc;
```

C++

```
__fastcall TEnumPropDesc GetEnumPropDesc(int DispID);
```

Description

Use GetEnumPropDesc to return descriptions of the ActiveX control's properties. GetEnumPropDesc returns the descriptions in a TEnumPropDesc (see page 6067) object.

63.1.4.4.27 TOleControl.GetExtendedControl Method

Pascal

```
function GetExtendedControl(out disp: IDispatch): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetExtendedControl(IDispatch disp);
```

Description

This is GetExtendedControl, a member of class TOleControl.

63.1.4.4.28 TOleControl.GetHelpContext Method

Retrieves the Help context of a member of the ActiveX control.

Pascal

```
function GetHelpContext(Member: string; var HelpCtx: Integer; var HelpFile: string): Boolean;
```

C++

```
__fastcall Boolean GetHelpContext(UnicodeString Member, int HelpCtx, UnicodeString HelpFile);
```

Description

Use GetHelpContext to retrieve the Help context of a member of the ActiveX control.

Member is the name of a property, method, or event of the ActiveX control. To retrieve the help context of the ActiveX control itself, set Member to nil (Delphi) or NULL (C++).

HelpCtx is a variable in which the Help context identifier of Member is to be stored.

HelpFile is a variable in which the name of the Help file containing HelpCtx is to be stored.

The information about the ActiveX control is retrieved from its ITypeInfo interfaces. For more information, search the OLE.HLP file for ITypeInfo.

GetHelpContext returns false if Member is not found; otherwise, GetHelpContext returns true.

63.1.4.4.29 TOleControl.GetIDispatchProp Method

Pascal

```
function GetIDispatchProp(Index: Integer): IDispatch;
```

C++

```
__fastcall IDispatch GetIDispatchProp(int Index);
```

Description

This is GetIDispatchProp, a member of class TOleControl.

63.1.4.4.30 TOleControl.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TOleControl.

63.1.4.4.31 TOleControl.GetIntegerProp Method

Pascal

```
function GetIntegerProp(Index: Integer): Integer;
```

C++

```
__fastcall int GetIntegerProp(int Index);
```

Description

This is GetIntegerProp, a member of class TOleControl.

63.1.4.4.32 TOleControl.GetIUnknownProp Method

Pascal

```
function GetIUnknownProp(Index: Integer): IUnknown;
```

C++

```
__fastcall IUnknown GetIUnknownProp(int Index);
```

Description

This is GetIUnknownProp, a member of class TOleControl.

63.1.4.4.33 TOleControl.GetMoniker Method

Pascal

```
function GetMoniker(dwAssign: Longint; dwWhichMoniker: Longint; out mk: IMoniker): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetMoniker(Longint dwAssign, Longint dwWhichMoniker, IMoniker mk);
```

Description

This is GetMoniker, a member of class TOleControl.

63.1.4.4.34 TOleControl.GetObjectVerbs Method

Retrieves the names of the verbs implemented by the ActiveX control.

Pascal

```
procedure GetObjectVerbs(List: TStrings);
```

C++

```
__fastcall GetObjectVerbs(TStrings List);
```

Description

Use GetObjectVerbs to retrieve the names of the verbs implemented by the ActiveX control.

List is where the names of the verbs are to be stored.

63.1.4.4.35 TOleControl.GetOleBoolProp Method

Pascal

```
function GetOleBoolProp(Index: Integer): TOleBool;
```

C++

```
__fastcall TOleBool GetOleBoolProp(int Index);
```

Description

This is GetOleBoolProp, a member of class TOleControl.

63.1.4.4.36 TOleControl.GetOleDateProp Method

Pascal

```
function GetOleDateProp(Index: Integer): TOleDate;
```

C++

```
__fastcall TOleDate GetOleDateProp(int Index);
```

Description

This is GetOleDateProp, a member of class TOleControl.

63.1.4.4.37 TOleControl.GetOleEnumProp Method

Pascal

```
function GetOleEnumProp(Index: Integer): TOleEnum;
```

C++

```
__fastcall TOleEnum GetOleEnumProp(int Index);
```

Description

This is GetOleEnumProp, a member of class TOleControl.

63.1.4.4.38 TOleControl.GetOleVariantProp Method

Pascal

```
function GetOleVariantProp(Index: Integer): OleVariant;
```

C++

```
__fastcall OleVariant GetOleVariantProp(int Index);
```

Description

This is GetOleVariantProp, a member of class TOleControl.

63.1.4.4.39 TOleControl.GetPropDisplayString Method

Returns the display string for an interface for the ActiveX control.

Pascal

```
function GetPropDisplayString(DispID: Integer): string;
```

C++

```
__fastcall UnicodeString GetPropDisplayString(int DispID);
```

Description

GetPropDisplayString returns the display string for an interface for the ActiveX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

For more information about GetPropDisplayString, search the latest Microsoft Developer Network CD for GetDisplayStrings.

See Also

GetPropDisplayStrings (see page 6092)

SetPropDisplayString (see page 6109)

63.1.4.4.40 TOleControl.GetPropDisplayStrings Method

Returns the display strings for an interface for the ActiveX control.

Pascal

```
procedure GetPropDisplayStrings(DispID: Integer; List: TStrings);
```

C++

```
__fastcall GetPropDisplayStrings(int DispID, TStrings List);
```

Description

List is where the strings are to be stored.

For more information about GetPropDisplayStrings, search the latest Microsoft Developer Network CD for GetDisplayStrings.

See Also

GetPropDisplayString (see page 6091)

63.1.4.4.41 TOleControl.GetProperty Method

Pascal

```
procedure GetProperty(Index: Integer; var Value: TVarData);
```

C++

```
__fastcall GetProperty(int Index, TVarData Value);
```

Description

This is GetProperty, a member of class TOleControl.

63.1.4.4.42 TOleControl.GetShortIntProp Method

Pascal

```
function GetShortIntProp(Index: Integer): ShortInt;
```

C++

```
__fastcall ShortInt GetShortIntProp(int Index);
```

Description

This is GetShortIntProp, a member of class TOleControl.

63.1.4.4.43 TOleControl.GetSingleProp Method

Pascal

```
function GetSingleProp(Index: Integer): Single;
```

C++

```
__fastcall Single GetSingleProp(int Index);
```

Description

This is GetSingleProp, a member of class TOleControl.

63.1.4.4.44 TOleControl.GetSmallintProp Method

Pascal

```
function GetSmallintProp(Index: Integer): Smallint;
```

C++

```
__fastcall Smallint GetSmallintProp(int Index);
```

Description

This is GetSmallintProp, a member of class TOleControl.

63.1.4.4.45 TOleControl.GetStringProp Method

Pascal

```
function GetStringProp(Index: Integer): string;
```

C++

```
__fastcall UnicodeString GetStringProp(int Index);
```

Description

This is GetStringProp, a member of class TOleControl.

63.1.4.4.46 TOleControl.GetTColorProp Method

Pascal

```
function GetTColorProp(Index: Integer): TColor;
```

C++

```
__fastcall TColor GetTColorProp(int Index);
```

Description

This is GetTColorProp, a member of class TOleControl.

63.1.4.4.47 TOleControl.GetTDateTimeProp Method

Pascal

```
function GetTDateTimeProp(Index: Integer): TDateTime;
```

C++

```
__fastcall TDateTime GetTDateTimeProp(int Index);
```

Description

This is GetTDateTimeProp, a member of class TOleControl.

63.1.4.4.48 TOleControl.GetTFontProp Method

Pascal

```
function GetTFontProp(Index: Integer): TFont;
```

C++

```
__fastcall TFont GetTFontProp(int Index);
```

Description

This is GetTFontProp, a member of class TOleControl.

63.1.4.4.49 TOleControl.GetTOleEnumProp Method

Pascal

```
function GetTOleEnumProp(Index: Integer): TOleEnum;
```

C++

```
__fastcall TOleEnum GetTOleEnumProp(int Index);
```

Description

This is GetTOleEnumProp, a member of class TOleControl.

63.1.4.4.50 TOleControl.GetTPictureProp Method

Pascal

```
function GetTPictureProp(Index: Integer): TPicture;
```

C++

```
__fastcall TPicture GetTPictureProp(int Index);
```

Description

This is GetTPictureProp, a member of class TOleControl.

63.1.4.4.51 TOleControl.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TOleControl.

63.1.4.4.52 TOleControl.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TOleControl.

63.1.4.4.53 TOleControl.GetVariantProp Method

Pascal

```
function GetVariantProp(Index: Integer): Variant;
```

C++

```
__fastcall Variant GetVariantProp(int Index);
```

Description

This is GetVariantProp, a member of class TOleControl.

63.1.4.4.54 TOleControl.GetWideStringProp Method

Pascal

```
function GetWideStringProp(Index: Integer): WideString;
```

C++

```
__fastcall BSTR GetWideStringProp(int Index);
```

Description

This is GetWideStringProp, a member of class TOleControl.

63.1.4.4.55 TOleControl.GetWindowContext Method

Pascal

```
function GetWindowContext(out frame: IOleInPlaceFrame; out doc: IOleInPlaceUIWindow; out
rcPosRect: TRect; out rcClipRect: TRect; out frameInfo: TOleInPlaceFrameInfo): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetWindowContext(IOleInPlaceFrame frame, IOleInPlaceUIWindow doc,
TRect rcPosRect, TRect rcClipRect, TOleInPlaceFrameInfo frameInfo);
```

Description

This is GetWindowContext, a member of class TOleControl.

63.1.4.4.56 TOleControl.GetWordBoolProp Method

Pascal

```
function GetWordBoolProp(Index: Integer): WordBool;
```

C++

```
__fastcall WordBool GetWordBoolProp(int Index);
```

Description

This is GetWordBoolProp, a member of class TOleControl.

63.1.4.4.57 TOleControl.GetWordProp Method

Pascal

```
function GetWordProp(Index: Integer): Word;
```

C++

```
__fastcall Word GetWordProp(int Index);
```

Description

This is GetWordProp, a member of class TOleControl.

63.1.4.4.58 TOleControl.InitControlData Method

Pascal

```
procedure InitControlData; virtual; abstract;
```

C++

```
virtual __fastcall InitControlData() = 0;
```

Description

This is InitControlData, a member of class TOleControl.

63.1.4.4.59 TOleControl.InitControlInterface Method

Pascal

```
procedure InitControlInterface(const Obj: IUnknown); virtual;
```

C++

```
virtual __fastcall InitControlInterface(const IUnknown Obj);
```

Description

This is InitControlInterface, a member of class TOleControl.

63.1.4.4.60 TOleControl.InsertMenus Method

Pascal

```
function InsertMenus(hmenuShared: HMenu; var menuWidths: TOleMenuGroupWidths): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT InsertMenus(HMenu hmenuShared, TOleMenuGroupWidths menuWidths);
```

Description

This is InsertMenus, a member of class TOleControl.

63.1.4.4.61 TOleControl.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TOleControl.

63.1.4.4.62 TOleControl.InvokeEvent Method

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TDispParams); virtual;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TDispParams Params);
```

Description

This is InvokeEvent, a member of class TOleControl.

63.1.4.4.63 TOleControl.InvokeMethod Method

Pascal

```
procedure InvokeMethod(const DispInfo; Result: Pointer);
```

C++

```
__fastcall InvokeMethod(const  DispInfo, void * Result);
```

Description

This is InvokeMethod, a member of class TOleControl.

63.1.4.4.64 TOleControl.IsCustomProperty Method

Indicates whether an interface for the ActiveX control is a custom property.

Pascal

```
function IsCustomProperty(DispID: Integer): Boolean;
```

C++

```
__fastcall Boolean IsCustomProperty(int DispID);
```

Description

Use IsCustomProperty to determine whether an interface for the ActiveX control is a custom property.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

IsCustomProperty returns true if an interface is a custom property; otherwise, IsCustomProperty returns false.

63.1.4.4.65 TOleControl.IsPropPageProperty Method

Indicates whether an interface for the OCX control is a property page property.

Pascal

```
function IsPropPageProperty(DispID: Integer): Boolean;
```

C++

```
__fastcall Boolean IsPropPageProperty(int DispID);
```

Description

Use IsPropPageProperty to determine whether an interface for the OCX control is a property page property.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

IsPropPageProperty returns true if an interface is a property page property; otherwise, IsPropPageProperty returns false.

63.1.4.4.66 TOleControl.LockInPlaceActive Method

Pascal

```
function LockInPlaceActive(fLock: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT LockInPlaceActive(BOOL fLock);
```

Description

This is LockInPlaceActive, a member of class TOleControl.

63.1.4.4.67 TOleControl.OleControlSite_TranslateAccelerator Method

Pascal

```
function OleControlSite_TranslateAccelerator(msg: PMsg; grfModifiers: Longint): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleControlSite_TranslateAccelerator(PMsg msg, Longint
grfModifiers);
```

Description

This is OleControlSite_TranslateAccelerator, a member of class TOleControl.

63.1.4.4.68 TOleControl.OleInPlaceFrame_GetWindow Method

Pascal

```
function OleInPlaceFrame_GetWindow(out wnd: HWnd): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceFrame_GetWindow(HWnd wnd);
```

Description

This is OleInPlaceFrame_GetWindow, a member of class TOleControl.

63.1.4.4.69 TOleControl.OleInPlaceFrame_TranslateAccelerator Method

Pascal

```
function OleInPlaceFrame_TranslateAccelerator(var msg: TMsg; wID: Word): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceFrame_TranslateAccelerator(TMsg msg, Word wID);
```

Description

This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.

63.1.4.4.70 TOleControl.OleInPlaceSite_GetWindow Method

Pascal

```
function OleInPlaceSite_GetWindow(out wnd: HWnd): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceSite_GetWindow(HWnd wnd);
```

Description

This is OleInPlaceSite_GetWindow, a member of class TOleControl.

63.1.4.4.71 TOleControl.OnChanged Method

Pascal

```
function OnChanged(dispid: TDispID): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT OnChanged(TDispID dispid);
```

Description

This is OnChanged, a member of class TOleControl.

63.1.4.4.72 TOleControl.OnControlInfoChanged Method

Pascal

```
function OnControlInfoChanged: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnControlInfoChanged();
```

Description

This is OnControlInfoChanged, a member of class TOleControl.

63.1.4.4.73 TOleControl.OnFocus Method

Pascal

```
function OnFocus(fGotFocus: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnFocus(BOOL fGotFocus);
```

Description

This is OnFocus, a member of class TOleControl.

63.1.4.4.74 TOleControl.OnInPlaceActivate Method

Pascal

```
function OnInPlaceActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnInPlaceActivate();
```

Description

This is OnInPlaceActivate, a member of class TOleControl.

63.1.4.4.75 TOleControl.OnInPlaceDeactivate Method

Pascal

```
function OnInPlaceDeactivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnInPlaceDeactivate();
```

Description

This is OnInPlaceDeactivate, a member of class TOleControl.

63.1.4.4.76 TOleControl.OnPosRectChange Method

Pascal

```
function OnPosRectChange(const rcPosRect: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnPosRectChange(const TRect rcPosRect);
```

Description

This is OnPosRectChange, a member of class TOleControl.

63.1.4.4.77 TOleControl.OnRequestEdit Method

Pascal

```
function OnRequestEdit(dispid: TDispID): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT OnRequestEdit(TDispID dispid);
```

Description

This is OnRequestEdit, a member of class TOleControl.

63.1.4.4.78 TOleControl.OnShowWindow Method

Pascal

```
function OnShowWindow(fShow: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnShowWindow(BOOL fShow);
```

Description

This is OnShowWindow, a member of class TOleControl.

63.1.4.4.79 TOleControl.OnUIActivate Method

Pascal

```
function OnUIActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnUIActivate();
```

Description

This is OnUIActivate, a member of class TOleControl.

63.1.4.4.80 TOleControl.OnUIDeactivate Method

Pascal

```
function OnUIDeactivate(fUndoable: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnUIDeactivate(BOOL fUndoable);
```

Description

This is OnUIDeactivate, a member of class TOleControl.

63.1.4.4.81 TOleControl.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

PaletteChanged is called automatically when the Windows system palette changes. TWinControl responds to these notifications by trying to realize the windowed control's palette, if any, into the current device context, and then passing on the PaletteChanged notification to each of the child controls in turn. If the change in the system palette causes the actual value of the control's palette to change, PaletteChanged will invalidate the control so that it can repaint with the new palette.

Windows paints the active window with a foreground palette, while other windows are forced to use background palettes. Background palettes are approximate matches to the colors specified by the control's logical palette, given the limitations imposed by implementing the foreground palette. Windows only allows a single foreground palette.

PaletteChanged allows the control to obtain a new realization of its palette when the system palette changes. When Foreground is true, the form has been activated and the control is specifying the new foreground palette. When Foreground is false, another application or form has changed the foreground palette, and controls that are sensitive to the available palette should realize new background palettes to best match their logical palettes.

If the control does not have a logical palette to be realized into the current device context, GetPalette will return a handle of 0, and PaletteChanged will not try to realize a palette for the windowed control. Instead, it simply passes the notification on to the child controls. If the windowed control or any of its child controls realize a palette into the current device context in response to

PaletteChanged, this method returns true.

Override PaletteChanged to change the way in which controls are given an opportunity to realize a new palette when the Windows system palette changes. For example, override PaletteChanged to change the order in which child controls realize their palettes, giving a particular control the foreground palette when the form is activated, or to pass palette change notification to custom controls implemented outside of the VCL, such as in DLLs.

Override GetPalette instead to affect whether the windowed control actually has a palette to realize, or to change the value of the control's logical palette.

Note: PaletteChanged is called only when the run-time video mode requires palette support, such as for 256 color mode, but not 16 million color mode.

See Also

GetDeviceContext

GetPalette

63.1.4.4.82 TOleControl.PictureChanged Method

Pascal

```
procedure PictureChanged(Sender: TObject);
```

C++

```
__fastcall PictureChanged(TObject * Sender);
```

Description

This is PictureChanged, a member of class TOleControl.

63.1.4.4.83 TOleControl.PostMessageFilter Method

Pascal

```
function PostMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res:
Integer; Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PostMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PostMessageFilter, a member of class TOleControl.

63.1.4.4.84 TOleControl.PreMessageFilter Method

Pascal

```
function PreMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res: Integer;
out Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PreMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PreMessageFilter, a member of class TOleControl.

63.1.4.4.85 TOleControl.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the component supports the interface specified by IID, and if so, returns a reference to that interface as the Obj parameter. If the component does not support the interface, the Obj parameter returns nil (Delphi) or NULL (C++).

Note: For components that act as COM object wrapper, QueryInterface calls the QueryInterface method of the internal COM object.

See Also

_AddRef

_Release

QueryInterface

QueryInterface

Invoke

ComObject

63.1.4.4.86 TOleControl.QueryService Method

Pascal

```
function QueryService(const rsid: TGuid; const iid: TGuid; out obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryService(const TGuid rsid, const TGuid iid,  obj);
```

Description

This is QueryService, a member of class TOleControl.

63.1.4.4.87 TOleControl.RemoveMenus Method

Pascal

```
function RemoveMenus(hmenuShared: HMenu): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RemoveMenus(HMenu hmenuShared);
```

Description

This is RemoveMenus, a member of class TOleControl.

63.1.4.4.88 TOleControl.RequestBorderSpace Method

Pascal

```
function RequestBorderSpace(const borderwidths: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RequestBorderSpace(const TRect borderwidths);
```

Description

This is RequestBorderSpace, a member of class TOleControl.

63.1.4.4.89 TOleControl.RequestNewObjectLayout Method

Pascal

```
function RequestNewObjectLayout: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RequestNewObjectLayout();
```

Description

This is RequestNewObjectLayout, a member of class TOleControl.

63.1.4.4.90 TOleControl.SaveObject Method

Pascal

```
function SaveObject: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SaveObject();
```

Description

This is SaveObject, a member of class TOleControl.

63.1.4.4.91 TOleControl.Scroll Method

Pascal

```
function Scroll(scrollExtent: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Scroll(TPoint scrollExtent);
```

Description

This is Scroll, a member of class TOleControl.

63.1.4.4.92 TOleControl.SetActiveObject Method

Pascal

```
function SetActiveObject(const activeObject: IOleInPlaceActiveObject; pszObjName: POleStr):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetActiveObject(const IOleInPlaceActiveObject activeObject,
POleStr pszObjName);
```

Description

This is SetActiveObject, a member of class TOleControl.

63.1.4.4.93 TOleControl.SetBorderSpace Method

Pascal

```
function SetBorderSpace(pborderwidths: PRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetBorderSpace(PRect pborderwidths);
```

Description

This is SetBorderSpace, a member of class TOleControl.

63.1.4.4.94 TOleControl.SetBounds Method

Sets the boundaries of the ActiveX control.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to set the boundaries of the ActiveX control.

ALeft, ATop, AWidth, and AHeight are the x-coordinate of the upper-left corner, the y-coordinate of the upper-left corner, the width and the height of the ActiveX control.

63.1.4.4.95 TOleControl.SetByteProp Method

Pascal

```
procedure SetByteProp(Index: Integer; Value: Byte);
```

C++

```
__fastcall SetByteProp(int Index, Byte Value);
```

Description

This is SetByteProp, a member of class TOleControl.

63.1.4.4.96 TOleControl.SetColorProp Method

Pascal

```
procedure SetColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall SetColorProp(int Index, TColor Value);
```

Description

This is SetColorProp, a member of class TOleControl.

63.1.4.4.97 TOleControl.SetCompProp Method

Pascal

```
procedure SetCompProp(Index: Integer; const Value: Comp);
```

C++

```
__fastcall SetCompProp(int Index, const Comp Value);
```

Description

This is SetCompProp, a member of class TOleControl.

63.1.4.4.98 TOleControl.SetCurrencyProp Method

Pascal

```
procedure SetCurrencyProp(Index: Integer; const Value: Currency);
```

C++

```
__fastcall SetCurrencyProp(int Index, const Currency Value);
```

Description

This is SetCurrencyProp, a member of class TOleControl.

63.1.4.4.99 TOleControl.SetDoubleProp Method

Pascal

```
procedure SetDoubleProp(Index: Integer; const Value: Double);
```

C++

```
__fastcall SetDoubleProp(int Index, const Double Value);
```

Description

This is SetDoubleProp, a member of class TOleControl.

63.1.4.4.100 TOleControl.SetIDispatchProp Method

Pascal

```
procedure SetIDispatchProp(Index: Integer; const Value: IDispatch);
```

C++

```
__fastcall SetIDispatchProp(int Index, const IDispatch Value);
```

Description

This is SetIDispatchProp, a member of class TOleControl.

63.1.4.4.101 TOleControl.SetIntegerProp Method

Pascal

```
procedure SetIntegerProp(Index: Integer; Value: Integer);
```

C++

```
__fastcall SetIntegerProp(int Index, int Value);
```

Description

This is SetIntegerProp, a member of class TOleControl.

63.1.4.4.102 TOleControl.SetIUnknownProp Method

Pascal

```
procedure SetIUnknownProp(Index: Integer; const Value: IUnknown);
```

C++

```
__fastcall SetIUnknownProp(int Index, const IUnknown Value);
```

Description

This is SetIUnknownProp, a member of class TOleControl.

63.1.4.4.103 TOleControl.SetMenu Method

Pascal

```
function SetMenu(hmenuShared: HMenu; holemenu: HMenu; hwndActiveObject: HWnd): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetMenu(HMenu hmenuShared, HMenu holemenu, HWnd hwndActiveObject);
```

Description

This is SetMenu, a member of class TOleControl.

63.1.4.4.104 TOleControl.SetName Method

Sets the name of the control.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. TControl updates the Text or Caption property, if appropriate, after changing the value of Name.

Override SetName to execute additional code when the value of the Name property changes.

See Also

Name

Caption

Text

63.1.4.4.105 TOleControl.SetOleBoolProp Method

Pascal

```
procedure SetOleBoolProp(Index: Integer; Value: TOleBool);
```

C++

```
__fastcall SetOleBoolProp(int Index, TOleBool Value);
```

Description

This is SetOleBoolProp, a member of class TOleControl.

63.1.4.4.106 TOleControl.SetOleDateProp Method

Pascal

```
procedure SetOleDateProp(Index: Integer; const Value: TOleDate);
```

C++

```
__fastcall SetOleDateProp(int Index, const TOleDate Value);
```

Description

This is SetOleDateProp, a member of class TOleControl.

63.1.4.4.107 TOleControl.SetOleEnumProp Method

Pascal

```
procedure SetOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall SetOleEnumProp(int Index, TOleEnum Value);
```

Description

This is SetOleEnumProp, a member of class TOleControl.

63.1.4.4.108 TOleControl.SetOleVariantProp Method

Pascal

```
procedure SetOleVariantProp(Index: Integer; const Value: OleVariant);
```

C++

```
__fastcall SetOleVariantProp(int Index, const OleVariant Value);
```

Description

This is SetOleVariantProp, a member of class TOleControl.

63.1.4.4.109 TOleControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

63.1.4.4.110 TOleControl.SetPropDisplayString Method

Sets the display string for an interface for the ActiveX control.

Pascal

```
procedure SetPropDisplayString(DispID: Integer; const Value: string);
```

C++

```
__fastcall SetPropDisplayString(int DispID, const UnicodeString Value);
```

Description

Use SetPropDisplayString to set the display string for an interface for the ActiveX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

Value is the new display string for the interface.

See Also

GetPropDisplayStrings (see page 6092)

SetPropDisplayString

63.1.4.4.111 TOleControl.SetProperty Method

Pascal

```
procedure SetProperty(Index: Integer; const Value: TVarData);
```

C++

```
__fastcall SetProperty(int Index, const TVarData Value);
```

Description

This is SetProperty, a member of class TOleControl.

63.1.4.4.112 TOleControl.SetShortIntProp Method

Pascal

```
procedure SetShortIntProp(Index: Integer; Value: Shortint);
```

C++

```
__fastcall SetShortIntProp(int Index, Shortint Value);
```

Description

This is SetShortIntProp, a member of class TOleControl.

63.1.4.4.113 TOleControl.SetSingleProp Method

Pascal

```
procedure SetSingleProp(Index: Integer; const Value: Single);
```

C++

```
__fastcall SetSingleProp(int Index, const Single Value);
```

Description

This is SetSingleProp, a member of class TOleControl.

63.1.4.4.114 TOleControl.SetSmallintProp Method

Pascal

```
procedure SetSmallintProp(Index: Integer; Value: Smallint);
```

C++

```
__fastcall SetSmallintProp(int Index, Smallint Value);
```

Description

This is SetSmallintProp, a member of class TOleControl.

63.1.4.4.115 TOleControl.SetStatusText Method

Pascal

```
function SetStatusText(pszStatusText: POleStr): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetStatusText(POleStr pszStatusText);
```

Description

This is SetStatusText, a member of class TOleControl.

63.1.4.4.116 TOleControl.SetStringProp Method

Pascal

```
procedure SetStringProp(Index: Integer; const Value: string);
```

C++

```
__fastcall SetStringProp(int Index, const UnicodeString Value);
```

Description

This is SetStringProp, a member of class TOleControl.

63.1.4.4.117 TOleControl.SetTColorProp Method

Pascal

```
procedure SetTColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall SetTColorProp(int Index, TColor Value);
```

Description

This is SetTColorProp, a member of class TOleControl.

63.1.4.4.118 TOleControl.SetTDateTimeProp Method

Pascal

```
procedure SetTDateTimeProp(Index: Integer; const Value: TDateTime);
```

C++

```
__fastcall SetTDateTimeProp(int Index, const TDateTime Value);
```

Description

This is SetTDateTimeProp, a member of class TOleControl.

63.1.4.4.119 TOleControl.SetTFontProp Method

Pascal

```
procedure SetTFontProp(Index: Integer; Value: TFont);
```

C++

```
__fastcall SetTFontProp(int Index, TFont Value);
```

Description

This is SetTFontProp, a member of class TOleControl.

63.1.4.4.120 TOleControl.SetTOleEnumProp Method

Pascal

```
procedure SetTOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall SetTOleEnumProp(int Index, TOleEnum Value);
```

Description

This is SetTOleEnumProp, a member of class TOleControl.

63.1.4.4.121 TOleControl.SetTPictureProp Method

Pascal

```
procedure SetTPictureProp(Index: Integer; Value: TPicture);
```

C++

```
__fastcall SetTPictureProp(int Index, TPicture Value);
```

Description

This is SetTPictureProp, a member of class TOleControl.

63.1.4.4.122 TOleControl.SetVariantProp Method

Pascal

```
procedure SetVariantProp(Index: Integer; const Value: Variant);
```

C++

```
__fastcall SetVariantProp(int Index, const Variant Value);
```

Description

This is SetVariantProp, a member of class TOleControl.

63.1.4.4.123 TOleControl.SetWideStringProp Method

Pascal

```
procedure SetWideStringProp(Index: Integer; const Value: WideString);
```

C++

```
__fastcall SetWideStringProp(int Index, const BSTR Value);
```

Description

This is SetWideStringProp, a member of class TOleControl.

63.1.4.4.124 TOleControl.SetWordBoolProp Method

Pascal

```
procedure SetWordBoolProp(Index: Integer; Value: WordBool);
```

C++

```
__fastcall SetWordBoolProp(int Index, WordBool Value);
```

Description

This is SetWordBoolProp, a member of class TOleControl.

63.1.4.4.125 TOleControl.SetWordProp Method

Pascal

```
procedure SetWordProp(Index: Integer; Value: Word);
```

C++

```
__fastcall SetWordProp(int Index, Word Value);
```

Description

This is SetWordProp, a member of class TOleControl.

63.1.4.4.126 TOleControl.ShowAboutBox Method

Notifies the ActiveX control to display its About box.

Pascal

```
procedure ShowAboutBox;
```

C++

```
__fastcall ShowAboutBox();
```

Description

Use ShowAboutBox to notify the ActiveX control to display its About box.

63.1.4.4.127 TOleControl.ShowObject Method

Pascal

```
function ShowObject: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ShowObject();
```

Description

This is ShowObject, a member of class TOleControl.

63.1.4.4.128 TOleControl.ShowPropertyFrame Method

Pascal

```
function ShowPropertyFrame: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ShowPropertyFrame();
```

Description

This is ShowPropertyFrame, a member of class TOleControl.

63.1.4.4.129 TOleControl.StandardEvent Method

Pascal

```
procedure StandardEvent(DispID: TDispID; var Params: TDispParams); virtual;
```

C++

```
virtual __fastcall StandardEvent(TDispID DispID, TDispParams Params);
```

Description

This is StandardEvent, a member of class TOleControl.

63.1.4.4.130 TOleControl.SuppressException Method

Pascal

```
function SuppressException(E: Exception): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SuppressException(Exception E);
```

Description

This is SuppressException, a member of class TOleControl.

63.1.4.4.131 TOleControl.TransformCoords Method

Pascal

```
function TransformCoords(var ptlHimetric: TPoint; var ptfContainer: TPointF; flags: Longint):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT TransformCoords(TPoint ptlHimetric, TPointF ptfContainer, Longint
flags);
```

Description

This is TransformCoords, a member of class TOleControl.

63.1.4.4.132 TOleControl.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
```

```
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
```

```
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
```

```
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
```

```
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc

MainWndProc

WndProc

WindowProc

63.1.4.5 TOleControl Properties

The properties of the TOleControl class are listed here.

Protected Properties

	Name	Description
	ControlData (see page 6119)	This is ControlData, a member of class TOleControl.

Public Properties

	Name	Description
	Anchors (see page 6119)	Specifies how the control is anchored to its parent.
R	DefaultDispatch (see page 6120)	The default dispatch interface.
R	OleObject (see page 6120)	The ActiveX control associated with the TOleControl (see page 6074) object.
R	PerPropBrowsing (see page 6120)	Provides access to information about the property page properties of the OleControl.
	TabStop (see page 6121)	Determines if the user can tab to a control.

63.1.4.5.1 TOleControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

63.1.4.5.2 TOleControl.ControlData Property

Pascal

```
property ControlData: PControlData;
```

C++

```
__property PControlData ControlData;
```

Description

This is ControlData, a member of class TOleControl.

63.1.4.5.3 TOleControl.DefaultDispatch Property

The default dispatch interface.

Pascal

```
property DefaultDispatch: IDispatch;
```

C++

```
__property IDispatch DefaultDispatch;
```

Description

DefaultDispatch default dispatch interface that the TOleControl (see page 6074) object encapsulates. For those ActiveX controls that have multiple dispatch interfaces, one is typically identified as the default.

63.1.4.5.4 TOleControl.OleObject Property

The ActiveX control associated with the TOleControl (see page 6074) object.

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

OleObject is used to get the ActiveX control that is associated with the TOleControl (see page 6074)

63.1.4.5.5 TOleControl.PerPropBrowsing Property

Provides access to information about the property page properties of the OleControl.

Pascal

```
property PerPropBrowsing: IPerPropertyBrowsing;
```

C++

```
__property IPerPropertyBrowsing PerPropBrowsing;
```

Description

PerPropBrowsing is an interface to information about the properties of the OleControl as they appear on the associated property page.

See Also

BrowseProperties (see page 6083)

63.1.4.5.6 TOleControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

63.1.4.6 TOleControl Events

The events of the TOleControl class are listed here.

Public Events

	Name	Description
	ServiceQuery (see page 6121)	This is ServiceQuery, a member of class TOleControl.

63.1.4.6.1 TOleControl.ServiceQuery Event

Pascal

```
property ServiceQuery: TServiceQuery;
```

C++

```
__property TServiceQuery ServiceQuery;
```

Description

This is ServiceQuery, a member of class TOleControl.

63.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	EmptyParam (see page 6122)	This is function OleCtrls.EmptyParam.
	FontToOleFont (see page 6122)	Returns a Variant containing an IFontDispatch interface that represents a TFont (see page 2730) object.
	OleFontToFont (see page 6123)	Fills in a TFont (see page 2730) structure to represent an IFontDispatch.

63.2.1 OleCtrls.EmptyParam Function

Pascal

```
function EmptyParam: OleVariant;
```

C++

```
OleVariant EmptyParam();
```

File

OleCtrls

Description

This is function OleCtrls.EmptyParam.

63.2.2 OleCtrls.FontToOleFont Function

Returns a Variant containing an IFontDispatch interface that represents a TFont (see page 2730) object.

Pascal

```
function FontToOleFont(Font: TFont): Variant;
```

C++

```
Variant FontToOleFont(TFont Font);
```

File

OleCtrls

Description

Use FontToOleFont to convert a font from TFont (see page 2730) to IFontDispatch format. The interface uses the TFont (see page 2730) object to implement the methods of the interface.

See Also

OleFontToFont (see page 6123)

OleStrToStrVar

StringToOleStr

OleStrToString

63.2.3 OleCtrls.OleFontToFont Function

Fills in a TFont (see page 2730) structure to represent an IFontDispatch.

Pascal

```
procedure OleFontToFont(const OleFont: Variant; Font: TFont);
```

C++

```
OleFontToFont(const Variant OleFont, TFont Font);
```

File

OleCtrls

Description

Use OleFontToFont to convert a font from COM (IFontDispatch) to TFont (see page 2730) format.

See Also

FontToOleFont (see page 6122)

OleStrToStrVar

StringToOleStr

OleStrToString

63.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TControlData (see page 6124)	This is record OleCtrls.TControlData.
	TControlData2 (see page 6125)	This is record OleCtrls.TControlData2.
	TEnumValue (see page 6125)	This is record OleCtrls.TEnumValue.

63.3.1 OleCtrls.TControlData Record

Pascal

```
TControlData = record
  ClassID: TGUID;
  EventIID: TGUID;
  EventCount: Longint;
  EventDispIDs: Pointer;
  LicenseKey: Pointer;
  Flags: DWORD;
  Version: Integer;
  FontCount: Integer;
  FontIDs: PDispIDList;
  PictureCount: Integer;
  PictureIDs: PDispIDList;
  Reserved: Integer;
  InstanceCount: Integer;
  EnumPropDescs: TList;
end;
```

C++

```
struct TControlData {
  TGUID ClassID;
  TGUID EventIID;
  Longint EventCount;
  void * EventDispIDs;
  void * LicenseKey;
  DWORD Flags;
  int Version;
  int FontCount;
  PDispIDList FontIDs;
  int PictureCount;
  PDispIDList PictureIDs;
  int Reserved;
  int InstanceCount;
  TList EnumPropDescs;
};
```

File

OleCtrls

Description

This is record OleCtrls.TControlData.

63.3.2 OleCtrls.TControlData2 Record

Pascal

```
TControlData2 = record
  ClassID: TGUID;
  EventIID: TGUID;
  EventCount: Longint;
  EventDispIDs: Pointer;
  LicenseKey: Pointer;
  Flags: DWORD;
  Version: Integer;
  FontCount: Integer;
  FontIDs: PDispIDList;
  PictureCount: Integer;
  PictureIDs: PDispIDList;
  Reserved: Integer;
  InstanceCount: Integer;
  EnumPropDescs: TList;
  FirstEventOfs: Cardinal;
end;
```

C++

```
struct TControlData2 {
  TGUID ClassID;
  TGUID EventIID;
  Longint EventCount;
  void * EventDispIDs;
  void * LicenseKey;
  DWORD Flags;
  int Version;
  int FontCount;
  PDispIDList FontIDs;
  int PictureCount;
  PDispIDList PictureIDs;
  int Reserved;
  int InstanceCount;
  TList EnumPropDescs;
  unsigned FirstEventOfs;
};
```

File

OleCtrls

Description

This is record OleCtrls.TControlData2.

63.3.3 OleCtrls.TEnumValue Record

Pascal

```
TEnumValue = record
  Value: Longint;
  Ident: string;
end;
```

C++

```
struct TEnumValue {
  Longint Value;
  UnicodeString Ident;
};
```

File

OleCtrls

Description

This is record OleCtrls.TEnumValue.

63.4 Types

The following table lists types in this documentation.

Types

Name	Description
PControlData (see page 6127)	This is type OleCtrls.PControlData.
PControlData2 (see page 6127)	This is type OleCtrls.PControlData2.
PEnumValueList (see page 6128)	This is type OleCtrls.PEnumValueList.
TEnumValueList (see page 6128)	This is type OleCtrls.TEnumValueList.
TOleEnum (see page 6128)	This is type OleCtrls.TOleEnum.
TServiceQuery (see page 6129)	This is type OleCtrls.TServiceQuery.

63.4.1 OleCtrls.PControlData Type

Pascal

```
PControlData = ^TControlData;
```

C++

```
^TControlData PControlData;
```

File

OleCtrls

Description

This is type OleCtrls.PControlData.

63.4.2 OleCtrls.PControlData2 Type

Pascal

```
PControlData2 = ^TControlData2;
```

C++

```
^TControlData2 PControlData2;
```

File

OleCtrls

Description

This is type OleCtrls.PControlData2.

63.4.3 OleCtrls.PEnumValueList Type

Pascal

```
PEnumValueList = ^TEnumValueList;
```

C++

```
^TEnumValueList PEnumValueList;
```

File

OleCtrls

Description

This is type OleCtrls.PEnumValueList.

63.4.4 OleCtrls.TEnumValueList Type

Pascal

```
TEnumValueList = array[0..32767] of TEnumValue;
```

C++

```
array[0..32767] of TEnumValue TEnumValueList;
```

File

OleCtrls

Description

This is type OleCtrls.TEnumValueList.

63.4.5 OleCtrls.TOleEnum Type

Pascal

```
TOleEnum = ActiveX.TOleEnum;
```

C++

```
ActiveX.TOleEnum TOleEnum;
```

File

OleCtrls

Description

This is type OleCtrls.TOleEnum.

63.4.6 OleCtrls.TServiceQuery Type

Pascal

```
TServiceQuery = function (const rsid, iid: TGuid; out obj): HResult of object;
```

C++

```
(const rsid, iid: TGuid; out obj): HResult ( TServiceQuery)();
```

File

OleCtrls

Description

This is type OleCtrls.TServiceQuery.

64 OleServer Namespace

This is namespace OleServer.

Topics

Name	Description
Classes (see page 6132)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6148)	The following table lists structs, records, enums in this documentation.
Types (see page 6150)	The following table lists types in this documentation.

Classes

	Name	Description
	TOleServer (see page 6132)	TOleServer is the base class for components that represent an imported COM server.
	TServerEventDispatch (see page 6144)	This is class OleServer.TServerEventDispatch.

Structs, Records, Enums

	Name	Description
	TConnectKind (see page 6148)	This is record OleServer.TConnectKind.
	TServerData (see page 6148)	This is record OleServer.TServerData.

Types

Name	Description
PServerData (see page 6150)	This is type OleServer.PServerData.
TVariantArray (see page 6150)	This is type OleServer.TVariantArray.

64.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOleServer (see page 6132)	TOleServer is the base class for components that represent an imported COM server.
	TServerEventDispatch (see page 6144)	This is class OleServer.TServerEventDispatch.

64.1.1 TOleServer Class

TOleServer is the base class for components that represent an imported COM server.

Class Hierarchy

Pascal

```
TOleServer = class(TComponent, IUnknown);
```

C++

```
class TOleServer : public TComponent, public IUnknown;
```

File

OleServer

Description

TOleServer is the base class for COM servers that are imported by choosing Project|Import Type Library from the IDE. TOleServer introduces several properties and methods that manage the connection to a COM server and for invoking its default events.

Descendants of TOleServer expose the properties, methods, and events of the server CoClass. When an application calls one of the TOleServer descendant's methods or sets one of its properties, the component automatically establishes a connection to the COM server. You can also connect explicitly by calling the Connect (see page 6136) method.

Do not create instances of TOleServer. TOleServer relies on abstract pure virtual methods to establish and break a connection to the COM server. Instead, use a descendant of TOleServer that wraps the desired server. Numerous TOleServer descendants are provided on the Servers tab of the Component Palette. In addition, you can create your own descendants by importing a server's type library.

See Also

Code Generated When You Import Type Library Information

64.1.1.1 TOleServer Members

The following tables list the members exposed by TOleServer.

Protected Methods

	Name	Description
	_AddRef (see page 6135)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 6136)	Decrements the reference count for the calling interface on a object.
	ConnectEvents (see page 6137)	Establishes a connection using the EventDispatch (see page 6143) property as an event sink.
	DisconnectEvents (see page 6137)	Terminates a connection that was established using the ConnectEvents (see page 6137) method.
	GetAutoConnect (see page 6138)	Returns the value of the AutoConnect (see page 6141) property.
	GetConnectKind (see page 6138)	Returns the value of the ConnectKind (see page 6142) property.
V	GetServer (see page 6138)	Returns an interface for the COM server.
A	InitServerData (see page 6139)	Initializes the ServerData (see page 6143) property to reflect the type library information of the COM server.
V	InvokeEvent (see page 6139)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 6140)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 6141) is True true.
V	QueryInterface (see page 6140)	Returns a reference to a specified interface if the OLE server supports that interface.
	SetAutoConnect (see page 6140)	Sets the value of the AutoConnect (see page 6141) property.
	SetConnectKind (see page 6141)	Sets the value of the ConnectKind (see page 6142) property.

Protected Properties

	Name	Description
	EventDispatch (see page 6143)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 6143)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
A	Connect (see page 6136)	Establishes a connection to the COM server.
V	Create (see page 6133)	Allocates memory and constructs a safely initialized instance of a component.
V	Destroy (see page 6134)	Disposes of the component and its owned components.
A	Disconnect (see page 6137)	Terminates the connection to the COM server.

Published Properties

	Name	Description
	AutoConnect (see page 6141)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 6142)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 6143)	Specifies the machine running the COM server when ConnectKind (see page 6142) is ckRemote.

64.1.1.2 TOleServer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create

Destroy

System::TObject::Free

Owner

64.1.1.3 TOleServer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleServer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy

DestroyComponents

Destroying

System::TObject::Free

Notification

Owner

Release (see page 2133)

Components

64.1.1.4 TOleServer Methods

The methods of the TOleServer class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6135)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 6136)	Decrements the reference count for the calling interface on a object.
	ConnectEvents (see page 6137)	Establishes a connection using the EventDispatch (see page 6143) property as an event sink.
	DisconnectEvents (see page 6137)	Terminates a connection that was established using the ConnectEvents (see page 6137) method.
	GetAutoConnect (see page 6138)	Returns the value of the AutoConnect (see page 6141) property.
	GetConnectKind (see page 6138)	Returns the value of the ConnectKind (see page 6142) property.
V	GetServer (see page 6138)	Returns an interface for the COM server.
A	InitServerData (see page 6139)	Initializes the ServerData (see page 6143) property to reflect the type library information of the COM server.
V	InvokeEvent (see page 6139)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 6140)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 6141) is True true.
V	QueryInterface (see page 6140)	Returns a reference to a specified interface if the OLE server supports that interface.
	SetAutoConnect (see page 6140)	Sets the value of the AutoConnect (see page 6141) property.
	SetConnectKind (see page 6141)	Sets the value of the ConnectKind (see page 6142) property.

Public Methods

	Name	Description
A	Connect (see page 6136)	Establishes a connection to the COM server.
A	Disconnect (see page 6137)	Terminates the connection to the COM server.

64.1.1.4.1 TOleServer._AddRef Method

Increments the reference count for the number of interface references dependent on the OLE server.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

_AddRef implements the IUnknown interface method, AddRef, which increments the reference count of objects using the OLE server's interface. _AddRef returns the resulting value of the reference count.

See Also

Invoke

QueryInterface (see page 6140)

_Release (see page 6136)

64.1.1.4.2 TOleServer._Release Method

Decrements the reference count for the calling interface on a object.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

_Release implements the IUnknown interface method, Release, which decrements the reference count of objects using the OLE server's interface. If the reference count on the interface falls to 0, the server object is freed from memory.

_Release returns the resulting value of the reference count.

See Also

Invoke

QueryInterface (see page 6140)

_AddRef (see page 6135)

64.1.1.4.3 TOleServer.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; virtual; abstract;
```

C++

```
virtual __fastcall Connect() = 0;
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect (see page 6141) property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind (see page 6142) is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind (see page 6142) is ckRemote, the RemoteMachineName (see page 6143) property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 6141)

ConnectKind (see page 6142)

RemoteMachineName (see page 6143)

Disconnect (see page 6137)

Using Component Wrappers

64.1.1.4.4 TOleServer.ConnectEvents Method

Establishes a connection using the EventDispatch (see page 6143) property as an event sink.

Pascal

```
procedure ConnectEvents(const Obj: IUnknown);
```

C++

```
__fastcall ConnectEvents(const IUnknown Obj);
```

Description

ConnectEvents descendants use ConnectEvents internally to implement their event handlers for events on the COM server. If the ConnectEvents descendant does not expose any server events, ConnectEvents is not used.

The Obj parameter is an interface for the COM server that generates events.

See Also

EventDispatch (see page 6143)

DisconnectEvents (see page 6137)

64.1.1.4.5 TOleServer.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; virtual; abstract;
```

C++

```
virtual __fastcall Disconnect() = 0;
```

Description

Call Disconnect to terminate a connection formed using the Connect (see page 6136) method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 6136)

Using Component Wrappers

64.1.1.4.6 TOleServer.DisconnectEvents Method

Terminates a connection that was established using the ConnectEvents (see page 6137) method.

Pascal

```
procedure DisconnectEvents(const Obj: Iunknown);
```

C++

```
__fastcall DisconnectEvents(const Iunknown Obj);
```

Description

DisconnectEvents descendants use DisconnectEvents internally to terminate the connection established by ConnectEvents (see page 6137). If the DisconnectEvents descendant does not expose any server events, DisconnectEvents is not used.

The Obj parameter is an interface for the COM server that generates events.

See Also

ConnectEvents (see page 6137)

64.1.1.4.7 TOleServer.GetAutoConnect Method

Returns the value of the AutoConnect (see page 6141) property.

Pascal

```
function GetAutoConnect: Boolean;
```

C++

```
__fastcall Boolean GetAutoConnect();
```

Description

GetAutoConnect is the protected read implementation for the AutoConnect (see page 6141) property.

See Also

AutoConnect (see page 6141)

64.1.1.4.8 TOleServer.GetConnectKind Method

Returns the value of the ConnectKind (see page 6142) property.

Pascal

```
function GetConnectKind: TConnectKind;
```

C++

```
__fastcall TConnectKind GetConnectKind();
```

Description

GetConnectKind is the protected read implementation for the ConnectKind (see page 6142) property.

See Also

ConnectKind (see page 6142)

64.1.1.4.9 TOleServer.GetServer Method

Returns an interface for the COM server.

Pascal

```
function GetServer: IUnknown; virtual;
```

C++

```
virtual __fastcall IUnknown GetServer();
```

Description

GetServer descendants call GetServer from the implementation of their Connect (see page 6136) methods. GetServer obtains an interface for the COM server described by ServerData (see page 6143). The ConnectKind (see page 6142) property indicates how GetServer can obtain the interface.

See Also

Connect (see page 6136)

ConnectKind (see page 6142)

ServerData (see page 6143)

64.1.1.4.10 TOleServer.InitServerData Method

Initializes the ServerData (see page 6143) property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; virtual; abstract;
```

C++

```
virtual __fastcall InitServerData() = 0;
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData (see page 6143) with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData (see page 6143) property reflects its particular COM server.

See Also

ServerData (see page 6143)

Create (see page 6133)

64.1.1.4.11 TOleServer.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); virtual;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch (see page 6143) can pass

events on to the appropriate event handlers.

See Also

EventDispatch (see page 6143)

64.1.1.4.12 TOleServer.Loaded Method

Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 6141) is True true.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

The VCL streaming system automatically calls Loaded after the Loaded component is loaded into memory.

Loaded checks the AutoConnect (see page 6141) property, and if it is True true, calls the Connect (see page 6136) method to establish a connection.

See Also

AutoConnect (see page 6141)

Loaded

64.1.1.4.13 TOleServer.QueryInterface Method

Returns a reference to a specified interface if the OLE server supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface implements the IUnknown interface method QueryInterface. QueryInterface returns a reference to the interface specified by the IID parameter as the Obj parameter. If the OLE server does not support the interface, the Obj parameter returns nil NULL and QueryInterface returns a nonzero error code.

See Also

_AddRef (see page 6135)

_Release (see page 6136)

Invoke

64.1.1.4.14 TOleServer.SetAutoConnect Method

Sets the value of the AutoConnect (see page 6141) property.

Pascal

```
procedure SetAutoConnect(flag: Boolean);
```

C++

```
__fastcall SetAutoConnect(Boolean flag);
```

Description

SetAutoConnect is the protected write implementation for the AutoConnect (see page 6141) property.

See Also

AutoConnect (see page 6141)

64.1.1.4.15 TOleServer.SetConnectKind Method

Sets the value of the ConnectKind (see page 6142) property.

Pascal

```
procedure SetConnectKind(ck: TConnectKind);
```

C++

```
__fastcall SetConnectKind(TConnectKind ck);
```

Description

SetConnectKind is the protected write implementation for the ConnectKind (see page 6142) property.

See Also

ConnectKind (see page 6142)

64.1.1.5 TOleServer Properties

The properties of the TOleServer class are listed here.

Protected Properties

	Name	Description
	EventDispatch (see page 6143)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 6143)	Encodes information about the COM server to which ServerData connects.

Published Properties

	Name	Description
	AutoConnect (see page 6141)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 6142)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 6143)	Specifies the machine running the COM server when ConnectKind (see page 6142) is ckRemote.

64.1.1.5.1 TOleServer.AutoConnect Property

Specifies whether the component automatically connects to the server.

Pascal

```
property AutoConnect: Boolean;
```

C++

```
__property Boolean AutoConnect;
```

Description

Set AutoConnect to True true to specify whether the server should be loaded automatically when the application first starts up. If AutoConnect is False false, the application establishes a connection when it calls one of the AutoConnect descendant's methods, sets a property, or calls the Connect (see page 6136) method.

Note: AutoConnect only establishes a connection when the application is first run. Changing the AutoConnect property at runtime has no effect.

See Also

Connect (see page 6136)

Using Component Wrappers

64.1.1.5.2 TOleServer.ConnectKind Property

Indicates the type of process to which the ConnectKind component establishes a connection.

Pascal

```
property ConnectKind: TConnectKind;
```

C++

```
__property TConnectKind ConnectKind;
```

Description

Set ConnectKind to indicate how the ConnectKind component establishes a connection. ConnectKind establishes the connection when the application is run (if AutoConnect (see page 6141) is True true) or when the application calls the Connect (see page 6136) (or ConnectTo) method.

The following table lists the possible values:

Value	Meaning
ckRunningOrNew	Attach to a running server or create a new instance of the server.
ckNewInstance	Always create a new instance of the server.
ckRunningInstance	Only attach to a running instance of the server.
ckRemote	Bind to a remote instance of the server. When using this option, you must supply a value for RemoteMachineName (see page 6143).
ckAttachToInterface	Don't bind to the server. Instead, the application supplies an interface using the ConnectTo method, which is introduced in descendant classes. This option can't be used with the AutoConnect (see page 6141) property.

See Also

Connect (see page 6136)

RemoteMachineName (see page 6143)

AutoConnect (see page 6141)

Using Component Wrappers

64.1.1.5.3 TOleServer.EventDispatch Property

Pascal

```
property EventDispatch: TServerEventDispatch;
```

C++

```
__property TServerEventDispatch EventDispatch;
```

Description

This is EventDispatch, a member of class TOleServer.

64.1.1.5.4 TOleServer.RemoteMachineName Property

Specifies the machine running the COM server when ConnectKind (see page 6142) is ckRemote.

Pascal

```
property RemoteMachineName: string;
```

C++

```
__property UnicodeString RemoteMachineName;
```

Description

Use RemoteMachineName when the COM server instance is running remotely on another machine. Specify the name of the machine on which the COM server is running as the value of RemoteMachineName.

Note: RemoteMachineName is only used if ConnectKind (see page 6142) is ckRemote. Setting RemoteMachineName to a non-empty string sets ConnectKind (see page 6142) to ckRemote.

See Also

ConnectKind (see page 6142)

Connect (see page 6136)

Using Component Wrappers

64.1.1.5.5 TOleServer.ServerData Property

Encodes information about the COM server to which ServerData connects.

Pascal

```
property ServerData: PServerData;
```

C++

```
__property PServerData ServerData;
```

Description

ServerData uses ServerData internally to store type library information describing its COM server. Each ServerData descendant initializes this property in the constructor, specifying the information it needs to establish a connection.

See Also

InitServerData (see page 6139)

64.1.2 TServerEventDispatch Class

Class Hierarchy

Pascal

```
TServerEventDispatch = class(TObject, IUnknown, IDispatch);
```

C++

```
class TServerEventDispatch : public TObject, public IUnknown, public IDispatch;
```

File

OleServer

Description

This is class OleServer.TServerEventDispatch.

64.1.2.1 TServerEventDispatch Members

The following tables list the members exposed by TServerEventDispatch.

Protected Methods

	Name	Description
	_AddRef (see page 6145)	This is _AddRef, a member of class TServerEventDispatch.
	_Release (see page 6145)	This is _Release, a member of class TServerEventDispatch.
	GetIDsOfNames (see page 6145)	This is GetIDsOfNames, a member of class TServerEventDispatch.
	GetTypeInfo (see page 6146)	This is GetTypeInfo, a member of class TServerEventDispatch.
	GetTypeInfoCount (see page 6146)	This is GetTypeInfoCount, a member of class TServerEventDispatch.
	Invoke (see page 6146)	This is Invoke, a member of class TServerEventDispatch.
	QueryInterface (see page 6146)	This is QueryInterface, a member of class TServerEventDispatch.
	ServerDisconnect (see page 6147)	This is ServerDisconnect, a member of class TServerEventDispatch.

Protected Properties

	Name	Description
	Server (see page 6147)	This is Server, a member of class TServerEventDispatch.

Public Methods

	Name	Description
	Create (see page 6144)	This is Create, a member of class TServerEventDispatch.

64.1.2.2 TServerEventDispatch.Create Constructor

Pascal

```
constructor Create(Server: TOleServer);
```

C++

```
__fastcall TServerEventDispatch(TOleServer Server);
```

Description

This is Create, a member of class TServerEventDispatch.

64.1.2.3 TServerEventDispatch Methods

The methods of the TServerEventDispatch class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6145)	This is _AddRef, a member of class TServerEventDispatch.
	_Release (see page 6145)	This is _Release, a member of class TServerEventDispatch.
	GetIDsOfNames (see page 6145)	This is GetIDsOfNames, a member of class TServerEventDispatch.
	GetTypeInfo (see page 6146)	This is GetTypeInfo, a member of class TServerEventDispatch.
	GetTypeInfoCount (see page 6146)	This is GetTypeInfoCount, a member of class TServerEventDispatch.
	Invoke (see page 6146)	This is Invoke, a member of class TServerEventDispatch.
	QueryInterface (see page 6146)	This is QueryInterface, a member of class TServerEventDispatch.
	ServerDisconnect (see page 6147)	This is ServerDisconnect, a member of class TServerEventDispatch.

64.1.2.3.1 TServerEventDispatch._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TServerEventDispatch.

64.1.2.3.2 TServerEventDispatch._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TServerEventDispatch.

64.1.2.3.3 TServerEventDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
```

```
  LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TServerEventDispatch.

64.1.2.3.4 TServerEventDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TServerEventDispatch.

64.1.2.3.5 TServerEventDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TServerEventDispatch.

64.1.2.3.6 TServerEventDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TServerEventDispatch.

64.1.2.3.7 TServerEventDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TServerEventDispatch.

64.1.2.3.8 TServerEventDispatch.ServerDisconnect Method

Pascal

```
function ServerDisconnect: Boolean;
```

C++

```
__fastcall Boolean ServerDisconnect();
```

Description

This is ServerDisconnect, a member of class TServerEventDispatch.

64.1.2.4 TServerEventDispatch Properties

The properties of the TServerEventDispatch class are listed here.

Protected Properties

	Name	Description
	Server (see page 6147)	This is Server, a member of class TServerEventDispatch.

64.1.2.4.1 TServerEventDispatch.Server Property

Pascal

```
property Server: TOleServer;
```

C++

```
__property TOleServer Server;
```

Description

This is Server, a member of class TServerEventDispatch.

64.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TConnectKind (see page 6148)	This is record OleServer.TConnectKind.

Records

	Name	Description
	TServerData (see page 6148)	This is record OleServer.TServerData.

64.2.1 OleServer.TConnectKind Enumeration

Pascal

```
TConnectKind = (
  ckRunningOrNew,
  ckNewInstance,
  ckRunningInstance,
  ckRemote,
  ckAttachToInterface
);
```

C++

```
enum TConnectKind {
  ckRunningOrNew,
  ckNewInstance,
  ckRunningInstance,
  ckRemote,
  ckAttachToInterface
};
```

File

OleServer

Description

This is record OleServer.TConnectKind.

64.2.2 OleServer.TServerData Record

Pascal

```
TServerData = record
  ClassID: TGUID;
  IntfIID: TGUID;
  EventIID: TGUID;
  LicenseKey: Pointer;
  Version: Integer;
  InstanceCount: Integer;
```

```
  end;
```

C++

```
  struct TServerData {
    TGUID ClassID;
    TGUID IntfIID;
    TGUID EventIID;
    void * LicenseKey;
    int Version;
    int InstanceCount;
  };
```

File

OleServer

Description

This is record OleServer.TServerData.

64.3 Types

The following table lists types in this documentation.

Types

Name	Description
PServerData (see page 6150)	This is type OleServer.PServerData.
TVariantArray (see page 6150)	This is type OleServer.TVariantArray.

64.3.1 OleServer.PServerData Type

Pascal

```
PServerData = ^TServerData;
```

C++

```
^TServerData PServerData;
```

File

OleServer

Description

This is type OleServer.PServerData.

64.3.2 OleServer.TVariantArray Type

Pascal

```
TVariantArray = Array of OleVariant;
```

C++

```
Array of OleVariant TVariantArray;
```

File

OleServer

Description

This is type OleServer.TVariantArray.

65 OPConvert Namespace

This is namespace OPConvert.

Topics

Name	Description
Interfaces (see page 6152)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 6162)	The following table lists structs, records, enums in this documentation.
Types (see page 6163)	The following table lists types in this documentation.

Interfaces

	Name	Description
	IOPConvert (see page 6152)	IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

Structs, Records, Enums

	Name	Description
	TSOAPConvertOption (see page 6162)	This is record OPConvert.TSOAPConvertOption.

Types

Name	Description
TSOAPConvertOptions (see page 6163)	This is type OPConvert.TSOAPConvertOptions.

65.1 Interfaces

The following table lists interfaces in this documentation.

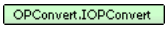
Interfaces

	Name	Description
	IOPConvert (see page 6152)	IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

65.1.1 IOPConvert Interface

IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

Class Hierarchy



Pascal

```
IOPConvert = interface;
```

C++

```
__interface IOPConvert;
```

File

OPConvert

Description

IOPConvert is implemented by classes that marshal and unmarshal method calls in Web Service applications. It defines methods for marshaling and unmarshaling method calls, as well as any exceptions that occur when attempting to execute those method calls.

The IOPConvert interface is used by remote interfaced objects on the client (TRIO descendants) and invoker components on the server (TSoapPascalInvoker descendants). Both the remote interfaced object and the invoker classes use the same IOPConvert implementation class, which represents a single encoding system (such as SOAP).

Classes that implement IOPConvert use the runtime type information (RTTI) on an invokable interface to interpret or encode method calls. They do not execute these calls, but rather, convert between transportable strings that are the encoding of interface method calls and an invocation context (TInvContext (see page 4369) object).

Note: C++ method declarations that use IOPConvert use the _di_IOPConvert type instead. This type is a DelphiInterface wrapper around the IOPConvert interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IOPConvert >  _di_IOPConvert;
```

See Also

TOPToSoapDomConvert (see page 6167)

TRio

TSoapPascalInvoker

TInvContext (see page 4369)

65.1.1.1 IOPConvert Members

The following tables list the members exposed by IOPConvert.

Public Methods

	Name	Description
	GetAttachments (see page 6154)	Returns the value of the Attachments (see page 6160) property.
	GetEncoding (see page 6154)	Returns the value of the Encoding (see page 6160) property.
	GetOptions (see page 6154)	Returns the value of the Options (see page 6161) property.
	GetTempDir (see page 6154)	Returns the value of the TempDir (see page 6161) property.
	InvContextToMsg (see page 6155)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 6155)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 6156)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 6156)	This is the overview for the MsgToInvContext method overload.
	ProcessResponse (see page 6157)	This is the overview for the ProcessResponse method overload.
	SetAttachments (see page 6158)	Sets the value of the Attachments (see page 6160) property.
	SetEncoding (see page 6159)	Sets the value of the Encoding (see page 6160) property.
	SetOptions (see page 6159)	Sets the value of the Options (see page 6161) property.
	SetTempDir (see page 6159)	Sets the value of the TempDir (see page 6161) property.

Public Properties

	Name	Description
	Attachments (see page 6160)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
	Encoding (see page 6160)	Indicates the character set to use for encoded method calls.
	Options (see page 6161)	Controls how certain types of conversions are performed.
	TempDir (see page 6161)	Specifies the directory where temporary files for attachments are saved.

65.1.1.2 IOPConvert Methods

The methods of the IOPConvert class are listed here.

Public Methods

	Name	Description
	GetAttachments (see page 6154)	Returns the value of the Attachments (see page 6160) property.
	GetEncoding (see page 6154)	Returns the value of the Encoding (see page 6160) property.
	GetOptions (see page 6154)	Returns the value of the Options (see page 6161) property.
	GetTempDir (see page 6154)	Returns the value of the TempDir (see page 6161) property.
	InvContextToMsg (see page 6155)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 6155)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 6156)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 6156)	This is the overview for the MsgToInvContext method overload.
	ProcessResponse (see page 6157)	This is the overview for the ProcessResponse method overload.
	SetAttachments (see page 6158)	Sets the value of the Attachments (see page 6160) property.
	SetEncoding (see page 6159)	Sets the value of the Encoding (see page 6160) property.
	SetOptions (see page 6159)	Sets the value of the Options (see page 6161) property.
	SetTempDir (see page 6159)	Sets the value of the TempDir (see page 6161) property.

65.1.1.2.1 IOPConvert.GetAttachments Method

Returns the value of the Attachments (see page 6160) property.

Pascal

```
function GetAttachments: TSoapDataList;
```

C++

```
__fastcall TSoapDataList GetAttachments();
```

Description

GetAttachments is the read implementation for the Attachments (see page 6160) property.

See Also

Attachments (see page 6160)

65.1.1.2.2 IOPConvert.GetEncoding Method

Returns the value of the Encoding (see page 6160) property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation for the Encoding (see page 6160) property.

See Also

Encoding (see page 6160)

65.1.1.2.3 IOPConvert.GetOptions Method

Returns the value of the Options (see page 6161) property.

Pascal

```
function GetOptions: TSOAPConvertOptions;
```

C++

```
__fastcall TSOAPConvertOptions GetOptions();
```

Description

GetOptions is the read implementation for the Options (see page 6161) property.

See Also

Options (see page 6161)

65.1.1.2.4 IOPConvert.GetTempDir Method

Returns the value of the TempDir (see page 6161) property.

Pascal

```
function GetTempDir: string;
```

C++

```
__fastcall UnicodeString GetTempDir();
```

Description

GetTempDir is the read implementation for the TempDir (see page 6161) property.

See Also

TempDir (see page 6161)

65.1.1.2.5 IOPConvert.InvContextToMsg Method

Marshals a method call from an invocation context to a transportable string.

Pascal

```
function InvContextToMsg(const IntfMD: TIntfMetaData; MethNum: Integer; Con: TInvContext;
Headers: THeaderList): TStream;
```

C++

```
__fastcall TStream InvContextToMsg(const TIntfMetaData IntfMD, int MethNum, TInvContext Con,
THeaderList Headers);
```

Description

InvContextToMsg looks up a specified method call, including the types and values of any arguments, from an invocation context, and returns an encoded version of that method call that can be transported to the application that executes it.

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum identifies the called method by its position in the interface's method table.

Con is the invocation context that contains details of the method call, including the parameter values passed in, the calling conventions, and any other temporary data used when making the call.

Headers is a list of TSOAPHeader (see page 4448) descendants that represent the headers of the SOAP envelope that contained the method call.

InvContextToMsg returns a stream from which an application can read the encoded version of the method call. This encoded method call can then be passed to an IWebNode interface for execution.

See Also

MsgToInvContext (see page 6157)

Execute

65.1.1.2.6 IOPConvert.MakeFault Method

Encodes an exception that was raised when trying to execute a method on an invokable interface.

Pascal

```
procedure MakeFault(const Ex: Exception; EStream: TStream);
```

C++

```
__fastcall MakeFault(const Exception Ex, TStream EStream);
```

Description

MakeFault encodes an exception that a Web Service application catches when trying to execute a method on an invokable interface. It generates an encoded exception packet, which the client can decode and use to raise the exception on the client application.

Ex is the exception object that represents the exception that occurred.

EStream is a stream to which MakeFault writes the encoded exception packet.

See Also

MakeResponse (see page 6156)

65.1.1.2.7 IOPConvert.MakeResponse Method

Encodes the results of successfully executing a method on an invokable interface.

Pascal

```
procedure MakeResponse(const IntfMD: TIntfMetaData; const MethNum: Integer; Context:
TInvContext; Response: TStream; Headers: THeaderList);
```

C++

```
__fastcall MakeResponse(const TIntfMetaData IntfMD, const int MethNum, TInvContext Context,
TStream Response, THeaderList Headers);
```

Description

MakeResponse retrieves the results of executing a method on an invokable interface from an invocation context and packages them into a transportable string, which it writes to a stream. This string is passed back to the client, which can unpack it using the ProcessResponse (see page 6158) method.

IntfMD represents the runtime type information (RTTI) from the interface whose method was executed.

MethNum identifies the called method by its position in the interface's method table.

Context is the invocation context that contains details of the method call.

Response is a stream to which MakeResponse writes the encoded version of the method result and any output parameters.

Headers is a list of TSOAPHeader (see page 4448) descendants that represent the headers that should be added to the response message.

See Also

MakeFault (see page 6155)

ProcessResponse (see page 6158)

65.1.1.2.8 MsgToInvContext Method

This is the overview for the MsgToInvContext method overload.

Overload List

	Name	Description
	IOPConvert.MsgToInvContext (InvString, TIntfMetaData, Integer, TInvContext) (see page 6157)	Unpacks an encoded method call into an invocation context so that it can be executed.

	IOPConvert.MsgToInvContext (TStream, TIntfMetaData, Integer, TInvContext, THeaderList) (see page 6157)	Unpacks an encoded method call into an invocation context so that it can be executed.

65.1.1.2.8.1 IOPConvert.MsgToInvContext Method (InvString, TIntfMetaData, Integer, TInvContext)

Unpacks an encoded method call into an invocation context so that it can be executed.

Pascal

```
procedure MsgToInvContext(const Request: InvString; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext); overload;
procedure MsgToInvContext(const Request: TStream; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext; Headers: THeaderList); overload;
```

C++

```
__fastcall MsgToInvContext(const InvString Request, const TIntfMetaData IntfMD, int MethNum,
TInvContext Context);
__fastcall MsgToInvContext(const TStream Request, const TIntfMetaData IntfMD, int MethNum,
TInvContext Context, THeaderList Headers);
```

Description

MsgToInvContext takes an encoded message that a server application receives and uses it to fill out an invocation context, which can then be used to execute the method call.

Request is the encoded method call, or a stream from which to read that method call. It is generated on the client by a call to InvContextToMsg (see page 6155).

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum returns the position of the called method in the interface's method table.

Context is the invocation context to which this method adds the information encoded in Request.

Headers is an object list to which MsgToInvContext adds TSOAPHeader (see page 4448) descendants to represent every header in the encoded message.

See Also

InvContextToMsg (see page 6155)

65.1.1.2.9 ProcessResponse Method

This is the overview for the ProcessResponse method overload.

Overload List

	Name	Description
	IOPConvert.ProcessResponse (InvString, TIntfMetaData, TIntfMethEntry, TInvContext) (see page 6158)	Unpacks the response packet from a method call into an invocation context.
	IOPConvert.ProcessResponse (TStream, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 6158)	Unpacks the response packet from a method call into an invocation context.

65.1.1.2.9.1 IOPConvert.ProcessResponse Method (InvString, TIntfMetaData, TIntfMethEntry, TInvContext)

Unpacks the response packet from a method call into an invocation context.

Pascal

```
procedure ProcessResponse(const Resp: TStream; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); overload;
procedure ProcessResponse(const Resp: InvString; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext); overload; deprecated;
```

C++

```
__fastcall ProcessResponse(const TStream Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
__fastcall ProcessResponse(const InvString Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context);
```

Description

ProcessResponse takes the encoded results of executing a method on an invokable interface from an invocation context and unpacks them into an invocation context.

Response is the encoded results of executing a method on the invokable interface, or a stream from which those results can be read. It is generated on the server by the MakeResponse (see page 6156) method if the call was successful, or the MakeFault (see page 6155) method if the call was unsuccessful.

XMLDoc is the interface to an XML document that represents the results of executing a method on the invokable interface. The values of its nodes come from the MakeResponse (see page 6156) method if the call was successful, or the MakeFault (see page 6155) method if the call was unsuccessful.

IntfMD describes the runtime type information (RTTI) of the interface whose method was called.

MD describes the runtime type information (RTTI) of the method that was called.

Context is the invocation context that ProcessResponse fills out based on the contents of Response.

Headers is a list that ProcessResponse fills with TSOAPHeader (see page 4448) descendants that represent the headers of the encoded result message.

See Also

MakeFault (see page 6155)

MakeResponse (see page 6156)

65.1.1.2.10 IOPConvert.SetAttachments Method

Sets the value of the Attachments (see page 6160) property.

Pascal

```
procedure SetAttachments(Value: TSoapDataList);
```

C++

```
__fastcall SetAttachments(TSoapDataList Value);
```

Description

SetAttachments is the write implementation for the Attachments (see page 6160) property.

Value is a list of TSoapAttachment objects that encapsulate the current set of attachments.

See Also

Attachments (see page 6160)

65.1.1.2.11 IOPConvert.SetEncoding Method

Sets the value of the Encoding (see page 6160) property.

Pascal

```
procedure SetEncoding(const Encoding: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Encoding);
```

Description

SetEncoding is the write implementation for the Encoding (see page 6160) property.

Encoding (see page 6160) is the name of the new encoding to use for method calls.

See Also

Encoding (see page 6160)

65.1.1.2.12 IOPConvert.SetOptions Method

Sets the value of the Options (see page 6161) property.

Pascal

```
procedure SetOptions(const Value: TSOAPConvertOptions);
```

C++

```
__fastcall SetOptions(const TSOAPConvertOptions Value);
```

Description

SetOptions is the write implementation for the Options (see page 6161) property.

Value is the new set of options to use when marshaling and unmarshaling method calls.

See Also

Options (see page 6161)

65.1.1.2.13 IOPConvert.SetTempDir Method

Sets the value of the TempDir (see page 6161) property.

Pascal

```
procedure SetTempDir(const Value: string);
```

C++

```
__fastcall SetTempDir(const UnicodeString Value);
```

Description

SetTempDir is the write implementation for the TempDir (see page 6161) property.

Value is the new temporary directory to use.

See Also

TempDir (see page 6161)

65.1.1.3 IOPConvert Properties

The properties of the IOPConvert class are listed here.

Public Properties

	Name	Description
	Attachments (see page 6160)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
	Encoding (see page 6160)	Indicates the character set to use for encoded method calls.
	Options (see page 6161)	Controls how certain types of conversions are performed.
	TempDir (see page 6161)	Specifies the directory where temporary files for attachments are saved.

65.1.1.3.1 IOPConvert.Attachments Property

Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Pascal

```
property Attachments: TSoapDataList;
```

C++

```
__property TSoapDataList Attachments;
```

Description

When a method on an invokable interface requires the user to send or receive attachments, these attachments are each represented by a TSoapAttachment object. Attachments lists the TSoapAttachment objects for each attachment of the current method.

Note: When the application that contains IOPConvert (see page 6152) receives attachments, it writes them to a temporary file rather than storing them in memory. The location for these temporary files is specified by the TempDir (see page 6161) property.

See Also

TempDir (see page 6161)

TSOAPAttachment (see page 4435)

65.1.1.3.2 IOPConvert.Encoding Property

Indicates the character set to use for encoded method calls.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Use Encoding to indicate the character set that is used for encoded method calls. If you do not specify an encoding (that is, if Encoding is an empty string), UTF-8 is assumed.

See Also

DefaultDOMVendor

Encoding

65.1.1.3.3 IOPConvert.Options Property

Controls how certain types of conversions are performed.

Pascal

```
property Options: TSOAPConvertOptions;
```

C++

```
__property TSOAPConvertOptions Options;
```

Description

Set Options to indicate how you want to handle various types of conversions. This set can include zero or more of the TSOAPConvertOption (see page 6162) flags.

See Also

MsgToInvContext (see page 6157)

ClassOptions (see page 4417)

InvContextToMsg (see page 6155)

65.1.1.3.4 IOPConvert.TempDir Property

Specifies the directory where temporary files for attachments are saved.

Pascal

```
property TempDir: string;
```

C++

```
__property UnicodeString TempDir;
```

Description

When the application receives an attachment as the value of a parameter (or the return value) of the current method on the invokable interface, it writes the attachment to a temporary file. This temporary file becomes the value of the CacheFile property on the TSoapAttachment object that represents the attachment.

Use TempDir to specify the directory where temporary files for attachments are written.

Note: If TempDir is not set, the application uses the default temporary directory. On Windows, this is the directory that the Windows API GetTempPath returns. On Linux, this is obtained from the TMPDIR environment setting (or, if TMPDIR is not set, /tmp/)

See Also

Attachments (see page 6160)

CacheFile (see page 4444)

65.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSOAPConvertOption (see page 6162)	This is record OPConvert.TSOAPConvertOption.

65.2.1 OPConvert.TSOAPConvertOption Enumeration

Pascal

```
TSOAPConvertOption = (
  soSendUntyped,
  soSendMultiRefObj,
  soSendMultiRefArray,
  soTryAllSchema,
  soRootRefNodesToBody,
  soDocument,
  soReturnSuccessForFault,
  soUTF8InHeader,
  soDontSendEmptyNodes,
  soCacheMimeResponse,
  soDontClearOutBoundHeaders,
  soCustomFaultAtDetailsNode,
  soLiteralParams,
  soUTF8EncodeXML,
  soXXXXHdr
);
```

C++

```
enum TSOAPConvertOption {
  soSendUntyped,
  soSendMultiRefObj,
  soSendMultiRefArray,
  soTryAllSchema,
  soRootRefNodesToBody,
  soDocument,
  soReturnSuccessForFault,
  soUTF8InHeader,
  soDontSendEmptyNodes,
  soCacheMimeResponse,
  soDontClearOutBoundHeaders,
  soCustomFaultAtDetailsNode,
  soLiteralParams,
  soUTF8EncodeXML,
  soXXXXHdr
};
```

File

OPConvert

Description

This is record OPConvert.TSOAPConvertOption.

65.3 Types

The following table lists types in this documentation.

Types

Name	Description
TSOAPConvertOptions (see page 6163)	This is type OPConvert.TSOAPConvertOptions.

65.3.1 OPConvert.TSOAPConvertOptions Type

Pascal

```
TSOAPConvertOptions = set of TSOAPConvertOption;
```

C++

```
TSOAPConvertOption TSOAPConvertOptions;
```

File

OPConvert

Description

This is type OPConvert.TSOAPConvertOptions.

66 OPToSOAPDomConv Namespace

This is namespace OPToSOAPDomConv.

Topics

Name	Description
Classes (see page 6167)	The following table lists classes in this documentation.
Functions (see page 6203)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6204)	The following table lists structs, records, enums in this documentation.
Types (see page 6206)	The following table lists types in this documentation.
Variables (see page 6209)	The following table lists variables in this documentation.
Constants (see page 6210)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESOAPDomConvertError (see page 6167)	ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.
	TOPToSoapDomConvert (see page 6167)	TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.
	TSOAPDomConv (see page 6178)	TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

Constants

Name	Description
SVarArrayType (see page 6210)	This is constant OPToSOAPDomConv.SVarArrayType.

Functions

	Name	Description
	GetOrdPropEx (see page 6203)	This is function OPToSOAPDomConv.GetOrdPropEx.

Structs, Records, Enums

	Name	Description
	ConvNodeState (see page 6204)	This is record OPToSOAPDomConv.ConvNodeState.
	TMultiRefNodeElem (see page 6204)	This is record OPToSOAPDomConv.TMultiRefNodeElem.
	TMultiRefNodeMapElem (see page 6205)	This is record OPToSOAPDomConv.TMultiRefNodeMapElem.
	TSOAPArrayElemDesc (see page 6205)	This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

Types

Name	Description
TMemberDataNotReceivedEvent (see page 6206)	This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.
TMultiRefNodeMap (see page 6206)	This is type OPToSOAPDomConv.TMultiRefNodeMap.
TMultiRefNodes (see page 6207)	This is type OPToSOAPDomConv.TMultiRefNodes.
TSOAPArrayDesc (see page 6207)	This is type OPToSOAPDomConv.TSOAPArrayDesc.
TUnhandledNodeEvent (see page 6207)	This is type OPToSOAPDomConv.TUnhandledNodeEvent.
TXMLNodeArray (see page 6208)	This is type OPToSOAPDomConv.TXMLNodeArray.

Variables

Name	Description
DefArrayElemName (see page 6209)	This is variable OPToSOAPDomConv.DefArrayElemName.

66.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESOAPDomConvertError (see page 6167)	ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.
	TOPToSoapDomConvert (see page 6167)	TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.
	TSOAPDomConv (see page 6178)	TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

66.1.1 ESOAPDomConvertError Class

ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.

Class Hierarchy

Pascal

```
ESOAPDomConvertError = class(Exception);
```

C++

```
class ESOAPDomConvertError : public Exception;
```

File

OPToSOAPDomConv

Description

TOPToSoapDomConvert (see page 6167) raises ESOAPDomConvertError exceptions when it can't marshal or unmarshal a SOAP packet that encodes a method call or its results.

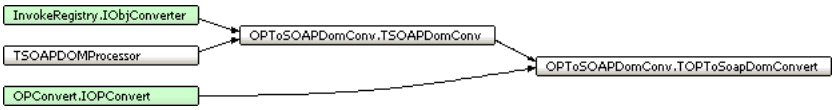
See Also

TOPToSoapDOMConvert

66.1.2 TOPToSoapDomConvert Class

TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.

Class Hierarchy



Pascal

```
TOPToSoapDomConvert = class(TSOAPDomConv, IOPConvert);
```

C++

```
class TOPToSoapDomConvert : public TSOAPDomConv, public IOPConvert;
```

File

OPToSOAPDomConv

Description

TOPToSoapDomConvert is an implementation of the IOPConvert (see page 6152) interface that handles the marshaling and unmarshaling of SOAP method calls. It uses a DOM parser to parse and edit the SOAP encoding of a method call and its results.

Typically, applications do not use TOPToSoapDomConvert directly. Rather, It is created and used by THTTPRio on client applications and THTTPSoapPascalInvoker or THTTPSoapCppInvoker on server applications.

TOPToSoapDomConvert converts between the SOAP encoding of a method call or its results and an invocation context. If the method call results in an exception, TOPToSoapDomConvert generates a SOAP error packet for that exception so that the client can raise the exception remotely.

See Also

THTTPRio

THTTPSoapPascalInvoker

THTTPSoapCppInvoker

IOPConvert (see page 6152)

TInvContext (see page 4369)

ESOAPDomConvertError (see page 6167)

66.1.2.1 TOPToSoapDomConvert Members

The following tables list the members exposed by TOPToSoapDomConvert.

Protected Methods

	Name	Description
	AddAttachment (see page 6183)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 6183)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 6183)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 6184)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
	CheckEncodingStyle (see page 6184)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 6184)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 6184)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 6184)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 6185)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 6186)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.
	ConvertSoapToEnum (see page 6186)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.
	ConvertSoapToNativeArray (see page 6186)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 6186)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 6187)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 6187)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 6188)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.

	CreateMultiRefNode (see page 6188)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 6188)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 6188)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 6189)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 6189)	This is CreateTypedNode, a member of class TSOAPDomConv.
	FinalizeMultiRefNodes (see page 6189)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 6189)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
	FindMultiRefNodeByID (see page 6190)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 6190)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 6190)	This is FindNodeByHREF, a member of class TSOAPDomConv.
	FindPrefixForURI (see page 6190)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetDataNode (see page 6191)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 6191)	This is GetElementType, a member of class TSOAPDomConv.
	GetNewID (see page 6191)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 6191)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 6192)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 6192)	This is GetOptions, a member of class TSOAPDomConv.
	GetTypeBySchemaNS (see page 6192)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	InitObjectFromSOAP (see page 6192)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	IsNodeAVarArray (see page 6193)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 6193)	This is IsObjectWriting, a member of class TSOAPDomConv.
	LoadObject (see page 6193)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 6193)	This is the overview for the MakeArrayNode method overload.
	MultiRefObject (see page 6194)	This is MultiRefObject, a member of class TSOAPDomConv.
	NewXMLDocument (see page 6174)	Creates a new empty XML document to hold the SOAP representation and returns its interface.
	NodeIsNULL (see page 6194)	This is NodeIsNULL, a member of class TSOAPDomConv.
	ObjectMemberNoShow (see page 6195)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 6195)	Returns the SOAP representation of a remotable object.
	ReadHeader (see page 6195)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 6196)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 6196)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 6196)	This is ReadRow, a member of class TSOAPDomConv.
	ReadVarArrayDim (see page 6196)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 6197)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 6197)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 6197)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 6197)	This is the overview for the SerializationOptions method overload.
	SetNodeType (see page 6198)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 6198)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 6199)	This is SetOptions, a member of class TSOAPDomConv.
	UnhandledNode (see page 6199)	This is UnhandledNode, a member of class TSOAPDomConv.
	WriteHeader (see page 6199)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 6199)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 6200)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 6200)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 6200)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.
	WriteVarArray (see page 6200)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 6201)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 6201)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	AddAttachment (see page 4462)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	ConvertNativeDataToSoap (see page 6185)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 6187)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.
	Create (see page 6171)	Creates a new TOPToSoapDomConvert (see page 6167) object.
	Destroy (see page 6171)	Frees the memory associated with the TOPToSoapDomConvert (see page 6167) object.
	FindAttachment (see page 4463)	Returns a TSOAPAttachment (see page 4435) instance that represent the attachment with a specified ID.
	GetAttachments (see page 6154)	Returns the value of the Attachments (see page 6160) property.
	GetEncoding (see page 6154)	Returns the value of the Encoding (see page 6160) property.
	GetOptions (see page 6154)	Returns the value of the Options (see page 6161) property.
	GetTempDir (see page 6154)	Returns the value of the TempDir (see page 6161) property.
	InitObjectFromSOAP (see page 4463)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	InvContextToMsg (see page 6172)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 6172)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 6173)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 6156)	This is the overview for the MsgToInvContext method overload.
	MsgToInvContext (see page 6173)	This is the overview for the MsgToInvContext method overload.
	ObjInstanceToSOAP (see page 4463)	Returns the SOAP representation of a remotable object.
	ProcessResponse (see page 6157)	This is the overview for the ProcessResponse method overload.
	ProcessResponse (see page 6175)	This is the overview for the ProcessResponse method overload.
	SetAttachments (see page 6158)	Sets the value of the Attachments (see page 6160) property.
	SetEncoding (see page 6159)	Sets the value of the Encoding (see page 6160) property.
	SetOptions (see page 6159)	Sets the value of the Options (see page 6161) property.
	SetTempDir (see page 6159)	Sets the value of the TempDir (see page 6161) property.

Public Properties

	Name	Description
	Attachments (see page 6176)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
	Options (see page 6161)	Controls how certain types of conversions are performed.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 6202)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 6202)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Published Properties

	Name	Description
	Encoding (see page 6177)	Indicates the character set to use for encoded method calls.
	Options (see page 6201)	Controls how the TSOAPDomConv (see page 6178) object performs certain types of conversions.
	TempDir (see page 6177)	Specifies the directory where temporary files for attachments are saved.
	WSDLView (see page 6178)	Provides access to information in a WSDL document that represents the SOAP request.

66.1.2.2 TOPToSoapDomConvert.Create Constructor

Creates a new TOPToSoapDomConvert (see page 6167) object.

Pascal

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

C++

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

Description

Most applications do not create instances of TOPToSoapDomConvert (see page 6167). It is created by the THTTPRio and TSoapPascalInvoker objects to handle the marshaling and unmarshaling of SOAP method calls.

The AOwner parameter specifies the component that is responsible for freeing the TOPToSoapDomConvert (see page 6167) instance.

See Also

Owner

THTTPRio

TSoapPascalInvoker

66.1.2.3 TOPToSoapDomConvert.Destroy Destructor

Frees the memory associated with the TOPToSoapDomConvert (see page 6167) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOPToSoapDomConvert();
```

Description

Typically, TOPToSoapDomConvert (see page 6167) objects are freed by the objects that create them to handle the marshaling and unmarshaling of SOAP method calls.

Do not call Destroy directly in an application. Instead, use the Free method, that checks whether the reference is nil before calling Destroy.

See Also

Create (see page 6171)

System::TObject::Free

66.1.2.4 TOPToSoapDomConvert Methods

The methods of the TOPToSoapDomConvert class are listed here.

Protected Methods

	Name	Description
	NewXMLDocument (see page 6174)	Creates a new empty XML document to hold the SOAP representation and returns its interface.

Public Methods

	Name	Description
	InvContextToMsg (see page 6172)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 6172)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 6173)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 6173)	This is the overview for the MsgToInvContext method overload.
	ProcessResponse (see page 6175)	This is the overview for the ProcessResponse method overload.

66.1.2.4.1 TOPToSoapDomConvert.InvContextToMsg Method

Marshals a method call from an invocation context to a transportable string.

Pascal

```
function InvContextToMsg(const IntfMD: TIntfMetaData; MethNum: Integer; Con: TInvContext;
Headers: THeaderList): TStream;
```

C++

```
__fastcall TStream InvContextToMsg(const TIntfMetaData IntfMD, int MethNum, TInvContext Con,
THeaderList Headers);
```

Description

InvContextToMsg looks up a specified method call, including the types and values of any arguments, from an invocation context, and returns an encoded version of that method call that can be transported to the application that executes it.

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum identifies the called method by its position in the interface's method table.

Con is the invocation context that contains details of the method call, including the parameter values passed in, the calling conventions, and any other temporary data used when making the call.

Headers is a list of TSOAPHeader (see page 4448) descendants that represent the headers of the SOAP envelope that contained the method call.

InvContextToMsg returns a stream from which an application can read the encoded version of the method call.

See Also

MsgToInvContext (see page 6174)

66.1.2.4.2 TOPToSoapDomConvert.MakeFault Method

Encodes an exception that was raised when trying to execute a method on an invokable interface.

Pascal

```
procedure MakeFault(const Ex: Exception; EStream: TStream); virtual;
```

C++

```
virtual __fastcall MakeFault(const Exception Ex, TStream EStream);
```

Description

MakeFault encodes an exception that a Web Service application catches when trying to execute a method on an invokable interface. It generates an encoded exception packet, which the client can decode and use to raise the exception on the client application.

Ex is the exception object that represents the exception that occurred.

EStream is a stream to which MakeFault writes the encoded exception packet.

See Also

MakeResponse (see page 6173)

66.1.2.4.3 TOPToSoapDomConvert.MakeResponse Method

Encodes the results of successfully executing a method on an invokable interface.

Pascal

```
procedure MakeResponse(const IntfMD: TIntfMetaData; const MethNum: Integer; Context:
TInvContext; Response: TStream; Headers: THeaderLIst); virtual;
```

C++

```
virtual __fastcall MakeResponse(const TIntfMetaData IntfMD, const int MethNum, TInvContext
Context, TStream Response, THeaderLIst Headers);
```

Description

MakeResponse retrieves the results of executing a method on an invokable interface from an invocation context and packages them into a transportable string, which it writes to a stream. This string is passed back to the client, which can unpack it using the ProcessResponse (see page 6175) method.

IntfMD represents the runtime type information (RTTI) from the interface whose method was executed.

MethNum identifies the called method by its position in the interface's method table.

Context is the invocation context that contains details of the method call.

Headers is a list of TSOAPHeader (see page 4448) descendants that should be converted to headers on the encoded response.

Response is a stream to which MakeResponse writes the encoded version of the method result and any output parameters.

Headers is a list of TSOAPHeader (see page 4448) descendants that represent the headers that should be added to the response message.

See Also

MakeFault (see page 6172)

ProcessResponse (see page 6175)

66.1.2.4.4 MsgToInvContext Method

This is the overview for the MsgToInvContext method overload.

Overload List

	Name	Description
	TOPToSoapDomConvert.MsgToInvContext (InvString, TIntfMetaData, Integer, TInvContext) (see page 6174)	Unpacks an encoded method call into an invocation context so that it can be executed.
	TOPToSoapDomConvert.MsgToInvContext (TStream, TIntfMetaData, Integer, TInvContext, THeaderList) (see page 6174)	Unpacks an encoded method call into an invocation context so that it can be executed.

66.1.2.4.4.1 TOPToSoapDomConvert.MsgToInvContext Method (InvString, TIntfMetaData, Integer, TInvContext)

Unpacks an encoded method call into an invocation context so that it can be executed.

Pascal

```
procedure MsgToInvContext(const Request: InvString; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext); virtual; overload;
procedure MsgToInvContext(const Request: TStream; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext; Headers: THeaderList); virtual; overload;
```

C++

```
virtual __fastcall MsgToInvContext(const InvString Request, const TIntfMetaData IntfMD, int
MethNum, TInvContext Context);
virtual __fastcall MsgToInvContext(const TStream Request, const TIntfMetaData IntfMD, int
MethNum, TInvContext Context, THeaderList Headers);
```

Description

MsgToInvContext takes an encoded message that a server application receives and uses it to fill out an invocation context, which can then be used to execute the method call.

Request is the encoded method call, or a stream from which to read that method call. It is generated on the client by a call to InvContextToMsg (see page 6172).

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum returns the position of the called method in the interface's method table.

Context is the invocation context to which this method adds the information encoded in Request.

Headers is an object list to which MsgToInvContext adds TSOAPHeader (see page 4448) descendants to represent every header in the encoded message.

See Also

InvContextToMsg (see page 6172)

66.1.2.4.5 TOPToSoapDomConvert.NewXMLDocument Method

Creates a new empty XML document to hold the SOAP representation and returns its interface.

Pascal

```
function NewXMLDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument NewXMLDocument();
```

Description

TOPToSoapDomConvert (see page 6167) uses NewXMLDocument internally to create the object that holds the SOAP representation of a message. It returns an interface for working with this representation. Note that the XML document that NewXMLDocument returns is empty. TOPToSoapDomConvert (see page 6167) subsequently uses the methods of the interface to fill out the body of this document.

66.1.2.4.6 ProcessResponse Method

This is the overview for the ProcessResponse method overload.

Overload List

	Name	Description
	TOPToSoapDomConvert.ProcessResponse (InvString, TIntfMetaData, TIntfMethEntry, TInvContext) (see page 6175)	Unpacks the response packet from a method call into an invocation context.
	TOPToSoapDomConvert.ProcessResponse (IXMLDocument, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 6175)	Unpacks the response packet from a method call into an invocation context.
	TOPToSoapDomConvert.ProcessResponse (TStream, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 6175)	Unpacks the response packet from a method call into an invocation context.

66.1.2.4.6.1 TOPToSoapDomConvert.ProcessResponse Method (IXMLDocument, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList)

Unpacks the response packet from a method call into an invocation context.

Pascal

```
procedure ProcessResponse(const Resp: InvString; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext); virtual; overload;
procedure ProcessResponse(const Resp: TStream; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); virtual; overload;
procedure ProcessResponse(const XMLDoc: IXMLDocument; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); virtual; overload;
```

C++

```
virtual __fastcall ProcessResponse(const InvString Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context);
virtual __fastcall ProcessResponse(const TStream Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
virtual __fastcall ProcessResponse(const IXMLDocument XMLDoc, const TIntfMetaData IntfMD,
const TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
```

Description

ProcessResponse takes the encoded results of executing a method on an invokable interface from an invocation context and unpacks them into an invocation context.

Response is the encoded results of executing a method on the invokable interface, or a stream from which those results can be read. It is generated on the server by the MakeResponse (see page 6173) method if the call was successful, or the MakeFault (see page 6172) method if the call was unsuccessful.

XMLDoc is the interface to an XML document that represents the results of executing a method on the invokable interface. The values of its nodes come from the MakeResponse (see page 6173) method if the call was successful, or the MakeFault (see page 6172) method if the call was unsuccessful.

IntfMD describes the runtime type information (RTTI) of the interface whose method was called.

MD describes the runtime type information (RTTI) of the method that was called.

Context is the invocation context that ProcessResponse fills out based on the contents of Response.

Headers is a list that ProcessResponse fills with TSOAPHeader (see page 4448) descendants that represent the headers of the encoded result message.

See Also

MakeFault (see page 6172)

MakeResponse (see page 6173)

66.1.2.5 TOPToSoapDomConvert Properties

The properties of the TOPToSoapDomConvert class are listed here.

Public Properties

	Name	Description
	Attachments (see page 6176)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Published Properties

	Name	Description
	Encoding (see page 6177)	Indicates the character set to use for encoded method calls.
	TempDir (see page 6177)	Specifies the directory where temporary files for attachments are saved.
	WSDLView (see page 6178)	Provides access to information in a WSDL document that represents the SOAP request.

66.1.2.5.1 TOPToSoapDomConvert.Attachments Property

Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Pascal

```
property Attachments: TSOAPDataList;
```

C++

```
__property TSOAPDataList Attachments;
```

Description

When a method on an invokable interface requires the user to send or receive attachments, these attachments are each represented by a TSoapAttachment object. Attachments lists the TSoapAttachment objects for each attachment of the current method.

Note: When the application that contains TOPToSoapDomConvert (see page 6167) receives attachments, it writes them to a temporary file rather than storing them in memory. The location for these temporary files is specified by the TempDir (see page 6177) property.

See Also

TempDir (see page 6177)

TSOAPAttachment (see page 4435)

66.1.2.5.2 TOPToSoapDomConvert.Encoding Property

Indicates the character set to use for encoded method calls.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Use Encoding to indicate the character set that is used for encoded method calls. Encoding causes the converter to make character set conversions using the DOM, rather than at the transport level. This approach is more efficient than transport-level encoding. If you set Encoding, the Options (see page 6201) property should not include soUTF8EncodeXML and soUTF8InHeader.

If you do not specify an encoding (that is, if Encoding is an empty string), TOPToSoapDomConvert (see page 6167) converts characters at the transport level assuming UTF-8.

TOPToSoapDomConvert (see page 6167) uses the DOM parser identified by the global DefaultDOMVendor variable to perform its conversions. Encoding must indicate an encoding that this default DOM parser supports.

Note: Some Web Services (such as Apache or .NET) require transport level encoding rather than DOM level encoding. In such cases, it is possible that setting Encoding can cause problems because the Web Service converts characters at the transport level and then converts a second time because of the encoding attribute that Encoding adds to the XML representation. If a Web Service requires transport-level encoding and assumes UTF8, Encoding should be an empty string and the Options (see page 6201) property should include soUTF8EncodeXML and soUTF8InHeader.

See Also

DefaultDOMVendor

Encoding

Options (see page 6201)

66.1.2.5.3 TOPToSoapDomConvert.TempDir Property

Specifies the directory where temporary files for attachments are saved.

Pascal

```
property TempDir: string;
```

C++

```
__property UnicodeString TempDir;
```

Description

When the application that includes this TOPToSoapDomConvert (see page 6167) instance receives an attachment as the value of a parameter (or the return value) of the current method, TOPToSoapDomConvert (see page 6167) writes the attachment to a temporary file. This temporary file becomes the value of the CacheFile property on the TSoapAttachment object that represents the attachment.

Use TempDir to specify the directory where TOPToSoapDomConvert (see page 6167) writes the temporary files for attachments.

Note: If TempDir is not set, the application uses the default temporary directory. On Windows, this is the directory that the Windows API GetTempPath returns. On Linux, this is obtained from the TMPDIR environment setting (or, if TMPDIR is not set, /tmp/)

See Also

Attachments (see page 6176)

CacheFile (see page 4444)

66.1.2.5.4 TOPToSoapDomConvert.WSDLView Property

Provides access to information in a WSDL document that represents the SOAP request.

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

WSDLView represents information about the SOAP request when the application uses a WSDL document to define information about the interface. If the application does not use a WSDL document, WSDLView is nil (Delphi) or NULL (C++).

WSDL includes the parsed WSDL document, as well as properties to identify the target method call.

66.1.3 TSOAPDomConv Class

TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

Class Hierarchy

Pascal

```
TSOAPDomConv = class(TSOAPDOMProcessor, IObjConverter);
```

C++

```
class TSOAPDomConv : public TSOAPDOMProcessor, public IObjConverter;
```

File

OPToSOAPDomConv

Description

TSOAPDomConv is the base class for DOM-based components that interpret and create SOAP method calls. It handles the logic of converting between the SOAP representation of a type and its native representation. TSOAPDomConv uses the remotable type registry (RemTypeRegistry (see page 4472)) to look up the native type or class that corresponds to a type from a SOAP method call and to generate an instance of that type to represent an incoming value. It uses runtime type information (RTTI) to convert a native type or remotable class to its SOAP representation.

TSOAPDomConv implements the IObjConverter (see page 4461) interface.

Do not create instances of TSOAPDomConv. It is intended only as a base class for components that interpret or create SOAP

method calls.

See Also

TOPToSoapDomConvert (see page 6167)

RemTypeRegistry

IObjConverter (see page 4461)

66.1.3.1 TSOAPDomConv Members

The following tables list the members exposed by TSOAPDomConv.

Protected Methods

	Name	Description
	AddAttachment (see page 6183)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 6183)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 6183)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 6184)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
	CheckEncodingStyle (see page 6184)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 6184)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 6184)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 6184)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 6185)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 6186)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.
	ConvertSoapToEnum (see page 6186)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.
	ConvertSoapToNativeArray (see page 6186)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 6186)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 6187)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 6187)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 6188)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.
	CreateMultiRefNode (see page 6188)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 6188)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 6188)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 6189)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 6189)	This is CreateTypedNode, a member of class TSOAPDomConv.
	FinalizeMultiRefNodes (see page 6189)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 6189)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
	FindMultiRefNodeByID (see page 6190)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 6190)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 6190)	This is FindNodeByHREF, a member of class TSOAPDomConv.
	FindPrefixForURI (see page 6190)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetDataNode (see page 6191)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 6191)	This is GetElementType, a member of class TSOAPDomConv.
	GetNewID (see page 6191)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 6191)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 6192)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 6192)	This is GetOptions, a member of class TSOAPDomConv.
	GetTypeBySchemaNS (see page 6192)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	InitObjectFromSOAP (see page 6192)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	IsNodeAVarArray (see page 6193)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 6193)	This is IsObjectWriting, a member of class TSOAPDomConv.

	LoadObject (see page 6193)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 6193)	This is the overview for the MakeArrayNode method overload.
	MultiRefObject (see page 6194)	This is MultiRefObject, a member of class TSOAPDomConv.
	NodeIsNULL (see page 6194)	This is NodeIsNULL, a member of class TSOAPDomConv.
	ObjectMemberNoShow (see page 6195)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 6195)	Returns the SOAP representation of a remotable object.
	ReadHeader (see page 6195)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 6196)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 6196)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 6196)	This is ReadRow, a member of class TSOAPDomConv.
	ReadVarArrayDim (see page 6196)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 6197)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 6197)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 6197)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 6197)	This is the overview for the SerializationOptions method overload.
	SetNodeType (see page 6198)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 6198)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 6199)	This is SetOptions, a member of class TSOAPDomConv.
	UnhandledNode (see page 6199)	This is UnhandledNode, a member of class TSOAPDomConv.
	WriteHeader (see page 6199)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 6199)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 6200)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 6200)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 6200)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.
	WriteVarArray (see page 6200)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 6201)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 6201)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	ConvertNativeDataToSoap (see page 6185)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 6187)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.
	Create (see page 6180)	Creates a new TSOAPDomConv (see page 6178) object.
	Destroy (see page 6181)	Frees the memory associated with the TSOAPDomConv (see page 6178) object.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 6202)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 6202)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Published Properties

	Name	Description
	Options (see page 6201)	Controls how the TSOAPDomConv (see page 6178) object performs certain types of conversions.

66.1.3.2 TSOAPDomConv.Create Constructor

Creates a new TSOAPDomConv (see page 6178) object.

Pascal

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

C++

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

Description

Typically, applications do not need to create instances of TSOAPDomConv (see page 6178). Instead, it is created and used by THTTPRio on client applications and THTTPSoapPascalInvoker or THTTPSoapCppInvoker on server applications.

The AOwner parameter specifies the component that is responsible for freeing the TSOAPDomConv (see page 6178) instance.

See Also

Owner

THTTPRio

THTTPSoapPascalInvoker

THTTPSoapCppInvoker

66.1.3.3 TSOAPDomConv.Destroy Destructor

Frees the memory associated with the TSOAPDomConv (see page 6178) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPDomConv();
```

Description

Do not call Destroy to free an instance of TSOAPDomConv (see page 6178). Instead, use the Free method, which first checks for a nil reference before calling Destroy.

See Also

System::TObject::Free

66.1.3.4 TSOAPDomConv Methods

The methods of the TSOAPDomConv class are listed here.

Protected Methods

	Name	Description
	AddAttachment (see page 6183)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 6183)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 6183)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 6184)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
	CheckEncodingStyle (see page 6184)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 6184)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 6184)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 6184)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 6185)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 6186)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.
	ConvertSoapToEnum (see page 6186)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.

	ConvertSoapToNativeArray (see page 6186)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 6186)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 6187)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 6187)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 6188)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.
	CreateMultiRefNode (see page 6188)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 6188)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 6188)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 6189)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 6189)	This is CreateTypedNode, a member of class TSOAPDomConv.
	FinalizeMultiRefNodes (see page 6189)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 6189)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
	FindMultiRefNodeByID (see page 6190)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 6190)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 6190)	This is FindNodeByHREF, a member of class TSOAPDomConv.
	FindPrefixForURI (see page 6190)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetDataNode (see page 6191)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 6191)	This is GetElementType, a member of class TSOAPDomConv.
	GetNewID (see page 6191)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 6191)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 6192)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 6192)	This is GetOptions, a member of class TSOAPDomConv.
	GetTypeBySchemaNS (see page 6192)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	InitObjectFromSOAP (see page 6192)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	IsNodeAVarArray (see page 6193)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 6193)	This is IsObjectWriting, a member of class TSOAPDomConv.
	LoadObject (see page 6193)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 6193)	This is the overview for the MakeArrayNode method overload.
	MultiRefObject (see page 6194)	This is MultiRefObject, a member of class TSOAPDomConv.
	NodeIsNULL (see page 6194)	This is NodeIsNULL, a member of class TSOAPDomConv.
	ObjectMemberNoShow (see page 6195)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 6195)	Returns the SOAP representation of a remotable object.
	ReadHeader (see page 6195)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 6196)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 6196)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 6196)	This is ReadRow, a member of class TSOAPDomConv.
	ReadVarArrayDim (see page 6196)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 6197)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 6197)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 6197)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 6197)	This is the overview for the SerializationOptions method overload.
	SetNodeType (see page 6198)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 6198)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 6199)	This is SetOptions, a member of class TSOAPDomConv.
	UnhandledNode (see page 6199)	This is UnhandledNode, a member of class TSOAPDomConv.
	WriteHeader (see page 6199)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 6199)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 6200)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 6200)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 6200)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.
	WriteVarArray (see page 6200)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 6201)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 6201)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	ConvertNativeDataToSoap (see page 6185)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 6187)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.

66.1.3.4.1 TSOAPDomConv.AddAttachment Method

Stores information about a SOAP attachment so that it is included in the final message that is created.

Pascal

```
procedure AddAttachment(Attachment: TSOAPAttachment; const AContentId: string);
```

C++

```
__fastcall AddAttachment(TSOAPAttachment Attachment, const UnicodeString AContentId);
```

Description

AddAttachment stores a SOAP attachment so that it can be used to generate part of the multipart form that includes the SOAP envelope as another of its parts.

Attachment is the attachment that should be added to the multipart form.

AContentId is a string that uniquely identifies the attachment. It is added as a header to the part of the form that contains the attachment so that XML in the SOAP envelope can refer to the attachment. When interpreting the encoded SOAP envelope, this value is passed to the FindAttachment method so that the information in the attachment part can be used to recreate the TSOAPAttachment instance.

See Also

FindAttachment (see page 4463)

66.1.3.4.2 TSOAPDomConv.AddMultiRefNode Method

Pascal

```
procedure AddMultiRefNode(const ID: string; Instance: Pointer);
```

C++

```
__fastcall AddMultiRefNode(const UnicodeString ID, void * Instance);
```

Description

This is AddMultiRefNode, a member of class TSOAPDomConv.

66.1.3.4.3 TSOAPDomConv.AddNamespaceDecl Method

Pascal

```
function AddNamespaceDecl(Node: IXMLNode; const URI: InvString): InvString;
```

C++

```
__fastcall InvString AddNamespaceDecl(IXMLNode Node, const InvString URI);
```

Description

This is AddNamespaceDecl, a member of class TSOAPDomConv.

66.1.3.4.4 TSOAPDomConv.AddObjectAsWriting Method

Pascal

```
procedure AddObjectAsWriting(Instance: TObject);
```

C++

```
__fastcall AddObjectAsWriting(TObject * Instance);
```

Description

This is AddObjectAsWriting, a member of class TSOAPDomConv.

66.1.3.4.5 TSOAPDomConv.CheckEncodingStyle Method

Pascal

```
procedure CheckEncodingStyle(Node: IXMLNode);
```

C++

```
__fastcall CheckEncodingStyle(IXMLNode Node);
```

Description

This is CheckEncodingStyle, a member of class TSOAPDomConv.

66.1.3.4.6 TSOAPDomConv.ChildNodesAreNull Method

Pascal

```
function ChildNodesAreNull(Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean ChildNodesAreNull(IXMLNode Node);
```

Description

This is ChildNodesAreNull, a member of class TSOAPDomConv.

66.1.3.4.7 TSOAPDomConv.ConvertByteArrayToSoap Method

Pascal

```
procedure ConvertByteArrayToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer);
```

C++

```
__fastcall ConvertByteArrayToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P);
```

Description

This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.

66.1.3.4.8 TSOAPDomConv.ConvertEnumToSoap Method

Pascal

```
function ConvertEnumToSoap(Info: PTypeInfo; P: Pointer; NumIndirect: Integer): InvString;
```

C++

```
__fastcall InvString ConvertEnumToSoap(PTypeInfo Info, void * P, int NumIndirect);
```

Description

This is ConvertEnumToSoap, a member of class TSOAPDomConv.

66.1.3.4.9 TSOAPDomConv.ConvertNativeArrayToSoap Method

Pascal

```
procedure ConvertNativeArrayToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer; NumIndirect: Integer; InlineElements: Boolean = False);
```

C++

```
__fastcall ConvertNativeArrayToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P, int NumIndirect, Boolean InlineElements = False);
```

Description

This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.

66.1.3.4.10 TSOAPDomConv.ConvertNativeDataToSoap Method

Converts a native type to its SOAP representation.

Pascal

```
procedure ConvertNativeDataToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer; NumIndirect: Integer); dynamic;
```

C++

```
__fastcall ConvertNativeDataToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P, int NumIndirect);
```

Description

ConvertNativeDataToSoap performs the conversion from a native type to its SOAP representation and adds that to the emerging DOM representation of a SOAP packet. It uses the options specified by the Options (see page 6201) property to determine whether to include type data and how to handle multiple references to the same remotable object.

RootNode is the node in the emerging SOAP representation that acts as a parent to any elements that represent an object that has multiple references.

Node is the parent node to which ConvertNativeDataToSoap adds a new child node that represents the native value.

Name is the type name of the SOAP encoding of the native type.

Info is a TypeInfo pointer that indicates the type in which the native data appears.

P is a pointer to the value to convert. It points (possibly with additional levels of indirection) to a value of the type indicated by Info.

NumIndirect indicates the number of levels of indirection by which P points to a value of the type indicated by Info.

See Also

ConvertSoapToNativeData (see page 6187)

Options (see page 6201)

66.1.3.4.11 TSOAPDomConv.ConvertObjectToSOAP Method

Pascal

```
procedure ConvertObjectToSOAP(const Name: InvString; ObjP: Pointer; RootNode: IXMLNOde; Node:
IXMLNOde; NumIndirect: Integer);
```

C++

```
__fastcall ConvertObjectToSOAP(const InvString Name, void * ObjP, IXMLNOde RootNode, IXMLNOde
Node, int NumIndirect);
```

Description

This is ConvertObjectToSOAP, a member of class TSOAPDomConv.

66.1.3.4.12 TSOAPDomConv.ConvertSoapToEnum Method

Pascal

```
function ConvertSoapToEnum(Info: PTypeInfo; S: InvString; IsNull: Boolean): Integer;
```

C++

```
__fastcall int ConvertSoapToEnum(PTypeInfo Info, InvString S, Boolean IsNull);
```

Description

This is ConvertSoapToEnum, a member of class TSOAPDomConv.

66.1.3.4.13 TSOAPDomConv.ConvertSoapToNativeArray Method

Pascal

```
function ConvertSoapToNativeArray(DataP: Pointer; TypeInfo: PTypeInfo; RootNode: IXMLNode;
Node: IXMLNode): Pointer;
```

C++

```
__fastcall void * ConvertSoapToNativeArray(void * DataP, PTypeInfo TypeInfo, IXMLNode
RootNode, IXMLNode Node);
```

Description

This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.

66.1.3.4.14 TSOAPDomConv.ConvertSoapToNativeArrayElem Method

Pascal

```
function ConvertSoapToNativeArrayElem(ArrayInfo: PTypeInfo; ElemInfo: PTypeInfo; RootNode:
IXMLNode; Node: IXMLNode; ArrayDesc: TSOAPArrayDesc; Dims: Integer; CurDim: Integer; DataP:
Pointer): Pointer;
```

C++

```
__fastcall void * ConvertSoapToNativeArrayElem(PTypeInfo ArrayInfo, PTypeInfo ElemInfo,
IXMLNode RootNode, IXMLNode Node, TSOAPArrayDesc ArrayDesc, int Dims, int CurDim, void *
DataP);
```

Description

This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.

66.1.3.4.15 TSOAPDomConv.ConvertSoapToNativeData Method

Converts the SOAP representation of a value to a pointer to the corresponding native representation.

Pascal

```
procedure ConvertSoapToNativeData(DataP: Pointer; TypeInfo: PTypeInfo; Context: TDataContext;
RootNode: IXMLNode; Node: IXMLNode; Translate: Boolean; ByRef: Boolean; NumIndirect: Integer);
dynamic;
```

C++

```
__fastcall ConvertSoapToNativeData(void * DataP, PTypeInfo TypeInfo, TDataContext Context,
IXMLNode RootNode, IXMLNode Node, Boolean Translate, Boolean ByRef, int NumIndirect);
```

Description

ConvertSoapToNativeData performs the conversion from a DOM node that represents a SOAP encoding of a value to the corresponding native representation.

DataP is a pointer to the memory allocated to hold the native representation of the value, and points to the results of the conversion.

TypeInfo is the typeinfo pointer that describes the native type that will represent the value.

Context is the TDataContext (see page 4354) object that stores the value.

RootNode is the node in the SOAP packet that acts as a parent to any elements to which there are multiple references.

Node is the node in the SOAP packet whose value is to be translated.

Translate indicates whether Node represents a value to be translated, as opposed to a reference to value that is found elsewhere in the SOAP packet.

ByRef indicates whether the node corresponds to a parameter that is passed by reference.

NumIndirect indicates the number of levels of indirection by which DataP points to its value.

See Also

ConvertNativeDataToSoap (see page 6185)

Options (see page 6201)

66.1.3.4.16 TSOAPDomConv.ConvertSOAPToObject Method

Pascal

```
function ConvertSOAPToObject(RootNode: IXMLNode; Node: IXMLNode; AClass: TClass; const URI:
InvString; const TypeName: InvString; ObjP: Pointer; NumIndirect: Integer): TObject;
```

C++

```
__fastcall TObject * ConvertSOAPToObject(IXMLNode RootNode, IXMLNode Node, TClass AClass,
const InvString URI, const InvString TypeName, void * ObjP, int NumIndirect);
```

Description

This is ConvertSOAPToObject, a member of class TSOAPDomConv.

66.1.3.4.17 TSOAPDomConv.ConvertSoapToVariant Method

Pascal

```
procedure ConvertSoapToVariant(Node: IXMLNode; InvData: Pointer);
```

C++

```
__fastcall ConvertSoapToVariant(IXMLNode Node, void * InvData);
```

Description

This is ConvertSoapToVariant, a member of class TSOAPDomConv.

66.1.3.4.18 TSOAPDomConv.ConvertVariantToSoap Method

Pascal

```
procedure ConvertVariantToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: PVarData; NumIndirect: Integer; V: Variant; UseVariant: Boolean);
```

C++

```
__fastcall ConvertVariantToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, PVarData P, int NumIndirect, Variant V, Boolean UseVariant);
```

Description

This is ConvertVariantToSoap, a member of class TSOAPDomConv.

66.1.3.4.19 TSOAPDomConv.CreateMultiRefNode Method

Pascal

```
function CreateMultiRefNode(RootNode: IXMLNode; const Name: InvString; const ID: InvString):
IXMLNode;
```

C++

```
__fastcall IXMLNode CreateMultiRefNode(IXMLNode RootNode, const InvString Name, const
InvString ID);
```

Description

This is CreateMultiRefNode, a member of class TSOAPDomConv.

66.1.3.4.20 TSOAPDomConv.CreateNULLNode Method

Pascal

```
function CreateNULLNode(RootNode: IXMLNode; ParentNode: IXMLNode; const Name: InvString;
UseParentNode: Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNULLNode(IXMLNode RootNode, IXMLNode ParentNode, const InvString
Name, Boolean UseParentNode = False);
```

Description

This is CreateNULLNode, a member of class TSOAPDomConv.

66.1.3.4.21 TSOAPDomConv.CreateObjectNode Method

Pascal

```
function CreateObjectNode(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
Name: InvString; const URI: InvString; ObjConvOpts: TObjectConvertOptions): InvString;
```

C++

```
__fastcall InvString CreateObjectNode(TObject * Instance, IXMLNode RootNode, IXMLNode
ParentNode, const InvString Name, const InvString URI, TObjectConvertOptions ObjConvOpts);
```

Description

This is CreateObjectNode, a member of class TSOAPDomConv.

66.1.3.4.22 TSOAPDomConv.CreateScalarNodeXS Method

Pascal

```
function CreateScalarNodeXS(RootNode: IXMLNode; ParentNode: IXMLNode; const NodeName:
InvString; const URI: InvString; const TypeName: InvString; const Value: InvString; GenPre:
Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateScalarNodeXS(IXMLNode RootNode, IXMLNode ParentNode, const InvString
NodeName, const InvString URI, const InvString TypeName, const InvString Value, Boolean GenPre
= False);
```

Description

This is CreateScalarNodeXS, a member of class TSOAPDomConv.

66.1.3.4.23 TSOAPDomConv.CreateTypedNode Method

Pascal

```
function CreateTypedNode(RootNode: IXMLNode; ParentNode: IXMLNode; const NodeName: InvString;
const URI: InvString; TypeName: InvString; GenPre: Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateTypedNode(IXMLNode RootNode, IXMLNode ParentNode, const InvString
NodeName, const InvString URI, InvString TypeName, Boolean GenPre = False);
```

Description

This is CreateTypedNode, a member of class TSOAPDomConv.

66.1.3.4.24 TSOAPDomConv.FinalizeMultiRefNodes Method

Pascal

```
procedure FinalizeMultiRefNodes;
```

C++

```
__fastcall FinalizeMultiRefNodes();
```

Description

This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.

66.1.3.4.25 TSOAPDomConv.FindAttachment Method

Returns a TSOAPAttachment instance that represent the attachment with a specified ID.

Pascal

```
function FindAttachment(const AContentId: string): TSOAPAttachment;
```

C++

```
__fastcall TSOAPAttachment FindAttachment(const UnicodeString AContentId);
```

Description

FindAttachment locates the attachment that is identified by a specified content ID (HREF).

AContentId is the unique identifier for the attachment that was supplied to the AddAttachment method when the attachment was first added to the multipart form.

FindAttachment returns a TSOAPAttachment instance that represents the attachment.

See Also

AddAttachment (see page 4462)

66.1.3.4.26 TSOAPDomConv.FindMultiRefNodeByID Method

Pascal

```
function FindMultiRefNodeByID(const ID: string): Pointer;
```

C++

```
__fastcall void * FindMultiRefNodeByID(const UnicodeString ID);
```

Description

This is FindMultiRefNodeByID, a member of class TSOAPDomConv.

66.1.3.4.27 TSOAPDomConv.FindMultiRefNodeByInstance Method

Pascal

```
function FindMultiRefNodeByInstance(Instance: Pointer): string;
```

C++

```
__fastcall UnicodeString FindMultiRefNodeByInstance(void * Instance);
```

Description

This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.

66.1.3.4.28 TSOAPDomConv.FindNodeByHREF Method

Pascal

```
function FindNodeByHREF(RootNode: IXMLNode; const HREF: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNodeByHREF(IXMLNode RootNode, const InvString HREF);
```

Description

This is FindNodeByHREF, a member of class TSOAPDomConv.

66.1.3.4.29 TSOAPDomConv.FindPrefixForURI Method

Pascal

```
function FindPrefixForURI(RootNode: IXMLNode; Node: IXMLNode; const URI: InvString;
DeclIfNone: Boolean = False): InvString;
```

C++

```
__fastcall InvString FindPrefixForURI(IXMLNode RootNode, IXMLNode Node, const InvString URI,
Boolean DeclIfNone = False);
```

Description

This is FindPrefixForURI, a member of class TSOAPDomConv.

66.1.3.4.30 TSOAPDomConv.GetDataNode Method

Pascal

```
function GetDataNode(RootNode: IXMLNode; Node: IXMLNode; var ID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDataNode(IXMLNode RootNode, IXMLNode Node, InvString ID);
```

Description

This is GetDataNode, a member of class TSOAPDomConv.

66.1.3.4.31 TSOAPDomConv.GetElementType Method

Pascal

```
function GetElementType(Node: IXMLNode; var TypeURI: InvString; var TypeName: InvString):
Boolean;
```

C++

```
__fastcall Boolean GetElementType(IXMLNode Node, InvString TypeURI, InvString TypeName);
```

Description

This is GetElementType, a member of class TSOAPDomConv.

66.1.3.4.32 TSOAPDomConv.GetNewID Method

Pascal

```
function GetNewID: string;
```

C++

```
__fastcall UnicodeString GetNewID();
```

Description

This is GetNewID, a member of class TSOAPDomConv.

66.1.3.4.33 TSOAPDomConv.GetNodeAsText Method

Pascal

```
function GetNodeAsText(Node: IXMLNode): InvString;
```

C++

```
__fastcall InvString GetNodeAsText(IXMLNode Node);
```

Description

This is GetNodeAsText, a member of class TSOAPDomConv.

66.1.3.4.34 TSOAPDomConv.GetObjectPropAsText Method

Pascal

```
function GetObjectPropAsText(Instance: TObject; PropInfo: PPropInfo): InvString;
```

C++

```
__fastcall InvString GetObjectPropAsText(TObject * Instance, PPropInfo PropInfo);
```

Description

This is GetObjectPropAsText, a member of class TSOAPDomConv.

66.1.3.4.35 TSOAPDomConv.GetOptions Method

Pascal

```
function GetOptions: TSOAPConvertOptions;
```

C++

```
__fastcall TSOAPConvertOptions GetOptions();
```

Description

This is GetOptions, a member of class TSOAPDomConv.

66.1.3.4.36 TSOAPDomConv.GetTypeBySchemaNS Method

Pascal

```
function GetTypeBySchemaNS(Node: IXMLNode; const URI: InvString): Variant;
```

C++

```
__fastcall Variant GetTypeBySchemaNS(IXMLNode Node, const InvString URI);
```

Description

This is GetTypeBySchemaNS, a member of class TSOAPDomConv.

66.1.3.4.37 TSOAPDomConv.InitObjectFromSOAP Method

Initializes an instance of a remotable object to reflect the values in its SOAP representation.

Pascal

```
procedure InitObjectFromSOAP(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall InitObjectFromSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

InitObjectFromSOAP assigns values to the published properties of a remotable object that represents a node from the SOAP envelope.

Instance is the remotable object to initialize from its SOAP representation.

RootNode is an ancestor node of the node for the remotable object that can contain information needed for initializing the remotable object.

Node is the node in the SOAP envelope that represents the remotable object.

See Also

ObjInstanceToSOAP (see page 4463)

66.1.3.4.38 TSOAPDomConv.IsNodeAVarArray Method

Pascal

```
function IsNodeAVarArray(const Node: IXMLNode; var VT: TVarType): Boolean;
```

C++

```
__fastcall Boolean IsNodeAVarArray(const IXMLNode Node, TVarType VT);
```

Description

This is IsNodeAVarArray, a member of class TSOAPDomConv.

66.1.3.4.39 TSOAPDomConv.IsObjectWriting Method

Pascal

```
function IsObjectWriting(Instance: TObject): Boolean;
```

C++

```
__fastcall Boolean IsObjectWriting(TObject * Instance);
```

Description

This is IsObjectWriting, a member of class TSOAPDomConv.

66.1.3.4.40 TSOAPDomConv.LoadObject Method

Pascal

```
procedure LoadObject(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall LoadObject(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

This is LoadObject, a member of class TSOAPDomConv.

66.1.3.4.41 MakeArrayNode Method

This is the overview for the MakeArrayNode method overload.

Overload List

	Name	Description
	TSOAPDomConv.MakeArrayNode (IXMLNode, IXMLNode, InvString, InvString, InvString, array of Integer) (see page 6194)	This is MakeArrayNode, a member of class TSOAPDomConv.
	TSOAPDomConv.MakeArrayNode (IXMLNode, IXMLNode, InvString, InvString, InvString, Integer, Integer) (see page 6194)	This is MakeArrayNode, a member of class TSOAPDomConv.

66.1.3.4.41.1 TSOAPDomConv.MakeArrayNode Method (IXMLNode, IXMLNode, InvString, InvString, InvString, Integer, Integer)

Pascal

```
function MakeArrayNode(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; const URI:
InvString; const TypeName: InvString; Dim: Integer; Len: Integer): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode MakeArrayNode(IXMLNode RootNode, IXMLNode Node, const InvString Name,
const InvString URI, const InvString TypeName, int Dim, int Len);
```

Description

This is MakeArrayNode, a member of class TSOAPDomConv.

66.1.3.4.41.2 TSOAPDomConv.MakeArrayNode Method (IXMLNode, IXMLNode, InvString, InvString, InvString, array of Integer)

Pascal

```
function MakeArrayNode(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; const URI:
InvString; const TypeName: InvString; Indices: array of Integer): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode MakeArrayNode(IXMLNode RootNode, IXMLNode Node, const InvString Name,
const InvString URI, const InvString TypeName, array of Integer Indices);
```

Description

This is MakeArrayNode, a member of class TSOAPDomConv.

66.1.3.4.42 TSOAPDomConv.MultiRefObject Method

Pascal

```
function MultiRefObject(Cls: TClass): Boolean;
```

C++

```
__fastcall Boolean MultiRefObject(TClass Cls);
```

Description

This is MultiRefObject, a member of class TSOAPDomConv.

66.1.3.4.43 TSOAPDomConv.NodeIsNULL Method

Pascal

```
function NodeIsNULL(Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean NodeIsNULL(IXMLNode Node);
```

Description

This is NodeIsNULL, a member of class TSOAPDomConv.

66.1.3.4.44 TSOAPDomConv.ObjectMemberNoShow Method

Pascal

```
procedure ObjectMemberNoShow(const ClassName: string; const MemberName: string);
```

C++

```
__fastcall ObjectMemberNoShow(const UnicodeString ClassName, const UnicodeString MemberName);
```

Description

This is ObjectMemberNoShow, a member of class TSOAPDomConv.

66.1.3.4.45 TSOAPDomConv.ObjInstanceToSOAP Method

Returns the SOAP representation of a remotable object.

Pascal

```
function ObjInstanceToSOAP(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
NodeName: InvString; const NodeNamespace: InvString; ObjConvOpts: TObjectConvertOptions; out
RefID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode ObjInstanceToSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode
ParentNode, const InvString NodeName, const InvString NodeNamespace, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjInstanceToSOAP adds a node to represent the value of a remotable object.

Instance is the remotable object to encode as its SOAP representation.

RootNode is an ancestor of the node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjInstanceToSOAP generates.

NodeName is the name of the node that is generated.

NodeNamespace is the URI for the namespace in which NodeName is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjInstanceToSOAP returns the XML for the newly generated node.

See Also

InitObjectFromSOAP (see page 4463)

66.1.3.4.46 TSOAPDomConv.ReadHeader Method

Pascal

```
procedure ReadHeader(const EnvNode: IXMLNode; const HdrNode: IXMLNode; Headers: THeaderList);
```

C++

```
__fastcall ReadHeader(const IXMLNode EnvNode, const IXMLNode HdrNode, THeaderList Headers);
```

Description

This is ReadHeader, a member of class TSOAPDomConv.

66.1.3.4.47 TSOAPDomConv.ReadRectDynArray Method

Pascal

```
procedure ReadRectDynArray(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Dims: Integer;
P: Pointer; CurElem: Integer);
```

C++

```
__fastcall ReadRectDynArray(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Dims, void *
P, int CurElem);
```

Description

This is ReadRectDynArray, a member of class TSOAPDomConv.

66.1.3.4.48 TSOAPDomConv.ReadRectDynArrayElem Method

Pascal

```
procedure ReadRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Size:
Integer; Dim: Integer; P: Pointer; var CurElem: Integer);
```

C++

```
__fastcall ReadRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Size,
int Dim, void * P, int CurElem);
```

Description

This is ReadRectDynArrayElem, a member of class TSOAPDomConv.

66.1.3.4.49 TSOAPDomConv.ReadRow Method

Pascal

```
procedure ReadRow(RootNode: IXMLNode; Node: IXMLNode; var CurElem: Integer; Size: Integer; P:
Pointer; Info: PTypeInfo);
```

C++

```
__fastcall ReadRow(IXMLNode RootNode, IXMLNode Node, int CurElem, int Size, void * P,
PTypeInfo Info);
```

Description

This is ReadRow, a member of class TSOAPDomConv.

66.1.3.4.50 TSOAPDomConv.ReadVarArrayDim Method

Pascal

```
function ReadVarArrayDim(Node: IXMLNode; IsVarVArray: Boolean = False; VT: TVarType = 0):
Variant;
```

C++

```
__fastcall Variant ReadVarArrayDim(IXMLNode Node, Boolean IsVarVArray = False, TVarType VT =
0);
```

Description

This is ReadVarArrayDim, a member of class TSOAPDomConv.

66.1.3.4.51 TSOAPDomConv.ReadVariant Method

Pascal

```
procedure ReadVariant(Node: IXMLNode; P: Pointer);
```

C++

```
__fastcall ReadVariant(IXMLNode Node, void * P);
```

Description

This is ReadVariant, a member of class TSOAPDomConv.

66.1.3.4.52 TSOAPDomConv.RemoveObjectAsWriting Method

Pascal

```
procedure RemoveObjectAsWriting(Instance: TObject);
```

C++

```
__fastcall RemoveObjectAsWriting(TObject * Instance);
```

Description

This is RemoveObjectAsWriting, a member of class TSOAPDomConv.

66.1.3.4.53 TSOAPDomConv.ResetMultiRef Method

Pascal

```
procedure ResetMultiRef;
```

C++

```
__fastcall ResetMultiRef();
```

Description

This is ResetMultiRef, a member of class TSOAPDomConv.

66.1.3.4.54 SerializationOptions Method

This is the overview for the SerializationOptions method overload.

Overload List

	Name	Description
	TSOAPDomConv.SerializationOptions (PTypeInfo) (see page 6198)	This is SerializationOptions, a member of class TSOAPDomConv.
	TSOAPDomConv.SerializationOptions (TClass) (see page 6198)	This is SerializationOptions, a member of class TSOAPDomConv.
	TSOAPDomConv.SerializationOptions (TObject) (see page 6198)	This is SerializationOptions, a member of class TSOAPDomConv.

66.1.3.4.54.1 TSOAPDomConv.SerializationOptions Method (PTypeInfo)

Pascal

```
function SerializationOptions(ATypeInfo: PTypeInfo): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(PTypeInfo ATypeInfo);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

66.1.3.4.54.2 TSOAPDomConv.SerializationOptions Method (TClass)

Pascal

```
function SerializationOptions(Cls: TClass): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(TClass Cls);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

66.1.3.4.54.3 TSOAPDomConv.SerializationOptions Method (TObject)

Pascal

```
function SerializationOptions(Obj: TObject): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(TObject * Obj);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

66.1.3.4.55 TSOAPDomConv.SetNodeType Method

Pascal

```
procedure SetNodeType(RootNode: IXMLNode; InstNode: IXMLNode; const ElemURI: InvString; const
TypeName: InvString);
```

C++

```
__fastcall SetNodeType(IXMLNode RootNode, IXMLNode InstNode, const InvString ElemURI, const
InvString TypeName);
```

Description

This is SetNodeType, a member of class TSOAPDomConv.

66.1.3.4.56 TSOAPDomConv.SetObjectPropFromText Method

Pascal

```
procedure SetObjectPropFromText(Instance: TObject; PropInfo: PPropInfo; const SoapData:
InvString);
```

C++

```
__fastcall SetObjectPropFromText(TObject * Instance, PPropInfo PropInfo, const InvString
SoapData);
```

Description

This is SetObjectPropFromText, a member of class TSOAPDomConv.

66.1.3.4.57 TSOAPDomConv.SetOptions Method

Pascal

```
procedure SetOptions(const Value: TSOAPConvertOptions);
```

C++

```
__fastcall SetOptions(const TSOAPConvertOptions Value);
```

Description

This is SetOptions, a member of class TSOAPDomConv.

66.1.3.4.58 TSOAPDomConv.UnhandledNode Method

Pascal

```
procedure UnhandledNode(const Name: string; NodeXML: InvString);
```

C++

```
__fastcall UnhandledNode(const UnicodeString Name, InvString NodeXML);
```

Description

This is UnhandledNode, a member of class TSOAPDomConv.

66.1.3.4.59 TSOAPDomConv.WriteHeader Method

Pascal

```
procedure WriteHeader(const Header: TObject; RootNode: IXMLNode; ParentNode: IXMLNode);
```

C++

```
__fastcall WriteHeader(const TObject * Header, IXMLNode RootNode, IXMLNode ParentNode);
```

Description

This is WriteHeader, a member of class TSOAPDomConv.

66.1.3.4.60 TSOAPDomConv.WriteNonRectDynArray Method

Pascal

```
procedure WriteNonRectDynArray(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; const URI: InvString; const TypeName: InvString; P: Pointer; Dim: Integer);
```

C++

```
__fastcall WriteNonRectDynArray(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, const InvString URI, const InvString TypeName, void * P, int Dim);
```

Description

This is WriteNonRectDynArray, a member of class TSOAPDomConv.

66.1.3.4.61 TSOAPDomConv.WriteNonRectDynArrayElem Method

Pascal

```
function WriteNonRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; const
URI: InvString; const TypeName: InvString; P: Pointer; Dim: Integer): Integer;
```

C++

```
__fastcall int WriteNonRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info,
const InvString URI, const InvString TypeName, void * P, int Dim);
```

Description

This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.

66.1.3.4.62 TSOAPDomConv.WriteRectDynArray Method

Pascal

```
procedure WriteRectDynArray(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Dims:
Integer; P: Pointer; const TypeName: InvString);
```

C++

```
__fastcall WriteRectDynArray(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Dims, void
* P, const InvString TypeName);
```

Description

This is WriteRectDynArray, a member of class TSOAPDomConv.

66.1.3.4.63 TSOAPDomConv.WriteRectDynArrayElem Method

Pascal

```
procedure WriteRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Size:
Integer; Dim: Integer; P: Pointer; const TypeName: InvString);
```

C++

```
__fastcall WriteRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Size,
int Dim, void * P, const InvString TypeName);
```

Description

This is WriteRectDynArrayElem, a member of class TSOAPDomConv.

66.1.3.4.64 TSOAPDomConv.WriteVarArray Method

Pascal

```
procedure WriteVarArray(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V: Variant);
```

C++

```
__fastcall WriteVarArray(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant V);
```

Description

This is WriteVarArray, a member of class TSOAPDomConv.

66.1.3.4.65 TSOAPDomConv.WriteVarArrayAsB64 Method

Pascal

```
procedure WriteVarArrayAsB64(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V:
Variant);
```

C++

```
__fastcall WriteVarArrayAsB64(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant
V);
```

Description

This is WriteVarArrayAsB64, a member of class TSOAPDomConv.

66.1.3.4.66 TSOAPDomConv.WriteVariant Method

Pascal

```
procedure WriteVariant(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V: Variant);
```

C++

```
__fastcall WriteVariant(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant V);
```

Description

This is WriteVariant, a member of class TSOAPDomConv.

66.1.3.5 TSOAPDomConv Properties

The properties of the TSOAPDomConv class are listed here.

Published Properties

	Name	Description
	Options (see page 6201)	Controls how the TSOAPDomConv (see page 6178) object performs certain types of conversions.

66.1.3.5.1 TSOAPDomConv.Options Property

Controls how the TSOAPDomConv (see page 6178) object performs certain types of conversions.

Pascal

```
property Options: TSOAPConvertOptions;
```

C++

```
__property TSOAPConvertOptions Options;
```

Description

Set Options to indicate how you want the TSOAPDomConv (see page 6178) object to handle various types of conversions. This set can include zero or more of the TSOAPConvertOption (see page 6162) flags.

See Also

ConvertNativeDataToSoap (see page 6185)

ClassOptions (see page 4417)

ConvertSoapToNativeData (see page 6187)

66.1.3.6 TSOAPDomConv Events

The events of the TSOAPDomConv class are listed here.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 6202)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 6202)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

66.1.3.6.1 TSOAPDomConv.OnMemberDataNotReceived Event

Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.

Pascal

```
property OnMemberDataNotReceived: TMemberDataNotReceivedEvent;
```

C++

```
__property TMemberDataNotReceivedEvent OnMemberDataNotReceived;
```

Description

Write an OnMemberDataNotReceived to detect cases where the converter does not receive the information it excepts When a method on an invokable interface requires the user to send or receive OnMemberDataNotReceived, these for deserializing a remotable type (a TRemotable (see page 4408) descendant). This event is especially useful during debugging. Typically, after detecting a problem using OnMemberDataNotReceived, the problem is then handled by changing the code that declares or registers the remotable type.

See Also

TRemotable (see page 4408)

66.1.3.6.2 TSOAPDomConv.OnUnhandledNode Event

Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Pascal

```
property OnUnhandledNode: TUnhandledNodeEvent;
```

C++

```
__property TUnhandledNodeEvent OnUnhandledNode;
```

Description

Write an OnUnhandledNode to detect and, possibly, fix cases where the converter can't correctly generate the native code corresponding to a node in a SOAP packet.

66.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetOrdPropEx (see page 6203)	This is function OPToSOAPDomConv.GetOrdPropEx.

66.2.1 OPToSOAPDomConv.GetOrdPropEx Function

Pascal

```
function GetOrdPropEx(Instance: TObject; PropInfo: PPropInfo): Longint;
```

C++

```
Longint GetOrdPropEx(TObject * Instance, PPropInfo PropInfo);
```

File

OPToSOAPDomConv

Description

This is function OPToSOAPDomConv.GetOrdPropEx.

66.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	ConvNodeState (see page 6204)	This is record OPToSOAPDomConv.ConvNodeState.

Records

	Name	Description
	TMultiRefNodeElem (see page 6204)	This is record OPToSOAPDomConv.TMultiRefNodeElem.
	TMultiRefNodeMapElem (see page 6205)	This is record OPToSOAPDomConv.TMultiRefNodeMapElem.
	TSOAPArrayElemDesc (see page 6205)	This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

66.3.1 OPToSOAPDomConv.ConvNodeState Enumeration

Pascal

```
ConvNodeState = (
  nsClientSend,
  nsServerReceive,
  nsServerSend,
  nsClientReceive
);
```

C++

```
enum ConvNodeState {
  nsClientSend,
  nsServerReceive,
  nsServerSend,
  nsClientReceive
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.ConvNodeState.

66.3.2 OPToSOAPDomConv.TMultiRefNodeElem Record

Pascal

```
TMultiRefNodeElem = record
  Node: IXMLNode;
  MultiRefChildren: TXMLNodeArray;
end;
```

C++

```
struct TMultiRefNodeElem {
  IXMLNode Node;
  TXMLNodeArray MultiRefChildren;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TMultiRefNodeElem.

66.3.3 OPToSOAPDomConv.TMultiRefNodeMapElem Record

Pascal

```
TMultiRefNodeMapElem = record
  Instance: Pointer;
  ID: string;
end;
```

C++

```
struct TMultiRefNodeMapElem {
  void * Instance;
  UnicodeString ID;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TMultiRefNodeMapElem.

66.3.4 OPToSOAPDomConv.TSOAPArrayElemDesc Record

Pascal

```
TSOAPArrayElemDesc = record
  MultiDim: Boolean;
  Dims: TIntegerDynArray;
end;
```

C++

```
struct TSOAPArrayElemDesc {
  Boolean MultiDim;
  TIntegerDynArray Dims;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

66.4 Types

The following table lists types in this documentation.

Types

Name	Description
TMemberDataNotReceivedEvent (see page 6206)	This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.
TMultiRefNodeMap (see page 6206)	This is type OPToSOAPDomConv.TMultiRefNodeMap.
TMultiRefNodes (see page 6207)	This is type OPToSOAPDomConv.TMultiRefNodes.
TSOAPArrayDesc (see page 6207)	This is type OPToSOAPDomConv.TSOAPArrayDesc.
TUnhandledNodeEvent (see page 6207)	This is type OPToSOAPDomConv.TUnhandledNodeEvent.
TXMLNodeArray (see page 6208)	This is type OPToSOAPDomConv.TXMLNodeArray.

66.4.1 OPToSOAPDomConv.TMemberDataNotReceivedEvent Type

Pascal

```
TMemberDataNotReceivedEvent = procedure (const ClassName: string; const Member: string) of object;
```

C++

```
(const ClassName: string; const Member: string) ( TMemberDataNotReceivedEvent)();
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.

66.4.2 OPToSOAPDomConv.TMultiRefNodeMap Type

Pascal

```
TMultiRefNodeMap = array of TMultiRefNodeMapElem;
```

C++

```
array of TMultiRefNodeMapElem TMultiRefNodeMap;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMultiRefNodeMap.

66.4.3 OPToSOAPDomConv.TMultiRefNodes Type

Pascal

```
TMultiRefNodes = array of TMultiRefNodeElem;
```

C++

```
array of TMultiRefNodeElem TMultiRefNodes;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMultiRefNodes.

66.4.4 OPToSOAPDomConv.TSOAPArrayDesc Type

Pascal

```
TSOAPArrayDesc = array of TSOAPArrayElemDesc;
```

C++

```
array of TSOAPArrayElemDesc TSOAPArrayDesc;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TSOAPArrayDesc.

66.4.5 OPToSOAPDomConv.TUnhandledNodeEvent Type

Pascal

```
TUnhandledNodeEvent = procedure (const Name: string; NodeXML: InvString) of object;
```

C++

```
(const Name: string; NodeXML: InvString) ( TUnhandledNodeEvent)();
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TUnhandledNodeEvent.

66.4.6 OPToSOAPDomConv.TXMLNodeArray Type

Pascal

```
TXMLNodeArray = array of IXMLNode;
```

C++

```
array of IXMLNode TXMLNodeArray;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TXMLNodeArray.

66.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefArrayElemName (see page 6209)	This is variable OPToSOAPDomConv.DefArrayElemName.

66.5.1 OPToSOAPDomConv.DefArrayElemName Variable

Pascal

```
DefArrayElemName: string = 'item';
```

C++

```
UnicodeString DefArrayElemName = 'item';
```

File

OPToSOAPDomConv

Description

This is variable OPToSOAPDomConv.DefArrayElemName.

66.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SVarArrayType (see page 6210)	This is constant OPToSOAPDomConv.SVarArrayType.

66.6.1 OPToSOAPDomConv.SVarArrayType Constant

Pascal

```
SVarArrayType = 'VarArrayType';
```

C++

```
SVarArrayType = 'VarArrayType';
```

File

OPToSOAPDomConv

Description

This is constant OPToSOAPDomConv.SVarArrayType.

67 OrbPas Namespace

This is namespace OrbPas.

Topics

Name	Description
Interfaces (see page 6213)	The following table lists interfaces in this documentation.
Functions (see page 6241)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6246)	The following table lists structs, records, enums in this documentation.
Types (see page 6248)	The following table lists types in this documentation.

Functions

	Name	Description
	AnyToVariant (see page 6241)	This is function OrbPas.AnyToVariant.
	BindStub (see page 6241)	This is function OrbPas.BindStub.
	CorbaAnyType (see page 6242)	This is function OrbPas.CorbaAnyType.
	CorbaDuplicateAny (see page 6242)	This is function OrbPas.CorbaDuplicateAny.
	CorbaReleaseAny (see page 6242)	This is function OrbPas.CorbaReleaseAny.
	CorbaStringFree (see page 6243)	This is function OrbPas.CorbaStringFree.
	CorbaWStringFree (see page 6243)	This is function OrbPas.CorbaWStringFree.
	CreateSkeleton (see page 6243)	This is function OrbPas.CreateSkeleton.
	CreateUserException (see page 6244)	This is function OrbPas.CreateUserException.
	InitORB (see page 6244)	This is function OrbPas.InitORB.
	RegisterUserException (see page 6244)	This is function OrbPas.RegisterUserException.
	SequenceToVariant (see page 6245)	This is function OrbPas.SequenceToVariant.
	UnRegisterUserException (see page 6245)	This is function OrbPas.UnRegisterUserException.
	VariantToAny (see page 6245)	This is function OrbPas.VariantToAny.

Interfaces

	Name	Description
	IBOA (see page 6213)	This is class OrbPas.IBOA.
	ICorbaObj (see page 6215)	This is class OrbPas.ICorbaObj.
	IMarshalInBuffer (see page 6216)	This is class OrbPas.IMarshalInBuffer.
	IMarshalOutBuffer (see page 6220)	This is class OrbPas.IMarshalOutBuffer.
	IORB (see page 6225)	This is class OrbPas.IORB.
	ISkeleton (see page 6229)	This is class OrbPas.ISkeleton.
	ISkeletonObject (see page 6230)	This is class OrbPas.ISkeletonObject.

	IStub (see page 6232)	This is class OrbPas.IStub.
	IStubObject (see page 6235)	This is class OrbPas.IStubObject.
	ITypeCode (see page 6236)	This is class OrbPas.ITypeCode.

Structs, Records, Enums

	Name	Description
	TCKind (see page 6246)	This is record OrbPas.TCKind.
	TStructMember (see page 6247)	This is record OrbPas.TStructMember.

Types

Name	Description
CorbaBoolean (see page 6248)	This is type OrbPas.CorbaBoolean.
CorbaULong (see page 6248)	This is type OrbPas.CorbaULong.
PCorbaAny (see page 6249)	This is type OrbPas.PCorbaAny.
PCorbaInterfaceDef (see page 6249)	This is type OrbPas.PCorbaInterfaceDef.
PCorbaObject (see page 6249)	This is type OrbPas.PCorbaObject.
PCorbaOrb (see page 6250)	This is type OrbPas.PCorbaOrb.
PExceptionDescription (see page 6250)	This is type OrbPas.PExceptionDescription.
PMarshalInbuffer (see page 6250)	This is type OrbPas.PMarshalInbuffer.
PMarshalOutbuffer (see page 6251)	This is type OrbPas.PMarshalOutbuffer.
PUserExceptionProxy (see page 6251)	This is type OrbPas.PUserExceptionProxy.
TArgv (see page 6251)	This is type OrbPas.TArgv.
TCopyUserExceptionProc (see page 6252)	This is type OrbPas.TCopyUserExceptionProc.
TStructMembers (see page 6252)	This is type OrbPas.TStructMembers.
TThrowUserExceptionProc (see page 6252)	This is type OrbPas.TThrowUserExceptionProc.
TUserExceptionFactoryProc (see page 6253)	This is type OrbPas.TUserExceptionFactoryProc.

67.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IBOA (see page 6213)	This is class OrbPas.IBOA.
	ICorbaObj (see page 6215)	This is class OrbPas.ICorbaObj.
	IMarshalInBuffer (see page 6216)	This is class OrbPas.IMarshalInBuffer.
	IMarshalOutBuffer (see page 6220)	This is class OrbPas.IMarshalOutBuffer.
	IORB (see page 6225)	This is class OrbPas.IORB.
	ISkeleton (see page 6229)	This is class OrbPas.ISkeleton.
	ISkeletonObject (see page 6230)	This is class OrbPas.ISkeletonObject.
	IStub (see page 6232)	This is class OrbPas.IStub.
	IStubObject (see page 6235)	This is class OrbPas.IStubObject.
	ITypeCode (see page 6236)	This is class OrbPas.ITypeCode.

67.1.1 IBOA Interface

Class Hierarchy

OrbPas.IBOA

Pascal

```
IBOA = interface;
```

C++

```
__interface IBOA;
```

File

OrbPas

Description

This is class OrbPas.IBOA.

67.1.1.1 IBOA Members

The following tables list the members exposed by IBOA.

Public Methods

	Name	Description
	Deactivate (see page 6214)	This is Deactivate, a member of class IBOA.
	GetPrincipal (see page 6214)	This is GetPrincipal, a member of class IBOA.
	GetPrincipalLength (see page 6214)	This is GetPrincipalLength, a member of class IBOA.
	ImplIsReady (see page 6214)	This is ImplIsReady, a member of class IBOA.
	ObjIsReady (see page 6215)	This is ObjIsReady, a member of class IBOA.

67.1.1.2 IBOA Methods

The methods of the IBOA class are listed here.

Public Methods

	Name	Description
	Deactivate (see page 6214)	This is Deactivate, a member of class IBOA.
	GetPrincipal (see page 6214)	This is GetPrincipal, a member of class IBOA.
	GetPrincipalLength (see page 6214)	This is GetPrincipalLength, a member of class IBOA.
	ImplIsReady (see page 6214)	This is ImplIsReady, a member of class IBOA.
	ObjIsReady (see page 6215)	This is ObjIsReady, a member of class IBOA.

67.1.1.2.1 IBOA.Deactivate Method

Pascal

```
procedure Deactivate(const Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall Deactivate(const ISkeleton Skeleton);
```

Description

This is Deactivate, a member of class IBOA.

67.1.1.2.2 IBOA.GetPrincipal Method

Pascal

```
function GetPrincipal(Skeleton: ISkeleton; Bytes: Pointer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetPrincipal(ISkeleton Skeleton, void * Bytes);
```

Description

This is GetPrincipal, a member of class IBOA.

67.1.1.2.3 IBOA.GetPrincipalLength Method

Pascal

```
function GetPrincipalLength(Skeleton: ISkeleton): Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetPrincipalLength(ISkeleton Skeleton);
```

Description

This is GetPrincipalLength, a member of class IBOA.

67.1.1.2.4 IBOA.ImplIsReady Method

Pascal

```
procedure ImplIsReady; stdcall;
```

C++

```
__fastcall __stdcall ImplIsReady();
```

Description

This is ImplIsReady, a member of class IBOA.

67.1.1.2.5 IBOA.ObjIsReady Method

Pascal

```
procedure ObjIsReady(const Skeleton: ISkeleton); stdcall;
```

C++

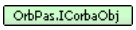
```
__fastcall __stdcall ObjIsReady(const ISkeleton Skeleton);
```

Description

This is ObjIsReady, a member of class IBOA.

67.1.2 ICorbaObj Interface

Class Hierarchy



Pascal

```
ICorbaObj = interface;
```

C++

```
__interface ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.ICorbaObj.

67.1.2.1 ICorbaObj Members

The following tables list the members exposed by ICorbaObj.

Public Methods

	Name	Description
	CorbaObject (see page 6216)	This is CorbaObject, a member of class ICorbaObj.
	IsLocal (see page 6216)	This is IsLocal, a member of class ICorbaObj.

67.1.2.2 ICorbaObj Methods

The methods of the ICorbaObj class are listed here.

Public Methods

	Name	Description
	CorbaObject (see page 6216)	This is CorbaObject, a member of class ICorbaObj.
	IsLocal (see page 6216)	This is IsLocal, a member of class ICorbaObj.

67.1.2.2.1 ICorbaObj.CorbaObject Method

Pascal

```
function CorbaObject: PCorbaObject; stdcall;
```

C++

```
__fastcall __stdcall PCorbaObject CorbaObject();
```

Description

This is CorbaObject, a member of class ICorbaObj.

67.1.2.2.2 ICorbaObj.IsLocal Method

Pascal

```
function IsLocal: CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean IsLocal();
```

Description

This is IsLocal, a member of class ICorbaObj.

67.1.3 IMarshalInBuffer Interface

Class Hierarchy

OrbPas.IMarshalInBuffer

Pascal

```
IMarshalInBuffer = interface;
```

C++

```
__interface IMarshalInBuffer;
```

File

OrbPas

Description

This is class OrbPas.IMarshalInBuffer.

67.1.3.1 IMarshalInBuffer Members

The following tables list the members exposed by IMarshalInBuffer.

Public Methods

	Name	Description
	Buffer (see page 6217)	This is Buffer, a member of class IMarshalInBuffer.
	GetAny (see page 6218)	This is GetAny, a member of class IMarshalInBuffer.
	GetChar (see page 6218)	This is GetChar, a member of class IMarshalInBuffer.
	GetDouble (see page 6218)	This is GetDouble, a member of class IMarshalInBuffer.
	GetFloat (see page 6218)	This is GetFloat, a member of class IMarshalInBuffer.
	GetLong (see page 6218)	This is GetLong, a member of class IMarshalInBuffer.
	GetObject (see page 6219)	This is GetObject, a member of class IMarshalInBuffer.
	GetShort (see page 6219)	This is GetShort, a member of class IMarshalInBuffer.
	GetText (see page 6219)	This is GetText, a member of class IMarshalInBuffer.
	GetUnsignedChar (see page 6219)	This is GetUnsignedChar, a member of class IMarshalInBuffer.
	GetUnsignedLong (see page 6220)	This is GetUnsignedLong, a member of class IMarshalInBuffer.
	GetUnsignedShort (see page 6220)	This is GetUnsignedShort, a member of class IMarshalInBuffer.
	GetWidechar (see page 6220)	This is GetWidechar, a member of class IMarshalInBuffer.
	GetWideText (see page 6220)	This is GetWideText, a member of class IMarshalInBuffer.

67.1.3.2 IMarshalInBuffer Methods

The methods of the IMarshalInBuffer class are listed here.

Public Methods

	Name	Description
	Buffer (see page 6217)	This is Buffer, a member of class IMarshalInBuffer.
	GetAny (see page 6218)	This is GetAny, a member of class IMarshalInBuffer.
	GetChar (see page 6218)	This is GetChar, a member of class IMarshalInBuffer.
	GetDouble (see page 6218)	This is GetDouble, a member of class IMarshalInBuffer.
	GetFloat (see page 6218)	This is GetFloat, a member of class IMarshalInBuffer.
	GetLong (see page 6218)	This is GetLong, a member of class IMarshalInBuffer.
	GetObject (see page 6219)	This is GetObject, a member of class IMarshalInBuffer.
	GetShort (see page 6219)	This is GetShort, a member of class IMarshalInBuffer.
	GetText (see page 6219)	This is GetText, a member of class IMarshalInBuffer.
	GetUnsignedChar (see page 6219)	This is GetUnsignedChar, a member of class IMarshalInBuffer.
	GetUnsignedLong (see page 6220)	This is GetUnsignedLong, a member of class IMarshalInBuffer.
	GetUnsignedShort (see page 6220)	This is GetUnsignedShort, a member of class IMarshalInBuffer.
	GetWidechar (see page 6220)	This is GetWidechar, a member of class IMarshalInBuffer.
	GetWideText (see page 6220)	This is GetWideText, a member of class IMarshalInBuffer.

67.1.3.2.1 IMarshalInBuffer.Buffer Method

Pascal

```
function Buffer: PMarshalInbuffer; stdcall;
```

C++

```
__fastcall __stdcall PMarshalInbuffer Buffer();
```

Description

This is Buffer, a member of class IMarshalInBuffer.

67.1.3.2.2 IMarshalInBuffer.GetAny Method

Pascal

```
function GetAny: PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny GetAny();
```

Description

This is GetAny, a member of class IMarshalInBuffer.

67.1.3.2.3 IMarshalInBuffer.GetChar Method

Pascal

```
function GetChar: Shortint; stdcall;
```

C++

```
__fastcall __stdcall Shortint GetChar();
```

Description

This is GetChar, a member of class IMarshalInBuffer.

67.1.3.2.4 IMarshalInBuffer.GetDouble Method

Pascal

```
function GetDouble: Double; stdcall;
```

C++

```
__fastcall __stdcall Double GetDouble();
```

Description

This is GetDouble, a member of class IMarshalInBuffer.

67.1.3.2.5 IMarshalInBuffer.GetFloat Method

Pascal

```
function GetFloat: Single; stdcall;
```

C++

```
__fastcall __stdcall Single GetFloat();
```

Description

This is GetFloat, a member of class IMarshalInBuffer.

67.1.3.2.6 IMarshalInBuffer.GetLong Method

Pascal

```
function GetLong: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetLong();
```

Description

This is GetLong, a member of class IMarshalInBuffer.

67.1.3.2.7 IMarshalInBuffer.GetObject Method

Pascal

```
procedure GetObject(out Obj: ICorbaObj; ref_cnt: CorbaBoolean = True); stdcall;
```

C++

```
__fastcall __stdcall GetObject(ICorbaObj Obj, CorbaBoolean ref_cnt = True);
```

Description

This is GetObject, a member of class IMarshalInBuffer.

67.1.3.2.8 IMarshalInBuffer.GetShort Method

Pascal

```
function GetShort: Smallint; stdcall;
```

C++

```
__fastcall __stdcall Smallint GetShort();
```

Description

This is GetShort, a member of class IMarshalInBuffer.

67.1.3.2.9 IMarshalInBuffer.GetText Method

Pascal

```
function GetText: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * GetText();
```

Description

This is GetText, a member of class IMarshalInBuffer.

67.1.3.2.10 IMarshalInBuffer.GetUnsignedChar Method

Pascal

```
function GetUnsignedChar: Byte; stdcall;
```

C++

```
__fastcall __stdcall Byte GetUnsignedChar();
```

Description

This is GetUnsignedChar, a member of class IMarshalInBuffer.

67.1.3.2.11 IMarshalInBuffer.GetUnsignedLong Method

Pascal

```
function GetUnsignedLong: UINT; stdcall;
```

C++

```
__fastcall __stdcall UINT GetUnsignedLong();
```

Description

This is GetUnsignedLong, a member of class IMarshalInBuffer.

67.1.3.2.12 IMarshalInBuffer.GetUnsignedShort Method

Pascal

```
function GetUnsignedShort: Word; stdcall;
```

C++

```
__fastcall __stdcall Word GetUnsignedShort();
```

Description

This is GetUnsignedShort, a member of class IMarshalInBuffer.

67.1.3.2.13 IMarshalInBuffer.GetWidechar Method

Pascal

```
function GetWidechar: Word; stdcall;
```

C++

```
__fastcall __stdcall Word GetWidechar();
```

Description

This is GetWidechar, a member of class IMarshalInBuffer.

67.1.3.2.14 IMarshalInBuffer.GetWideText Method

Pascal

```
function GetWideText: PWideChar; stdcall;
```

C++

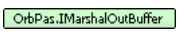
```
__fastcall __stdcall PWideChar GetWideText();
```

Description

This is GetWideText, a member of class IMarshalInBuffer.

67.1.4 IMarshalOutBuffer Interface

Class Hierarchy



Pascal

```
IMarshalOutBuffer = interface;
```

C++

```
__interface IMarshalOutBuffer;
```

File

OrbPas

Description

This is class OrbPas.IMarshalOutBuffer.

67.1.4.1 IMarshalOutBuffer Members

The following tables list the members exposed by IMarshalOutBuffer.

Public Methods

	Name	Description
	Buffer (see page 6222)	This is Buffer, a member of class IMarshalOutBuffer.
	PutAny (see page 6222)	This is PutAny, a member of class IMarshalOutBuffer.
	PutChar (see page 6222)	This is PutChar, a member of class IMarshalOutBuffer.
	PutDouble (see page 6222)	This is PutDouble, a member of class IMarshalOutBuffer.
	PutFloat (see page 6222)	This is PutFloat, a member of class IMarshalOutBuffer.
	PutLong (see page 6223)	This is PutLong, a member of class IMarshalOutBuffer.
	PutObject (see page 6223)	This is PutObject, a member of class IMarshalOutBuffer.
	PutShort (see page 6223)	This is PutShort, a member of class IMarshalOutBuffer.
	PutText (see page 6223)	This is PutText, a member of class IMarshalOutBuffer.
	PutUnsignedChar (see page 6224)	This is PutUnsignedChar, a member of class IMarshalOutBuffer.
	PutUnsignedLong (see page 6224)	This is PutUnsignedLong, a member of class IMarshalOutBuffer.
	PutUnsignedShort (see page 6224)	This is PutUnsignedShort, a member of class IMarshalOutBuffer.
	PutWidechar (see page 6224)	This is PutWidechar, a member of class IMarshalOutBuffer.
	PutWideText (see page 6224)	This is PutWideText, a member of class IMarshalOutBuffer.

67.1.4.2 IMarshalOutBuffer Methods

The methods of the IMarshalOutBuffer class are listed here.

Public Methods

	Name	Description
	Buffer (see page 6222)	This is Buffer, a member of class IMarshalOutBuffer.
	PutAny (see page 6222)	This is PutAny, a member of class IMarshalOutBuffer.
	PutChar (see page 6222)	This is PutChar, a member of class IMarshalOutBuffer.
	PutDouble (see page 6222)	This is PutDouble, a member of class IMarshalOutBuffer.
	PutFloat (see page 6222)	This is PutFloat, a member of class IMarshalOutBuffer.
	PutLong (see page 6223)	This is PutLong, a member of class IMarshalOutBuffer.
	PutObject (see page 6223)	This is PutObject, a member of class IMarshalOutBuffer.
	PutShort (see page 6223)	This is PutShort, a member of class IMarshalOutBuffer.
	PutText (see page 6223)	This is PutText, a member of class IMarshalOutBuffer.
	PutUnsignedChar (see page 6224)	This is PutUnsignedChar, a member of class IMarshalOutBuffer.
	PutUnsignedLong (see page 6224)	This is PutUnsignedLong, a member of class IMarshalOutBuffer.
	PutUnsignedShort (see page 6224)	This is PutUnsignedShort, a member of class IMarshalOutBuffer.
	PutWidechar (see page 6224)	This is PutWidechar, a member of class IMarshalOutBuffer.

	PutWideText (see page 6224)	This is PutWideText, a member of class IMarshalOutBuffer.

67.1.4.2.1 IMarshalOutBuffer.Buffer Method

Pascal

```
function Buffer: PMarshalOutbuffer; stdcall;
```

C++

```
__fastcall __stdcall PMarshalOutbuffer Buffer();
```

Description

This is Buffer, a member of class IMarshalOutBuffer.

67.1.4.2.2 IMarshalOutBuffer.PutAny Method

Pascal

```
procedure PutAny(Value: PCorbaAny); stdcall;
```

C++

```
__fastcall __stdcall PutAny(PCorbaAny Value);
```

Description

This is PutAny, a member of class IMarshalOutBuffer.

67.1.4.2.3 IMarshalOutBuffer.PutChar Method

Pascal

```
procedure PutChar(Value: Shortint); stdcall;
```

C++

```
__fastcall __stdcall PutChar(Shortint Value);
```

Description

This is PutChar, a member of class IMarshalOutBuffer.

67.1.4.2.4 IMarshalOutBuffer.PutDouble Method

Pascal

```
procedure PutDouble(Value: Double); stdcall;
```

C++

```
__fastcall __stdcall PutDouble(Double Value);
```

Description

This is PutDouble, a member of class IMarshalOutBuffer.

67.1.4.2.5 IMarshalOutBuffer.PutFloat Method

Pascal

```
procedure PutFloat(Value: Single); stdcall;
```

C++

```
__fastcall __stdcall PutFloat(Single Value);
```

Description

This is PutFloat, a member of class IMarshalOutBuffer.

67.1.4.2.6 IMarshalOutBuffer.PutLong Method

Pascal

```
procedure PutLong(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall PutLong(int Value);
```

Description

This is PutLong, a member of class IMarshalOutBuffer.

67.1.4.2.7 IMarshalOutBuffer.PutObject Method

Pascal

```
procedure PutObject(const Value: ICorbaObj); stdcall;
```

C++

```
__fastcall __stdcall PutObject(const ICorbaObj Value);
```

Description

This is PutObject, a member of class IMarshalOutBuffer.

67.1.4.2.8 IMarshalOutBuffer.PutShort Method

Pascal

```
procedure PutShort(Value: Smallint); stdcall;
```

C++

```
__fastcall __stdcall PutShort(Smallint Value);
```

Description

This is PutShort, a member of class IMarshalOutBuffer.

67.1.4.2.9 IMarshalOutBuffer.PutText Method

Pascal

```
procedure PutText(Value: PChar); stdcall;
```

C++

```
__fastcall __stdcall PutText(const char * Value);
```

Description

This is PutText, a member of class IMarshalOutBuffer.

67.1.4.2.10 IMarshalOutBuffer.PutUnsignedChar Method

Pascal

```
procedure PutUnsignedChar(Value: Byte); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedChar(Byte Value);
```

Description

This is PutUnsignedChar, a member of class IMarshalOutBuffer.

67.1.4.2.11 IMarshalOutBuffer.PutUnsignedLong Method

Pascal

```
procedure PutUnsignedLong(Value: UINT); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedLong(UINT Value);
```

Description

This is PutUnsignedLong, a member of class IMarshalOutBuffer.

67.1.4.2.12 IMarshalOutBuffer.PutUnsignedShort Method

Pascal

```
procedure PutUnsignedShort(Value: Word); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedShort(Word Value);
```

Description

This is PutUnsignedShort, a member of class IMarshalOutBuffer.

67.1.4.2.13 IMarshalOutBuffer.PutWidechar Method

Pascal

```
procedure PutWidechar(Value: Word); stdcall;
```

C++

```
__fastcall __stdcall PutWidechar(Word Value);
```

Description

This is PutWidechar, a member of class IMarshalOutBuffer.

67.1.4.2.14 IMarshalOutBuffer.PutWideText Method

Pascal

```
procedure PutWideText(Value: PWideChar); stdcall;
```

C++

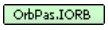
```
__fastcall __stdcall PutWideText(PWideChar Value);
```

Description

This is PutWideText, a member of class IMarshalOutBuffer.

67.1.5 IORB Interface

Class Hierarchy



Pascal

```
IORB = interface;
```

C++

```
__interface IORB;
```

File

OrbPas

Description

This is class OrbPas.IORB.

67.1.5.1 IORB Members

The following tables list the members exposed by IORB.

Public Methods

	Name	Description
	BOAInit (see page 6226)	This is BOAInit, a member of class IORB.
	CreateAliasTC (see page 6226)	This is CreateAliasTC, a member of class IORB.
	CreateArrayTC (see page 6226)	This is CreateArrayTC, a member of class IORB.
	CreateObjRefTC (see page 6227)	This is CreateObjRefTC, a member of class IORB.
	CreateSequenceTC (see page 6227)	This is CreateSequenceTC, a member of class IORB.
	CreateStructTC (see page 6227)	This is CreateStructTC, a member of class IORB.
	CreateTC (see page 6227)	This is CreateTC, a member of class IORB.
	DispatchStruct (see page 6227)	This is DispatchStruct, a member of class IORB.
	FindRepositoryTC (see page 6228)	This is FindRepositoryTC, a member of class IORB.
	MakeAny (see page 6228)	This is MakeAny, a member of class IORB.
	ObjectToString (see page 6228)	This is ObjectToString, a member of class IORB.
	Orb (see page 6228)	This is Orb, a member of class IORB.
	Shutdown (see page 6229)	This is Shutdown, a member of class IORB.
	StringToObject (see page 6229)	This is StringToObject, a member of class IORB.

67.1.5.2 IORB Methods

The methods of the IORB class are listed here.

Public Methods

	Name	Description
	BOAInit (see page 6226)	This is BOAInit, a member of class IORB.

	CreateAliasTC (see page 6226)	This is CreateAliasTC, a member of class IORB.
	CreateArrayTC (see page 6226)	This is CreateArrayTC, a member of class IORB.
	CreateObjRefTC (see page 6227)	This is CreateObjRefTC, a member of class IORB.
	CreateSequenceTC (see page 6227)	This is CreateSequenceTC, a member of class IORB.
	CreateStructTC (see page 6227)	This is CreateStructTC, a member of class IORB.
	CreateTC (see page 6227)	This is CreateTC, a member of class IORB.
	DispatchStruct (see page 6227)	This is DispatchStruct, a member of class IORB.
	FindRepositoryTC (see page 6228)	This is FindRepositoryTC, a member of class IORB.
	MakeAny (see page 6228)	This is MakeAny, a member of class IORB.
	ObjectToString (see page 6228)	This is ObjectToString, a member of class IORB.
	Orb (see page 6228)	This is Orb, a member of class IORB.
	Shutdown (see page 6229)	This is Shutdown, a member of class IORB.
	StringToObject (see page 6229)	This is StringToObject, a member of class IORB.

67.1.5.2.1 IORB.BOAInit Method

Pascal

```
procedure BOAInit(Argc: Integer; const Argv: TArgv; out BOA: IBOA); stdcall;
```

C++

```
__fastcall __stdcall BOAInit(int Argc, const TArgv Argv, IBOA BOA);
```

Description

This is BOAInit, a member of class IORB.

67.1.5.2.2 IORB.CreateAliasTC Method

Pascal

```
procedure CreateAliasTC(RepID: PChar; TypeName: PChar; const TC: ITypeCode; out Result:
ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateAliasTC(const char * RepID, const char * TypeName, const ITypeCode
TC, ITypeCode Result);
```

Description

This is CreateAliasTC, a member of class IORB.

67.1.5.2.3 IORB.CreateArrayTC Method

Pascal

```
procedure CreateArrayTC(Bound: CorbaULong; const TC: ITypeCode; out Result: ITypeCode);
stdcall;
```

C++

```
__fastcall __stdcall CreateArrayTC(CorbaULong Bound, const ITypeCode TC, ITypeCode Result);
```

Description

This is CreateArrayTC, a member of class IORB.

67.1.5.2.4 IORB.CreateObjRefTC Method

Pascal

```
procedure CreateObjRefTC(RepId: PChar; Name: PChar; out Result: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateObjRefTC(const char * RepId, const char * Name, ITypeCode Result);
```

Description

This is CreateObjRefTC, a member of class IORB.

67.1.5.2.5 IORB.CreateSequenceTC Method

Pascal

```
procedure CreateSequenceTC(Bound: CorbaULong; const TC: ITypeCode; out Result: ITypeCode);
stdcall;
```

C++

```
__fastcall __stdcall CreateSequenceTC(CorbaULong Bound, const ITypeCode TC, ITypeCode Result);
```

Description

This is CreateSequenceTC, a member of class IORB.

67.1.5.2.6 IORB.CreateStructTC Method

Pascal

```
procedure CreateStructTC(Kind: TCKind; RepId: PChar; Name: PChar; Members: TStructMembers;
nMember: Integer; out Result: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateStructTC(TCKind Kind, const char * RepId, const char * Name,
TStructMembers Members, int nMember, ITypeCode Result);
```

Description

This is CreateStructTC, a member of class IORB.

67.1.5.2.7 IORB.CreateTC Method

Pascal

```
procedure CreateTC(Kind: TCKind; out TC: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateTC(TCKind Kind, ITypeCode TC);
```

Description

This is CreateTC, a member of class IORB.

67.1.5.2.8 IORB.DispatchStruct Method

Pascal

```
function DispatchStruct(Any: PCorbaAny; CallDesc: PCallDesc; Params: Pointer; out Result:
```

```
  Variant): Integer; stdcall;
```

C++

```
  __fastcall __stdcall int DispatchStruct(PCorbaAny Any, PCallDesc CallDesc, void * Params,
  Variant Result);
```

Description

This is DispatchStruct, a member of class IORB.

67.1.5.2.9 IORB.FindRepositoryTC Method

Pascal

```
  procedure FindRepositoryTC(RepID: PChar; out TC: ITypeCode); stdcall;
```

C++

```
  __fastcall __stdcall FindRepositoryTC(const char * RepID, ITypeCode TC);
```

Description

This is FindRepositoryTC, a member of class IORB.

67.1.5.2.10 IORB.MakeAny Method

Pascal

```
  function MakeAny(const TC: ITypeCode; VS: array of Variant): PCorbaAny; stdcall;
```

C++

```
  __fastcall __stdcall PCorbaAny MakeAny(const ITypeCode TC, array of Variant VS);
```

Description

This is MakeAny, a member of class IORB.

67.1.5.2.11 IORB.ObjectToString Method

Pascal

```
  function ObjectToString(const Obj: IStub): PChar; stdcall;
```

C++

```
  __fastcall __stdcall const char * ObjectToString(const IStub Obj);
```

Description

This is ObjectToString, a member of class IORB.

67.1.5.2.12 IORB.Orb Method

Pascal

```
  function Orb: PCorbaOrb;
```

C++

```
  __fastcall PCorbaOrb Orb();
```

Description

This is Orb, a member of class IORB.

67.1.5.2.13 IORB.Shutdown Method

Pascal

```
procedure Shutdown; stdcall;
```

C++

```
__fastcall __stdcall Shutdown();
```

Description

This is Shutdown, a member of class IORB.

67.1.5.2.14 IORB.StringToObject Method

Pascal

```
procedure StringToObject(Str: PChar; out Obj: IStub); stdcall;
```

C++

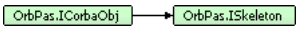
```
__fastcall __stdcall StringToObject(const char * Str, IStub Obj);
```

Description

This is StringToObject, a member of class IORB.

67.1.6 ISkeleton Interface

Class Hierarchy



Pascal

```
ISkeleton = interface(ICorbaObj);
```

C++

```
__interface ISkeleton : public ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.ISkeleton.

67.1.6.1 ISkeleton Members

The following tables list the members exposed by ISkeleton.

Public Methods

	Name	Description
	CorbaObject (see page 6216)	This is CorbaObject, a member of class ICorbaObj.
	GetImplementation (see page 6230)	This is GetImplementation, a member of class ISkeleton.
	GetReplyBuffer (see page 6230)	This is GetReplyBuffer, a member of class ISkeleton.

	IsLocal (see page 6216)	This is IsLocal, a member of class ICorbaObj.

67.1.6.2 ISkeleton Methods

The methods of the ISkeleton class are listed here.

Public Methods

	Name	Description
	GetImplementation (see page 6230)	This is GetImplementation, a member of class ISkeleton.
	GetReplyBuffer (see page 6230)	This is GetReplyBuffer, a member of class ISkeleton.

67.1.6.2.1 ISkeleton.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IUnknown); stdcall;
```

C++

```
__fastcall __stdcall GetImplementation(IUnknown Impl);
```

Description

This is GetImplementation, a member of class ISkeleton.

67.1.6.2.2 ISkeleton.GetReplyBuffer Method

Pascal

```
procedure GetReplyBuffer(cookie: Pointer; out Outbuf: IMarshalOutBuffer); stdcall;
```

C++

```
__fastcall __stdcall GetReplyBuffer(void * cookie, IMarshalOutBuffer Outbuf);
```

Description

This is GetReplyBuffer, a member of class ISkeleton.

67.1.7 ISkeletonObject Interface

Class Hierarchy

OrbPas.ISkeletonObject

Pascal

```
ISkeletonObject = interface;
```

C++

```
__interface ISkeletonObject;
```

File

OrbPas

Description

This is class OrbPas.ISkeletonObject.

67.1.7.1 ISkeletonObject Members

The following tables list the members exposed by ISkeletonObject.

Public Methods

	Name	Description
	Execute (see page 6231)	This is Execute, a member of class ISkeletonObject.
	GetImplementation (see page 6231)	This is GetImplementation, a member of class ISkeletonObject.
	GetSkeleton (see page 6231)	This is GetSkeleton, a member of class ISkeletonObject.

67.1.7.2 ISkeletonObject Methods

The methods of the ISkeletonObject class are listed here.

Public Methods

	Name	Description
	Execute (see page 6231)	This is Execute, a member of class ISkeletonObject.
	GetImplementation (see page 6231)	This is GetImplementation, a member of class ISkeletonObject.
	GetSkeleton (see page 6231)	This is GetSkeleton, a member of class ISkeletonObject.

67.1.7.2.1 ISkeletonObject.Execute Method

Pascal

```
function Execute(Operation: PAnsiChar; const Strm: IMarshalInBuffer; Cookie: Pointer):
CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Execute(PAnsiChar Operation, const IMarshalInBuffer Strm,
void * Cookie);
```

Description

This is Execute, a member of class ISkeletonObject.

67.1.7.2.2 ISkeletonObject.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IUnknown); stdcall;
```

C++

```
__fastcall __stdcall GetImplementation(IUnknown Impl);
```

Description

This is GetImplementation, a member of class ISkeletonObject.

67.1.7.2.3 ISkeletonObject.GetSkeleton Method

Pascal

```
procedure GetSkeleton(out Skeleton: ISkeleton); stdcall;
```

C++

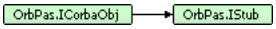
```
__fastcall __stdcall GetSkeleton(ISkeleton Skeleton);
```

Description

This is GetSkeleton, a member of class ISkeletonObject.

67.1.8 IStub Interface

Class Hierarchy



Pascal

```
IStub = interface(ICorbaObj);
```

C++

```
__interface IStub : public ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.IStub.

67.1.8.1 IStub Members

The following tables list the members exposed by IStub.

Public Methods

	Name	Description
	CorbaObject (see page 6216)	This is CorbaObject, a member of class ICorbaObj.
	CreateRequest (see page 6233)	This is CreateRequest, a member of class IStub.
	Dispatch (see page 6233)	This is Dispatch, a member of class IStub.
	GetInterface (see page 6233)	This is GetInterface, a member of class IStub.
	Hash (see page 6233)	This is Hash, a member of class IStub.
	Invoke (see page 6234)	This is Invoke, a member of class IStub.
	IsA (see page 6234)	This is IsA, a member of class IStub.
	IsLocal (see page 6216)	This is IsLocal, a member of class ICorbaObj.
	NonExistent (see page 6234)	This is NonExistent, a member of class IStub.
	RepositoryID (see page 6234)	This is RepositoryID, a member of class IStub.
	SetPrincipal (see page 6235)	This is SetPrincipal, a member of class IStub.

67.1.8.2 IStub Methods

The methods of the IStub class are listed here.

Public Methods

	Name	Description
	CreateRequest (see page 6233)	This is CreateRequest, a member of class IStub.

	Dispatch (see page 6233)	This is Dispatch, a member of class IStub.
	GetInterface (see page 6233)	This is GetInterface, a member of class IStub.
	Hash (see page 6233)	This is Hash, a member of class IStub.
	Invoke (see page 6234)	This is Invoke, a member of class IStub.
	IsA (see page 6234)	This is IsA, a member of class IStub.
	NonExistent (see page 6234)	This is NonExistent, a member of class IStub.
	RepositoryID (see page 6234)	This is RepositoryID, a member of class IStub.
	SetPrincipal (see page 6235)	This is SetPrincipal, a member of class IStub.

67.1.8.2.1 IStub.CreateRequest Method

Pascal

```
procedure CreateRequest(Operation: PChar; ResponseExpected: CorbaBoolean; out Outbuf:
IMarshalOutBuffer); stdcall;
```

C++

```
__fastcall __stdcall CreateRequest(const char * Operation, CorbaBoolean ResponseExpected,
IMarshalOutBuffer Outbuf);
```

Description

This is CreateRequest, a member of class IStub.

67.1.8.2.2 IStub.Dispatch Method

Pascal

```
function Dispatch(CallDesc: PCallDesc; Params: Pointer; out Result: Variant): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Dispatch(PCallDesc CallDesc, void * Params, Variant Result);
```

Description

This is Dispatch, a member of class IStub.

67.1.8.2.3 IStub.GetInterface Method

Pascal

```
function GetInterface: PCorbaInterfaceDef; stdcall;
```

C++

```
__fastcall __stdcall PCorbaInterfaceDef GetInterface();
```

Description

This is GetInterface, a member of class IStub.

67.1.8.2.4 IStub.Hash Method

Pascal

```
function Hash(Maximum: CorbaULong): CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong Hash(CorbaULong Maximum);
```

Description

This is Hash, a member of class IStub.

67.1.8.2.5 IStub.Invoke Method

Pascal

```
procedure Invoke(const Outbuf: IMarshalOutBuffer; out Inbuf: IMarshalInBuffer); stdcall;
```

C++

```
__fastcall __stdcall Invoke(const IMarshalOutBuffer Outbuf, IMarshalInBuffer Inbuf);
```

Description

This is Invoke, a member of class IStub.

67.1.8.2.6 IStub.IsA Method

Pascal

```
function IsA(LogicalTypeId: PChar): CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean IsA(const char * LogicalTypeId);
```

Description

This is IsA, a member of class IStub.

67.1.8.2.7 IStub.NonExistent Method

Pascal

```
function NonExistent: CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean NonExistent();
```

Description

This is NonExistent, a member of class IStub.

67.1.8.2.8 IStub.RepositoryID Method

Pascal

```
function RepositoryID: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * RepositoryID();
```

Description

This is RepositoryID, a member of class IStub.

67.1.8.2.9 IStub.SetPrincipal Method

Pascal

```
procedure SetPrincipal(Bytes: Pointer; Length: CorbaULong); stdcall;
```

C++

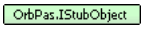
```
__fastcall __stdcall SetPrincipal(void * Bytes, CorbaULong Length);
```

Description

This is SetPrincipal, a member of class IStub.

67.1.9 IStubObject Interface

Class Hierarchy



Pascal

```
IStubObject = interface;
```

C++

```
__interface IStubObject;
```

File

OrbPas

Description

This is class OrbPas.IStubObject.

67.1.9.1 IStubObject Members

The following tables list the members exposed by IStubObject.

Public Methods

	Name	Description
	GetStub (see page 6236)	This is GetStub, a member of class IStubObject.

67.1.9.2 IStubObject Methods

The methods of the IStubObject class are listed here.

Public Methods

	Name	Description
	GetStub (see page 6236)	This is GetStub, a member of class IStubObject.

67.1.9.2.1 IStubObject.GetStub Method

Pascal

```
procedure GetStub(out stub: IStub); stdcall;
```

C++

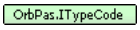
```
__fastcall __stdcall GetStub(IStub stub);
```

Description

This is GetStub, a member of class IStubObject.

67.1.10 ITypeCode Interface

Class Hierarchy



Pascal

```
ITypeCode = interface;
```

C++

```
__interface ITypeCode;
```

File

OrbPas

Description

This is class OrbPas.ITypeCode.

67.1.10.1 ITypeCode Members

The following tables list the members exposed by ITypeCode.

Public Methods

	Name	Description
	ContentType (see page 6237)	This is ContentType, a member of class ITypeCode.
	DefaultIndex (see page 6237)	This is DefaultIndex, a member of class ITypeCode.
	DiscriminatorType (see page 6237)	This is DiscriminatorType, a member of class ITypeCode.
	Equal (see page 6238)	This is Equal, a member of class ITypeCode.
	Id (see page 6238)	This is Id, a member of class ITypeCode.
	Kind (see page 6238)	This is Kind, a member of class ITypeCode.
	Length (see page 6238)	This is Length, a member of class ITypeCode.
	MemberCount (see page 6239)	This is MemberCount, a member of class ITypeCode.
	MemberLabel (see page 6239)	This is MemberLabel, a member of class ITypeCode.
	MemberName (see page 6239)	This is MemberName, a member of class ITypeCode.
	MemberType (see page 6239)	This is MemberType, a member of class ITypeCode.
	Name (see page 6239)	This is Name, a member of class ITypeCode.
	Parameter (see page 6240)	This is Parameter, a member of class ITypeCode.
	ParameterCount (see page 6240)	This is ParameterCount, a member of class ITypeCode.

67.1.10.2 ITypeCode Methods

The methods of the ITypeCode class are listed here.

Public Methods

	Name	Description
	ContentType (see page 6237)	This is ContentType, a member of class ITypeCode.
	DefaultIndex (see page 6237)	This is DefaultIndex, a member of class ITypeCode.
	DiscriminatorType (see page 6237)	This is DiscriminatorType, a member of class ITypeCode.
	Equal (see page 6238)	This is Equal, a member of class ITypeCode.
	Id (see page 6238)	This is Id, a member of class ITypeCode.
	Kind (see page 6238)	This is Kind, a member of class ITypeCode.
	Length (see page 6238)	This is Length, a member of class ITypeCode.
	MemberCount (see page 6239)	This is MemberCount, a member of class ITypeCode.
	MemberLabel (see page 6239)	This is MemberLabel, a member of class ITypeCode.
	MemberName (see page 6239)	This is MemberName, a member of class ITypeCode.
	MemberType (see page 6239)	This is MemberType, a member of class ITypeCode.
	Name (see page 6239)	This is Name, a member of class ITypeCode.
	Parameter (see page 6240)	This is Parameter, a member of class ITypeCode.
	ParameterCount (see page 6240)	This is ParameterCount, a member of class ITypeCode.

67.1.10.2.1 ITypeCode.ContentType Method

Pascal

```
procedure ContentType(out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall ContentType(ITypeCode tc);
```

Description

This is ContentType, a member of class ITypeCode.

67.1.10.2.2 ITypeCode.DefaultIndex Method

Pascal

```
function DefaultIndex: Integer; stdcall;
```

C++

```
__fastcall __stdcall int DefaultIndex();
```

Description

This is DefaultIndex, a member of class ITypeCode.

67.1.10.2.3 ITypeCode.DiscriminatorType Method

Pascal

```
procedure DiscriminatorType(out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall DiscriminatorType(ITypeCode tc);
```

Description

This is DiscriminatorType, a member of class ITypeCode.

67.1.10.2.4 ITypeCode.Equal Method

Pascal

```
function Equal(const tc: ITypeCode): CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Equal(const ITypeCode tc);
```

Description

This is Equal, a member of class ITypeCode.

67.1.10.2.5 ITypeCode.Id Method

Pascal

```
function Id: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * Id();
```

Description

This is Id, a member of class ITypeCode.

67.1.10.2.6 ITypeCode.Kind Method

Pascal

```
function Kind: TCKind; stdcall;
```

C++

```
__fastcall __stdcall TCKind Kind();
```

Description

This is Kind, a member of class ITypeCode.

67.1.10.2.7 ITypeCode.Length Method

Pascal

```
function Length: CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong Length();
```

Description

This is Length, a member of class ITypeCode.

67.1.10.2.8 ITypeCode.MemberCount Method

Pascal

```
function MemberCount: CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong MemberCount();
```

Description

This is MemberCount, a member of class ITypeCode.

67.1.10.2.9 ITypeCode.MemberLabel Method

Pascal

```
function MemberLabel(index: CorbaULong): PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny MemberLabel(CorbaULong index);
```

Description

This is MemberLabel, a member of class ITypeCode.

67.1.10.2.10 ITypeCode.MemberName Method

Pascal

```
function MemberName(index: CorbaULong): PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * MemberName(CorbaULong index);
```

Description

This is MemberName, a member of class ITypeCode.

67.1.10.2.11 ITypeCode.MemberType Method

Pascal

```
procedure MemberType(index: CorbaULong; out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall MemberType(CorbaULong index, ITypeCode tc);
```

Description

This is MemberType, a member of class ITypeCode.

67.1.10.2.12 ITypeCode.Name Method

Pascal

```
function Name: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * Name();
```

Description

This is Name, a member of class ITypeCode.

67.1.10.2.13 ITypeCode.Parameter Method

Pascal

```
function Parameter(index: Integer): PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny Parameter(int index);
```

Description

This is Parameter, a member of class ITypeCode.

67.1.10.2.14 ITypeCode.ParameterCount Method

Pascal

```
function ParameterCount: Integer; stdcall;
```

C++

```
__fastcall __stdcall int ParameterCount();
```

Description

This is ParameterCount, a member of class ITypeCode.

67.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnyToVariant (see page 6241)	This is function OrbPas.AnyToVariant.
	BindStub (see page 6241)	This is function OrbPas.BindStub.
	CorbaAnyType (see page 6242)	This is function OrbPas.CorbaAnyType.
	CorbaDuplicateAny (see page 6242)	This is function OrbPas.CorbaDuplicateAny.
	CorbaReleaseAny (see page 6242)	This is function OrbPas.CorbaReleaseAny.
	CorbaStringFree (see page 6243)	This is function OrbPas.CorbaStringFree.
	CorbaWStringFree (see page 6243)	This is function OrbPas.CorbaWStringFree.
	CreateSkeleton (see page 6243)	This is function OrbPas.CreateSkeleton.
	CreateUserException (see page 6244)	This is function OrbPas.CreateUserException.
	InitORB (see page 6244)	This is function OrbPas.InitORB.
	RegisterUserException (see page 6244)	This is function OrbPas.RegisterUserException.
	SequenceToVariant (see page 6245)	This is function OrbPas.SequenceToVariant.
	UnRegisterUserException (see page 6245)	This is function OrbPas.UnRegisterUserException.
	VariantToAny (see page 6245)	This is function OrbPas.VariantToAny.

67.2.1 OrbPas.AnyToVariant Function

Pascal

```
function AnyToVariant(Value: PCorbaAny; OV: PVariant): Boolean;
```

C++

```
Boolean AnyToVariant(PCorbaAny Value, PVariant OV);
```

File

OrbPas

Description

This is function OrbPas.AnyToVariant.

67.2.2 OrbPas.BindStub Function

Pascal

```
procedure BindStub(RepositoryID: PChar; InstanceName: PChar; HostName: PChar; const Orb: IORB;
RefCountServer: CorbaBoolean; out Stub: IStub);
```

C++

```
BindStub(const char * RepositoryID, const char * InstanceName, const char * HostName, const
IORB Orb, CorbaBoolean RefCountServer, IStub Stub);
```

File

OrbPas

Description

This is function OrbPas.BindStub.

67.2.3 OrbPas.CorbaAnyType Function

Pascal

```
procedure CorbaAnyType(Any: PCorbaAny; out TypeCode: ITypeCode);
```

C++

```
CorbaAnyType(PCorbaAny Any, ITypeCode TypeCode);
```

File

OrbPas

Description

This is function OrbPas.CorbaAnyType.

67.2.4 OrbPas.CorbaDuplicateAny Function

Pascal

```
function CorbaDuplicateAny(Any: PCorbaAny): PCorbaAny;
```

C++

```
PCorbaAny CorbaDuplicateAny(PCorbaAny Any);
```

File

OrbPas

Description

This is function OrbPas.CorbaDuplicateAny.

67.2.5 OrbPas.CorbaReleaseAny Function

Pascal

```
procedure CorbaReleaseAny(Any: PCorbaAny);
```

C++

```
CorbaReleaseAny(PCorbaAny Any);
```

File

OrbPas

Description

This is function OrbPas.CorbaReleaseAny.

67.2.6 OrbPas.CorbaStringFree Function

Pascal

```
procedure CorbaStringFree(Str: PChar);
```

C++

```
CorbaStringFree(const char * Str);
```

File

OrbPas

Description

This is function OrbPas.CorbaStringFree.

67.2.7 OrbPas.CorbaWStringFree Function

Pascal

```
procedure CorbaWStringFree(Str: PWideChar);
```

C++

```
CorbaWStringFree(PWideChar Str);
```

File

OrbPas

Description

This is function OrbPas.CorbaWStringFree.

67.2.8 OrbPas.CreateSkeleton Function

Pascal

```
procedure CreateSkeleton(InterfaceName: PChar; const DelphiObject: ISkeletonObject; Serialize:
CorbaBoolean; InstanceName: PChar; RepositoryID: PChar; ClientRefCount: CorbaBoolean; out
Skeleton: ISkeleton);
```

C++

```
CreateSkeleton(const char * InterfaceName, const ISkeletonObject DelphiObject, CorbaBoolean
Serialize, const char * InstanceName, const char * RepositoryID, CorbaBoolean ClientRefCount,
ISkeleton Skeleton);
```

File

OrbPas

Description

This is function OrbPas.CreateSkeleton.

67.2.9 OrbPas.CreateUserException Function

Pascal

```
function CreateUserException(copy: TCopyUserExceptionProc; throwobj: TThrowUserExceptionProc):
PUserExceptionProxy;
```

C++

```
PUserExceptionProxy CreateUserException(TCopyUserExceptionProc copy, TThrowUserExceptionProc
throwobj);
```

File

OrbPas

Description

This is function OrbPas.CreateUserException.

67.2.10 OrbPas.InitORB Function

Pascal

```
procedure InitORB(const Argv: TArgv; out Orb: IORB);
```

C++

```
InitORB(const TArgv Argv, IORB Orb);
```

File

OrbPas

Description

This is function OrbPas.InitORB.

67.2.11 OrbPas.RegisterUserException Function

Pascal

```
function RegisterUserException(Name: PChar; RepositoryID: PChar; Factory:
TUserExceptionFactoryProc): PExceptionDescription;
```

C++

```
PExceptionDescription RegisterUserException(const char * Name, const char * RepositoryID,
TUserExceptionFactoryProc Factory);
```

File

OrbPas

Description

This is function OrbPas.RegisterUserException.

67.2.12 OrbPas.SequenceToVariant Function

Pascal

```
function SequenceToVariant(Value: PCorbaAny; OV: PVariant): Boolean;
```

C++

```
Boolean SequenceToVariant(PCorbaAny Value, PVariant OV);
```

File

OrbPas

Description

This is function OrbPas.SequenceToVariant.

67.2.13 OrbPas.UnRegisterUserException Function

Pascal

```
procedure UnRegisterUserException(Description: PExceptionDescription);
```

C++

```
UnRegisterUserException(PExceptionDescription Description);
```

File

OrbPas

Description

This is function OrbPas.UnRegisterUserException.

67.2.14 OrbPas.VariantToAny Function

Pascal

```
function VariantToAny(Value: PVariant): PCorbaAny;
```

C++

```
PCorbaAny VariantToAny(PVariant Value);
```

File

OrbPas

Description

This is function OrbPas.VariantToAny.

67.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCKind (see page 6246)	This is record OrbPas.TCKind.

Records

	Name	Description
	TStructMember (see page 6247)	This is record OrbPas.TStructMember.

67.3.1 OrbPas.TCKind Enumeration

Pascal

```
TCKind = (
  tk_null,
  tk_void,
  tk_short,
  tk_long,
  tk_ushort,
  tk_ulong,
  tk_float,
  tk_double,
  tk_boolean,
  tk_char,
  tk_octet,
  tk_any,
  tk_TypeCode,
  tk_Principal,
  tk_objref,
  tk_struct,
  tk_union,
  tk_enum,
  tk_string,
  tk_sequence,
  tk_array,
  tk_alias,
  tk_except,
  tk_longlong,
  tk_ulonglong,
  tk_longdouble,
  tk_wchar,
  tk_wstring,
  tk_fixed
);
```

C++

```
enum TCKind {
  tk_null,
  tk_void,
  tk_short,
  tk_long,
  tk_ushort,
```

```
    tk_ulong,
    tk_float,
    tk_double,
    tk_boolean,
    tk_char,
    tk_octet,
    tk_any,
    tk_TypeCode,
    tk_Principal,
    tk_objref,
    tk_struct,
    tk_union,
    tk_enum,
    tk_string,
    tk_sequence,
    tk_array,
    tk_alias,
    tk_except,
    tk_longlong,
    tk_ulonglong,
    tk_longdouble,
    tk_wchar,
    tk_wstring,
    tk_fixed
  };
```

File

OrbPas

Description

This is record OrbPas.TCKind.

67.3.2 OrbPas.TStructMember Record

Pascal

```
  TStructMember = record
    Name: string;
    TC: ITypeCode;
  end;
```

C++

```
  struct TStructMember {
    UnicodeString Name;
    ITypeCode TC;
  };
```

File

OrbPas

Description

This is record OrbPas.TStructMember.

67.4 Types

The following table lists types in this documentation.

Types

Name	Description
CorbaBoolean (see page 6248)	This is type OrbPas.CorbaBoolean.
CorbaULong (see page 6248)	This is type OrbPas.CorbaULong.
PCorbaAny (see page 6249)	This is type OrbPas.PCorbaAny.
PCorbaInterfaceDef (see page 6249)	This is type OrbPas.PCorbaInterfaceDef.
PCorbaObject (see page 6249)	This is type OrbPas.PCorbaObject.
PCorbaOrb (see page 6250)	This is type OrbPas.PCorbaOrb.
PExceptionDescription (see page 6250)	This is type OrbPas.PExceptionDescription.
PMarshalInbuffer (see page 6250)	This is type OrbPas.PMarshalInbuffer.
PMarshalOutbuffer (see page 6251)	This is type OrbPas.PMarshalOutbuffer.
PUserExceptionProxy (see page 6251)	This is type OrbPas.PUserExceptionProxy.
TArgv (see page 6251)	This is type OrbPas.TArgv.
TCopyUserExceptionProc (see page 6252)	This is type OrbPas.TCopyUserExceptionProc.
TStructMembers (see page 6252)	This is type OrbPas.TStructMembers.
TThrowUserExceptionProc (see page 6252)	This is type OrbPas.TThrowUserExceptionProc.
TUserExceptionFactoryProc (see page 6253)	This is type OrbPas.TUserExceptionFactoryProc.

67.4.1 OrbPas.CorbaBoolean Type

Pascal

```
CorbaBoolean = ByteBool;
```

C++

```
ByteBool CorbaBoolean;
```

File

OrbPas

Description

This is type OrbPas.CorbaBoolean.

67.4.2 OrbPas.CorbaULong Type

Pascal

```
CorbaULong = Integer;
```

C++

```
int CorbaULong;
```

File

OrbPas

Description

This is type OrbPas.CorbaULong.

67.4.3 OrbPas.PCorbaAny Type

Pascal

```
PCorbaAny = type Pointer;
```

C++

```
type Pointer PCorbaAny;
```

File

OrbPas

Description

This is type OrbPas.PCorbaAny.

67.4.4 OrbPas.PCorbaInterfaceDef Type

Pascal

```
PCorbaInterfaceDef = type Pointer;
```

C++

```
type Pointer PCorbaInterfaceDef;
```

File

OrbPas

Description

This is type OrbPas.PCorbaInterfaceDef.

67.4.5 OrbPas.PCorbaObject Type

Pascal

```
PCorbaObject = type Pointer;
```

C++

```
type Pointer PCorbaObject;
```

File

OrbPas

Description

This is type OrbPas.PCorbaObject.

67.4.6 OrbPas.PCorbaOrb Type

Pascal

```
PCorbaOrb = type Pointer;
```

C++

```
type Pointer PCorbaOrb;
```

File

OrbPas

Description

This is type OrbPas.PCorbaOrb.

67.4.7 OrbPas.PExceptionDescription Type

Pascal

```
PExceptionDescription = type Pointer;
```

C++

```
type Pointer PExceptionDescription;
```

File

OrbPas

Description

This is type OrbPas.PExceptionDescription.

67.4.8 OrbPas.PMarshalInbuffer Type

Pascal

```
PMarshalInbuffer = type Pointer;
```

C++

```
type Pointer PMarshalInbuffer;
```

File

OrbPas

Description

This is type OrbPas.PMarshalInbuffer.

67.4.9 OrbPas.PMarshalOutbuffer Type

Pascal

```
PMarshalOutbuffer = type Pointer;
```

C++

```
type Pointer PMarshalOutbuffer;
```

File

OrbPas

Description

This is type OrbPas.PMarshalOutbuffer.

67.4.10 OrbPas.PUserExceptionProxy Type

Pascal

```
PUserExceptionProxy = type Pointer;
```

C++

```
type Pointer PUserExceptionProxy;
```

File

OrbPas

Description

This is type OrbPas.PUserExceptionProxy.

67.4.11 OrbPas.TArgv Type

Pascal

```
TArgv = array of string;
```

C++

```
array of string TArgv;
```

File

OrbPas

Description

This is type OrbPas.TArgv.

67.4.12 OrbPas.TCopyUserExceptionProc Type

Pascal

```
TCopyUserExceptionProc = procedure (const InBuf: IMarshalInBuffer) of object;
```

C++

```
(const InBuf: IMarshalInBuffer) ( TCopyUserExceptionProc)();
```

File

OrbPas

Description

This is type OrbPas.TCopyUserExceptionProc.

67.4.13 OrbPas.TStructMembers Type

Pascal

```
TStructMembers = array of TStructMember;
```

C++

```
array of TStructMember TStructMembers;
```

File

OrbPas

Description

This is type OrbPas.TStructMembers.

67.4.14 OrbPas.TThrowUserExceptionProc Type

Pascal

```
TThrowUserExceptionProc = procedure of object;
```

C++

```
( TThrowUserExceptionProc)();
```

File

OrbPas

Description

This is type OrbPas.TThrowUserExceptionProc.

67.4.15 OrbPas.TUserExceptionFactoryProc Type

Pascal

```
TUserExceptionFactoryProc = function : PUserExceptionProxy;
```

C++

```
: PUserExceptionProxy ( TUserExceptionFactoryProc)();
```

File

OrbPas

Description

This is type OrbPas.TUserExceptionFactoryProc.

68 Outline Namespace

This is namespace Outline.

Topics

Name	Description
Classes (see page 6257)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6380)	The following table lists structs, records, enums in this documentation.
Types (see page 6385)	The following table lists types in this documentation.

Classes

	Name	Description
	EOutlineError (see page 6257)	EOutlineError is the exception class for outline components.
	OutlineError (see page 6258)	OutlineError is a special exception class for invalid outline-node indexes.
	TCustomOutline (see page 6258)	TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 6363) objects).
	TOutline (see page 6296)	TOutline is a multilevel outline that displays the values of TOutlineNode (see page 6363) objects.
	TOutlineNode (see page 6363)	TOutlineNode represents a node in a TCustomOutline (see page 6258) object.

Structs, Records, Enums

	Name	Description
	TAttachMode (see page 6380)	TAttachMode indicates how an outline node should be inserted into an outline.
	TOutlineBitmap (see page 6381)	This is record Outline.TOutlineBitmap.
	TOutlineNodeCompare (see page 6381)	This is record Outline.TOutlineNodeCompare.
	TOutlineOption (see page 6382)	TOutlineOptions (see page 6387) represents the style in which an outline is drawn.
	TOutlineStyle (see page 6382)	TOutlineStyle determines how the nodes of an outline are drawn.
	TOutlineType (see page 6383)	TOutlineType indicates whether an outline is owner-drawn.

Types

Name	Description
EOutlineChange (see page 6385)	EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.
TBitmapArray (see page 6385)	This is type Outline.TBitmapArray.
TBitmapArrayRange (see page 6386)	This is type Outline.TBitmapArrayRange.
TChangeRange (see page 6386)	TChangeRange indicates the direction in which a node is moved.
TOutlineBitmaps (see page 6386)	This is type Outline.TOutlineBitmaps.

TOutlineOptions (see page 6387)	TOutlineOptions represents the style in which an outline is drawn.

68.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOutlineError (see page 6257)	EOutlineError is the exception class for outline components.
	OutlineError (see page 6258)	OutlineError is a special exception class for invalid outline-node indexes.
	TCustomOutline (see page 6258)	TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 6363) objects).
	TOutline (see page 6296)	TOutline is a multilevel outline that displays the values of TOutlineNode (see page 6363) objects.
	TOutlineNode (see page 6363)	TOutlineNode represents a node in a TCustomOutline (see page 6258) object.

68.1.1 EOutlineError Class

EOutlineError is the exception class for outline components.

Class Hierarchy

Pascal

```
EOutlineError = class(Exception);
```

C++

```
class EOutlineError : public Exception;
```

File

Outline

Description

EOutlineError is raised when an error occurs as an application works with an outline component. For example, an EOutlineError exception occurs when an application

Attempts to access an outline using an invalid node index.

Attempts to expand a node whose parent isn't expanded.

Attempts to load items from a file using an invalid file extension.

Note: EOutlineError is raised by the TOutline (see page 6296) component, which is maintained for backward compatibility.

See Also

OutlineError (see page 6258)

68.1.2 OutlineError Class

OutlineError is a special exception class for invalid outline-node indexes.

Class Hierarchy

Pascal

```
OutlineError = class(TObject);
```

C++

```
class OutlineError : public TObject;
```

File

Outline

Description

OutlineError is raised when an application cannot access a specified outline node. In most cases, this occurs because the index used is invalid.

OutlineError is raised internally to abort look-ups of outline components. Because it does not descend from Exception, it never appears at runtime—even if the debugger is set to break on exceptions. Thus it is a useful tool for aborting deep recursions.

Note: OutlineError is raised by the TOutline (see page 6296) component, which is maintained for backward compatibility.

See Also

EOutlineError (see page 6257)

Exception

68.1.3 TCustomOutline Class

TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 6363) objects).

Class Hierarchy

Pascal

```
TCustomOutline = class(TCustomGrid);
```

C++

```
class TCustomOutline : public TCustomGrid;
```

File

Outline

Description

Use TCustomOutline as a base class when defining a control that displays outline nodes in a two-dimensional layout. TCustomOutline introduces the ability to manage a hierarchy of outline nodes, including the ability to control the appearance and layout of those nodes and to expand or collapse branches of the outline tree.

Do not create instances of TCustomOutline. To add an outline control to a form, use the TOutline (see page 6296) object, which publishes many of the properties, events, and methods of TCustomOutline, or TTreeView, which wraps the Windows control that performs the same function.

See Also

TOutline (see page 6296)

TOutlineNode (see page 6363)

TTreeView

68.1.3.1 TCustomOutline Members

The following tables list the members exposed by TCustomOutline.

Protected Events

	Name	Description
	OnCollapse (see page 6295)	Occurs immediately after a node in the outline is collapsed.
	OnDrawItem (see page 6295)	Occurs when a node must be drawn in an owner-draw outline.
	OnExpand (see page 6296)	Occurs immediately after a node in the outline is expanded.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BadIndex (see page 6267)	Indicates whether a node's index is up to date.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
V	Click (see page 6268)	Updates the SelectedItem (see page 6294) property.
	Collapse (see page 6269)	Generates an OnCollapse (see page 6295) event.
	ColumnMoved (see page 3039)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
V	CreateEditor (see page 3040)	Creates the inplace editor control.

	CreateParams (see page 3041)	Initializes the window-creation parameter record.
	DblClick (see page 6269)	Expands or collapses the selected node.
	DefineProperties (see page 6269)	Creates a binary property for customized bitmap values.
	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
	DeleteNode (see page 6270)	Fixes up the outline after a node has been deleted.
	DeleteRow (see page 3042)	Deletes a specified from the grid.
	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
	DrawCell (see page 6271)	Draws a node in the outline.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	Expand (see page 6272)	Generates an OnExpand (see page 6296) event.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 3046)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 6276)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	KeyDown (see page 6277)	Provides special processing when the user presses a key.
	KeyPress (see page 6277)	Provides special processing when a key is pressed.
	Loaded (see page 6278)	Sets up the outline after it has been streamed in.
	MouseDown (see page 6280)	Updates the SelectedItem (see page 6294) property.
	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	Move (see page 6280)	Repositions a node in the outline.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3055)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
	SetDisplayWidth (see page 6282)	Sets the display width of the outline.
	SetEditText (see page 3056)	Provides the interface for a method that updates the text associated with a cell.
	SetGoodIndex (see page 6282)	Returns the first node in the Items (see page 6287) array that must be reindexed to make the index of the Value parameter correct.
	SetLevel (see page 6283)	Sets the level of an outline node.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.

	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	BorderStyle (see page 3060)	Determines whether a single line border is drawn around the grid.
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	Color (see page 3063)	Specifies the background color of the control.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	ItemHeight (see page 6286)	Specifies the height of each node in the outline.
	ItemSeparator (see page 6288)	Specifies a delimiting string for outline nodes.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Lines (see page 6288)	Lists the Text property values of all nodes in the outline.
	Options (see page 6288)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 6289)	Specifies how the outline tree is displayed.
	ParentColor (see page 3073)	Determines where a control looks for its color information.
	PictureClosed (see page 6290)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 6291)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 6291)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 6291)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 6292)	Specifies the bitmap used to represent a collapsed item that has subitems.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	ScrollBars (see page 3077)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	Style (see page 6294)	Determines whether the nodes and tree are drawn automatically.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Public Methods

	Name	Description
	Add (see page 6264)	Adds a node to the outline.
	AddChild (see page 6265)	Adds a child node to a node in the outline.
	AddChildObject (see page 6266)	Adds a child node containing data to a node in the outline.
	AddObject (see page 6266)	Adds a node containing data to the outline.

	BeginUpdate (see page 6268)	Prevents the updating of the outline until the EndUpdate (see page 6271) method is called.
	Clear (see page 6268)	Removes all items from the outline.
V	Create (see page 6262)	Creates an instance of TCustomOutline (see page 6258).
	Delete (see page 6270)	Deletes a node from the outline.
V	Destroy (see page 6263)	Destroys an instance of TCustomOutline (see page 6258).
	EndUpdate (see page 6271)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 6268).
	FullCollapse (see page 6272)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 6272)	Expands every node in the outline.
	GetDataItem (see page 6273)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetItem (see page 6273)	Returns the index of the node at the position indicated by the X and Y parameters.
	GetNodeDisplayWidth (see page 6274)	Returns the length of the row containing a specified node.
	GetTextItem (see page 6274)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetVisibleNode (see page 6274)	Returns the TOutlineNode (see page 6363) object for the node indicated by a position in the image of the outline.
	Insert (see page 6275)	Inserts a node at a specified position.
	InsertObject (see page 6275)	Inserts a node containing data at a specified position.
	LoadFromFile (see page 6278)	This is the overview for the LoadFromFile method overload.
	LoadFromStream (see page 6279)	This is the overview for the LoadFromStream method overload.
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	SaveToFile (see page 6280)	This is the overview for the SaveToFile method overload.
	SaveToStream (see page 6281)	This is the overview for the SaveToStream method overload.
	SetUpdateState (see page 6283)	Enables or disables the maintenance of the node indexes.

Public Properties

	Name	Description
	Canvas (see page 6284)	Specifies the TCanvas object that presents a drawing surface for the control.
R	ItemCount (see page 6286)	Indicates the number of nodes in the outline.
R	Items (see page 6287)	Lists the nodes in the outline.
	Row (see page 6292)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 6294)	Specifies the index of the selected node.

Published Properties

	Name	Description
	TabStop (see page 3079)	Determines if the user can tab to a control.

68.1.3.2 TCustomOutline.Create Constructor

Creates an instance of TCustomOutline (see page 6258).

Pascal

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

C++

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

Description

Call Create to instantiate a TCustomOutline (see page 6258) object at runtime. Outlines placed on forms at design time are created automatically.

Create initializes the outline object, setting the values of some internal properties, as well as setting

Color to clWindow (see page 2965) and ParentColor to false.

Height to 97 and Width to 121.

ItemSeparator (see page 6288) to '\'.

Options (see page 6288) to [ooDrawTreeRoot, ooDrawFocusRect].

OutlineStyle (see page 6289) to osTreePictureText.

See Also

Height

ItemSeparator (see page 6288)

Options (see page 6288)

OutlineStyle (see page 6289)

Color

ParentColor

Create

Width

68.1.3.3 TCustomOutline.Destroy Destructor

Destroys an instance of TCustomOutline (see page 6258).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomOutline();
```

Description

Destroy frees all memory associated with the outline. While the destructor frees the TOutlineNode (see page 6363) objects that comprise the outline, any objects associated with those nodes are not freed.

68.1.3.4 TCustomOutline Methods

The methods of the TCustomOutline class are listed here.

Protected Methods

	Name	Description
	BadIndex (see page 6267)	Indicates whether a node's index is up to date.
V	Click (see page 6268)	Updates the SelectedItem (see page 6294) property.
	Collapse (see page 6269)	Generates an OnCollapse (see page 6295) event.
V	DblClick (see page 6269)	Expands or collapses the selected node.
V	DefineProperties (see page 6269)	Creates a binary property for customized bitmap values.
	DeleteNode (see page 6270)	Fixes up the outline after a node has been deleted.
V	DrawCell (see page 6271)	Draws a node in the outline.
	Expand (see page 6272)	Generates an OnExpand (see page 6296) event.
	InvalidateCell (see page 6276)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.

	KeyDown (see page 6277)	Provides special processing when the user presses a key.
	KeyPress (see page 6277)	Provides special processing when a key is pressed.
	Loaded (see page 6278)	Sets up the outline after it has been streamed in.
	MouseDown (see page 6280)	Updates the SelectedItem (see page 6294) property.
	Move (see page 6280)	Repositions a node in the outline.
	SetDisplayWidth (see page 6282)	Sets the display width of the outline.
	SetGoodIndex (see page 6282)	Returns the first node in the Items (see page 6287) array that must be reindexed to make the index of the Value parameter correct.
	SetLevel (see page 6283)	Sets the level of an outline node.

Public Methods

	Name	Description
	Add (see page 6264)	Adds a node to the outline.
	AddChild (see page 6265)	Adds a child node to a node in the outline.
	AddChildObject (see page 6266)	Adds a child node containing data to a node in the outline.
	AddObject (see page 6266)	Adds a node containing data to the outline.
	BeginUpdate (see page 6268)	Prevents the updating of the outline until the EndUpdate (see page 6271) method is called.
	Clear (see page 6268)	Removes all items from the outline.
	Delete (see page 6270)	Deletes a node from the outline.
	EndUpdate (see page 6271)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 6268).
	FullCollapse (see page 6272)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 6272)	Expands every node in the outline.
	GetDataItem (see page 6273)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetItem (see page 6273)	Returns the index of the node at the position indicated by the X and Y parameters.
	GetNodeDisplayWidth (see page 6274)	Returns the length of the row containing a specified node.
	GetTextItem (see page 6274)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetVisibleNode (see page 6274)	Returns the TOutlineNode (see page 6363) object for the node indicated by a position in the image of the outline.
	Insert (see page 6275)	Inserts a node at a specified position.
	InsertObject (see page 6275)	Inserts a node containing data at a specified position.
	LoadFromFile (see page 6278)	This is the overview for the LoadFromFile method overload.
	LoadFromStream (see page 6279)	This is the overview for the LoadFromStream method overload.
	SaveToFile (see page 6280)	This is the overview for the SaveToFile method overload.
	SaveToStream (see page 6281)	This is the overview for the SaveToStream method overload.
	SetUpdateState (see page 6283)	Enables or disables the maintenance of the node indexes.

68.1.3.4.1 TCustomOutline.Add Method

Adds a node to the outline.

Pascal

```
function Add(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt Add(LongInt Index, const UnicodeString Text);
```

Description

Use Add to add a TOutlineNode (see page 6363) object to the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last sibling of the node identified by Index. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

Add returns the Index property of the new node.

See Also

AddChild (see page 6265)

AddObject (see page 6266)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

Insert (see page 6275)

Items (see page 6287)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.2 TCustomOutline.AddChild Method

Adds a child node to a node in the outline.

Pascal

```
function AddChild(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt AddChild(LongInt Index, const UnicodeString Text);
```

Description

Use AddChild to add a TOutlineNode (see page 6363) object as the child of a node in the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last child of the node identified by Index. To add nodes to the top level of the outline (off the root), specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

AddChild returns the Index property of the new node.

See Also

Add (see page 6264)

AddChildObject (see page 6266)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

Insert (see page 6275)

Items (see page 6287)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.3 TCustomOutline.AddChildObject Method

Adds a child node containing data to a node in the outline.

Pascal

```
function AddChildObject(Index: LongInt; const Text: string; const Data: TCustomData): LongInt;
```

C++

```
__fastcall LongInt AddChildObject(LongInt Index, const UnicodeString Text, const TCustomData
Data);
```

Description

Use AddChildObject to add a TOutlineNode (see page 6363) object as the child of a node in the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last child of the node identified by Index. To add nodes to the top level of the outline (off the root), specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

AddChildObject returns the Index property of the new node.

See Also

AddChild (see page 6265)

AddObject (see page 6266)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

InsertObject (see page 6275)

Items (see page 6287)

Data (see page 6374)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.4 TCustomOutline.AddObject Method

Adds a node containing data to the outline.

Pascal

```
function AddObject(Index: LongInt; const Text: string; const Data: TCustomData): LongInt;
```

C++

```
__fastcall LongInt AddObject(LongInt Index, const UnicodeString Text, const TCustomData Data);
```

Description

Use AddObject to add a TOutlineNode (see page 6363) object to the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last sibling of the node identified by Index. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

AddChild (see page 6265) returns the Index property of the new node.

See Also

Add (see page 6264)

AddChildObject (see page 6266)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

InsertObject (see page 6275)

Items (see page 6287)

Data (see page 6374)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.5 TCustomOutline.BadIndex Method

Indicates whether a node's index is up to date.

Pascal

```
function BadIndex(Value: TOutlineNode): Boolean;
```

C++

```
__fastcall Boolean BadIndex(TOutlineNode Value);
```

Description

Use BadIndex to determine whether the Index property of the TOutlineNode (see page 6363) object specified by the Value parameter accurately reflects the position of the node in the outline.

If nodes are added or removed after a call to BeginUpdate (see page 6268), the indexes may be out of date until the EndUpdate (see page 6271) method is called.

See Also

BeginUpdate (see page 6268)

EndUpdate (see page 6271)

Index (see page 6376)

68.1.3.4.6 TCustomOutline.BeginUpdate Method

Prevents the updating of the outline until the EndUpdate (see page 6271) method is called.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before adding, removing, or moving multiple outline items. BeginUpdate speeds processing time by deferring repaints and the adjustment of index values until EndUpdate (see page 6271) is called.

Note: Make sure that EndUpdate (see page 6271) is called after all changes have been made to the nodes in the Items (see page 6287) array.

See Also

BadIndex (see page 6267)

EndUpdate (see page 6271)

Items (see page 6287)

68.1.3.4.7 TCustomOutline.Clear Method

Removes all items from the outline.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the outline of all its nodes. Clear frees the memory for the removed items but not any memory for associated data.

68.1.3.4.8 TCustomOutline.Click Method

Updates the SelectedItem (see page 6294) property.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user clicks the mouse on a node in the outline. Click updates the SelectedItem (see page 6294) property before the OnClick event handler is called by the inherited method.

See Also

DblClick (see page 6269)

SelectedItem (see page 6294)

68.1.3.4.9 TCustomOutline.Collapse Method

Generates an OnCollapse (see page 6295) event.

Pascal

```
procedure Collapse(Index: LongInt); dynamic;
```

C++

```
__fastcall Collapse(LongInt Index);
```

Description

Collapse is called automatically after a node in the outline is collapsed. It calls the OnCollapse (see page 6295) event handler if it is assigned.

Override the Collapse method to provide additional processing after nodes in the outline have been collapsed, or to block the OnCollapse (see page 6295) event.

See Also

Expand (see page 6272)

OnCollapse (see page 6295)

68.1.3.4.10 TCustomOutline.DblClick Method

Expands or collapses the selected node.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick is called automatically when the user double clicks the mouse on a node in the outline. After calling the OnDblClick event handler, DblClick updates the SelectedItem (see page 6294) property and expands the selected node if it is collapsed or collapses the selected node if it is expanded.

See Also

Click (see page 6268)

SelectedItem (see page 6294)

Expanded (see page 6375)

68.1.3.4.11 TCustomOutline.DefineProperties Method

Creates a binary property for customized bitmap values.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

The streaming system calls DefineProperties to allow the outline to define new properties that should be saved with the outline object. DefineProperties checks whether the outline uses any bitmaps other than the default values of PictureClosed (see page 6290), PictureLeaf (see page 6291), PictureMinus (see page 6291), PictureOpen (see page 6291), and PicturePlus (see page 6292). If the outline has customized any of these bitmaps, DefineProperties uses the DefineBinaryProperty method of the Filer parameter to create a binary property for the customized values.

See Also

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

TWriter

DefineBinaryProperty

68.1.3.4.12 TCustomOutline.Delete Method

Deletes a node from the outline.

Pascal

```
procedure Delete(Index: LongInt);
```

C++

```
__fastcall Delete(LongInt Index);
```

Description

Call Delete to remove a node and all its descendants from the outline. Index is the index of the node to delete.

Delete removes the node and its children, frees all associated memory, and then calls DeleteNode (see page 6270) to update the indexes and make sure that SelectedItem (see page 6294) is valid.

See Also

Add (see page 6264)

DeleteNode (see page 6270)

SelectedItem (see page 6294)

68.1.3.4.13 TCustomOutline.DeleteNode Method

Fixes up the outline after a node has been deleted.

Pascal

```
procedure DeleteNode(Node: TOutlineNode; CurIndex: LongInt);
```

C++

```
__fastcall DeleteNode(TOutlineNode Node, LongInt CurIndex);
```

Description

DeleteNode is called automatically whenever a node in the outline is destroyed. It fixes up the index and makes sure that the

SelectedItem (see page 6294) property is valid.

To Delete (see page 6270) a node from the outline, call the Delete (see page 6270) method of the outline or free the TOulineNode object.

See Also

Delete (see page 6270)

SelectedItem (see page 6294)

68.1.3.4.14 TCustomOutline.DrawCell Method

Draws a node in the outline.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever the outline needs to draw a node. If the Style (see page 6294) property is otOwnerDraw and an OnDrawItem (see page 6295) event handler has been supplied, DrawCell generates an OnDrawItem (see page 6295) event. Otherwise, DrawCell draws the node in the manner indicated by the OutlineStyle (see page 6289) property.

See Also

OnDrawItem (see page 6295)

OutlineStyle (see page 6289)

68.1.3.4.15 TCustomOutline.EndUpdate Method

Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 6268).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate at the end of a segment of code that began with a call to BeginUpdate (see page 6268). Every time BeginUpdate (see page 6268) is called, an internal counter is incremented. When EndUpdate is called, the counter is decremented. When every call to BeginUpdate (see page 6268) has been matched by a call to EndUpdate, EndUpdate fixes up the indexes and repaints the outline.

See Also

BeginUpdate (see page 6268)

Items (see page 6287)

68.1.3.4.16 TCustomOutline.Expand Method

Generates an OnExpand (see page 6296) event.

Pascal

```
procedure Expand(Index: LongInt); dynamic;
```

C++

```
__fastcall Expand(LongInt Index);
```

Description

Expand is called automatically after a node in the outline is expanded. It calls the OnExpand (see page 6296) event handler if it is assigned.

Override the Expand method to provide additional processing after nodes in the outline have been expanded, or to block the OnExpand (see page 6296) event.

See Also

Collapse (see page 6269)

OnExpand (see page 6296)

68.1.3.4.17 TCustomOutline.FullCollapse Method

Causes all top-level nodes in the outline to collapse.

Pascal

```
procedure FullCollapse;
```

C++

```
__fastcall FullCollapse();
```

Description

Call FullCollapse to shrink the outline as much as possible. FullCollapse sets the Expanded property of each TOutlineNode (see page 6363) that is a direct descendant of the root of the outline to false. This causes all the descendants of these top-level nodes to be hidden. Depending on the value of OutlineStyle (see page 6289), each collapsed top-level node is then represented with the PictureClosed (see page 6290) bitmap and/or the PicturePlus (see page 6292) bitmap.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PicturePlus (see page 6292)

Expanded (see page 6375)

68.1.3.4.18 TCustomOutline.FullExpand Method

Expands every node in the outline.

Pascal

```
procedure FullExpand;
```

C++

```
__fastcall FullExpand();
```

Description

Call FullExpand to show all nodes in the outline. FullExpand calls the FullExpand method of the root node. This causes the Expanded property of all nodes in the outline to be set to true. Every node in the Items (see page 6287) array appears fully expanded in the outline. Depending on the value of OutlineStyle (see page 6289), each node that has descendants is then represented with the PictureOpen (see page 6291) bitmap and/or the PictureMinus (see page 6291) bitmap. The appearance of leaf nodes is unaffected.

See Also

OutlineStyle (see page 6289)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

FullExpand (see page 6368)

68.1.3.4.19 TCustomOutline.GetDataItem Method

Returns the index of the first node in the outline that has the Value pointer as its Data property.

Pascal

```
function GetDataItem(Value: TCustomData): Longint;
```

C++

```
__fastcall Longint GetDataItem(TCustomData Value);
```

Description

Call GetDataItem to locate the node that corresponds to the Value parameter. GetDataItem returns an index into the Items (see page 6287) property array.

See Also

Items (see page 6287)

Data (see page 6374)

68.1.3.4.20 TCustomOutline.GetItem Method

Returns the index of the node at the position indicated by the X and Y parameters.

Pascal

```
function GetItem(X: Integer; Y: Integer): LongInt;
```

C++

```
__fastcall LongInt GetItem(int X, int Y);
```

Description

Call GetItem to locate the node at the specified screen pixel coordinates. GetItem returns an index into the Items (see page 6287) property array.

See Also

Items (see page 6287)

68.1.3.4.21 TCustomOutline.GetNodeDisplayWidth Method

Returns the length of the row containing a specified node.

Pascal

```
function GetNodeDisplayWidth(Node: TOutlineNode): Integer;
```

C++

```
__fastcall int GetNodeDisplayWidth(TOutlineNode Node);
```

Description

Call GetNodeDisplayWidth to determine the length, in pixels, of the line in the outline that contains the node specified by the Node parameter. GetNodeDisplayWidth adjusts for the lines and bitmaps specified by the OutlineStyle (see page 6289) property.

See Also

ItemHeight (see page 6286)

OutlineStyle (see page 6289)

68.1.3.4.22 TCustomOutline.GetTextItem Method

Returns the index of the first node in the outline with the Value parameter as its Text property.

Pascal

```
function GetTextItem(const Value: string): Longint;
```

C++

```
__fastcall Longint GetTextItem(const UnicodeString Value);
```

Description

Call GetTextItem to locate the node that corresponds to the Value parameter. GetTextItem returns an index into the Items (see page 6287) property array.

See Also

Items (see page 6287)

Text (see page 6378)

68.1.3.4.23 TCustomOutline.GetVisibleNode Method

Returns the TOutlineNode (see page 6363) object for the node indicated by a position in the image of the outline.

Pascal

```
function GetVisibleNode(Index: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetVisibleNode(LongInt Index);
```

Description

Call GetVisibleNode to locate a node with the position given by the Index parameter, where all nodes that are hidden because their parent node is collapsed are ignored. An Index of 0 indicates the first visible node, an Index of 1 indicates the second visible node, and so on. If the outline is fully expanded, GetVisibleNode returns the value of Items (see page 6287)[Index+1].

See Also

Items (see page 6287)

68.1.3.4.24 TCustomOutline.Insert Method

Inserts a node at a specified position.

Pascal

```
function Insert(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt Insert(LongInt Index, const UnicodeString Text);
```

Description

Use Insert to add a TOutlineNode (see page 6363) object to the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline in the position currently occupied by the node identified by Index. The new node is inserted at the same level as the node currently identified by Index, so that the original node and the new node are siblings and share the same parent. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node, including the one specified by Index before Insert is called, are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

Insert returns the Index property of the new node.

See Also

Add (see page 6264)

AddChild (see page 6265)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

InsertObject (see page 6275)

Items (see page 6287)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.25 TCustomOutline.InsertObject Method

Inserts a node containing data at a specified position.

Pascal

```
function InsertObject(Index: LongInt; const Text: string; const Data: TCustomData): LongInt;
```

C++

```
__fastcall LongInt InsertObject(LongInt Index, const UnicodeString Text, const TCustomData
Data);
```

Description

Use InsertObject to add a TOutlineNode (see page 6363) object to the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline in the position currently occupied by the node identified by Index. The new node is inserted at the same level as the node currently identified by Index, so that the original node and the new node are siblings and share the same parent. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node, including the one specified by Index before Insert (see page 6275) is called, are moved down one row and reindexed in the Items (see page 6287) property array with valid Index values. If BeginUpdate (see page 6268) has been called, the Items (see page 6287) array is not reindexed until the EndUpdate (see page 6271) method is called.

Insert (see page 6275) returns the Index property of the new node.

See Also

AddObject (see page 6266)

AddChildObject (see page 6266)

BeginUpdate (see page 6268)

Delete (see page 6270)

EndUpdate (see page 6271)

Insert (see page 6275)

Items (see page 6287)

Data (see page 6374)

Index (see page 6376)

Text (see page 6378)

68.1.3.4.26 TCustomOutline.InvalidateCell Method

Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateCell(ACol: Longint; ARow: Longint);
```

C++

```
__fastcall InvalidateCell(Longint ACol, Longint ARow);
```

Description

Call InvalidateCell when the contents of a cell change so that the cell will need to be repainted. The ACol parameter is the column index of the cell, and the ARow parameter is the row index.

See Also

CellRect (see page 3038)

InvalidateCol (see page 3048)

InvalidateEditor (see page 3049)

InvalidateRow (see page 3050)

68.1.3.4.27 TCustomOutline.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown is called automatically when the user presses a key. After calling the inherited method, which generates an OnKeyDown event, KeyDown processes the following keystrokes:

Key	Behavior
Home	Selects the first node in the outline.
End	Selects the last node in the outline.
Return	Toggles the selected node between its collapsed and expanded states.
Ctrl+*	Fully expands the outline.
Right arrow	Selects the first child of the selected node.
Left arrow	Selects the parent of the selected node.
Ctrl+Up arrow	Selects the node following the selected node.
Ctrl+Down arrow	Selects the node preceding the selected node.

See Also

KeyPress (see page 6277)

SelectedItem (see page 6294)

KeyDown (see page 3050)

68.1.3.4.28 TCustomOutline.KeyPress Method

Provides special processing when a key is pressed.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically after the user presses a key but before it is released. After calling the inherited method, which generates an OnKeyPress event, KeyPress processes the following characters :

Key	Behavior
+	Expands the selected node.
-	Collapses the selected node.
*	Fully expands the outline.

See Also

KeyDown (see page 6277)

KeyPress (see page 3050)

68.1.3.4.29 TCustomOutline.Loaded Method

Sets up the outline after it has been streamed in.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

The streaming system calls Loaded after the outline has been fully loaded into memory. Loaded makes sure that all indexes are correct, loads any custom bitmaps that replaced the default values, and selects the first node in the outline.

See Also

DefineProperties (see page 6269)

SelectedItem (see page 6294)

SetGoodIndex (see page 6282)

68.1.3.4.30 LoadFromFile Method

This is the overview for the LoadFromFile method overload.

Overload List

	Name	Description
	TCustomOutline.LoadFromFile (string) (see page 6278)	Loads the outline from a text image stored in a file.
	TCustomOutline.LoadFromFile (string, TEncoding) (see page 6278)	Loads the outline from a text image stored in a file.

68.1.3.4.30.1 TCustomOutline.LoadFromFile Method (string)

Loads the outline from a text image stored in a file.

Pascal

```
procedure LoadFromFile(const FileName: string); overload;
procedure LoadFromFile(const FileName: string; Encoding: TEncoding); overload;
```

C++

```
__fastcall LoadFromFile(const UnicodeString FileName);
__fastcall LoadFromFile(const UnicodeString FileName, TEncoding Encoding);
```

Description

Call LoadFromFile to fill an outline from the file specified by the FileName parameter. Each line in the file represents a node in the outline. The file does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items.

For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

To fill the outline from a text image read from a stream, use the LoadFromStream (see page 6279) method. To fill the outline from a text image represented as a TStrings object, use the Lines (see page 6288) property.

See Also

Lines (see page 6288)

LoadFromStream (see page 6279)

SaveToFile (see page 6281)

68.1.3.4.31 LoadFromStream Method

This is the overview for the LoadFromStream method overload.

Overload List

	Name	Description
	TCustomOutline.LoadFromStream (TStream) (see page 6279)	Loads the outline from a text image read from a stream.
	TCustomOutline.LoadFromStream (TStream, TEncoding) (see page 6279)	Loads the outline from a text image read from a stream.

68.1.3.4.31.1 TCustomOutline.LoadFromStream Method (TStream)

Loads the outline from a text image read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); overload;
procedure LoadFromStream(Stream: TStream; Encoding: TEncoding); overload;
```

C++

```
__fastcall LoadFromStream(TStream Stream);
__fastcall LoadFromStream(TStream Stream, TEncoding Encoding);
```

Description

Call LoadFromStream to fill an outline from the stream specified by the Stream parameter. For example, use LoadFromStream to load an outline from an image stored in a database table, using a TBlobStream object. Each node in the outline is separated by a newline character. The stream does not provide the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

To fill the outline from a text image read from a file, use the LoadFromFile (see page 6278) method. To fill the outline from a

text image represented as a TStrings object, use the Lines (see page 6288) property.

See Also

Lines (see page 6288)

LoadFromFile (see page 6278)

SaveToStream (see page 6281)

68.1.3.4.32 TCustomOutline.MouseDown Method

Updates the SelectedItem (see page 6294) property.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method to generate an OnMouseDown event and locate the line under the mouse, MouseDown sets the SelectedItem (see page 6294) property to the node in the line under the mouse.

See Also

SelectedItem (see page 6294)

68.1.3.4.33 TCustomOutline.Move Method

Repositions a node in the outline.

Pascal

```
procedure Move(Destination: LongInt; Source: LongInt; AttachMode: TAttachMode);
```

C++

```
__fastcall Move(LongInt Destination, LongInt Source, TAttachMode AttachMode);
```

Description

Call Move to reposition the node with the index specified by the Source parameter to the position indicated by the Destination parameter. The meaning of the Destination parameter depends on the value of the AttachMode parameter.

See Also

Add (see page 6264)

AddChild (see page 6265)

Insert (see page 6275)

Items (see page 6287)

68.1.3.4.34 SaveToFile Method

This is the overview for the SaveToFile method overload.

Overload List

	Name	Description
	TCustomOutline.SaveToFile (string) (see page 6281)	Saves the Lines (see page 6288) property to a file.
	TCustomOutline.SaveToFile (string, TEncoding) (see page 6281)	Saves the Lines (see page 6288) property to a file.

68.1.3.4.34.1 TCustomOutline.SaveToFile Method (string)

Saves the Lines (see page 6288) property to a file.

Pascal

```
procedure SaveToFile(const FileName: string); overload;
procedure SaveToFile(const FileName: string; Encoding: TEncoding); overload;
```

C++

```
__fastcall SaveToFile(const UnicodeString FileName);
__fastcall SaveToFile(const UnicodeString FileName, TEncoding Encoding);
```

Description

Call SaveToFile to save a text image of the outline to the file specified by the FileName parameter. The image saved is the value of the Lines (see page 6288) property, where each line of the file represents a node, and each level of the outline is represented by a tab character.

To write the outline to an arbitrary stream, use the SaveToStream (see page 6281) method.

See Also

Lines (see page 6288)

LoadFromFile (see page 6278)

SaveToStream (see page 6281)

68.1.3.4.35 SaveToStream Method

This is the overview for the SaveToStream method overload.

Overload List

	Name	Description
	TCustomOutline.SaveToStream (TStream) (see page 6281)	Writes the Lines (see page 6288) property to a stream.
	TCustomOutline.SaveToStream (TStream, TEncoding) (see page 6281)	Writes the Lines (see page 6288) property to a stream.

68.1.3.4.35.1 TCustomOutline.SaveToStream Method (TStream)

Writes the Lines (see page 6288) property to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); overload;
procedure SaveToStream(Stream: TStream; Encoding: TEncoding); overload;
```

C++

```
__fastcall SaveToStream(TStream Stream);
__fastcall SaveToStream(TStream Stream, TEncoding Encoding);
```

Description

Call SaveToStream to write a text image of the outline to the stream specified by the Stream parameter. The image saved is the value of the Lines (see page 6288) property, where each line of text represents a node, and each level of the outline is represented by a tab character.

To write the outline to a file, use the SaveToFile (see page 6281) method.

See Also

Lines (see page 6288)

LoadFromStream (see page 6279)

SaveToFile (see page 6281)

68.1.3.4.36 TCustomOutline.SetDisplayWidth Method

Sets the display width of the outline.

Pascal

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

C++

```
__fastcall SetDisplayWidth(int Value);
```

Description

Call SetDisplayWidth to set the width of the image of the outline to the number of pixels specified by the Value parameter. Value should large enough to display the most deeply nested node. Use the GetNodeDisplayWidth (see page 6274) method to determine the size needed to display the visible nodes.

If the display width is greater than the client width of the control, users can scroll the image to view the entire width.

See Also

ClientWidth

GetNodeDisplayWidth (see page 6274)

GetVisibleNode (see page 6274)

68.1.3.4.37 TCustomOutline.SetGoodIndex Method

Returns the first node in the Items (see page 6287) array that must be reindexed to make the index of the Value parameter correct.

Pascal

```
function SetGoodIndex(Value: TOutlineNode): TOutlineNode;
```

C++

```
__fastcall TOutlineNode SetGoodIndex(TOutlineNode Value);
```

Description

Call SetGoodIndex to determine how much of the Items (see page 6287) array must be reindexed after changing the nodes in the outline. All nodes from the root to the value returned by SetGoodIndex do not need to be reindexed.

See Also

BadIndex (see page 6267)

68.1.3.4.38 TCustomOutline.SetLevel Method

Sets the level of an outline node.

Pascal

```
procedure SetLevel(Node: TOutlineNode; CurLevel: Cardinal; NewLevel: Cardinal);
```

C++

```
__fastcall SetLevel(TOutlineNode Node, unsigned CurLevel, unsigned NewLevel);
```

Description

Call SetLevel to change the nesting depth of the node specified by the Node parameter. CurLevel is the node's current level and NewLevel is the node's new level.

For example, consider the outline specified by the following value of Lines (see page 6288):

```
NodeA
 NodeB
  NodeC
 NodeD
```

Calling SetLevel for NodeD with CurLevel set to 2 and NewLevel set to 4 results in the outline with the following value of Lines (see page 6288):

```
NodeA
 NodeB
  NodeC
   NodeD
```

That is, the parent of NodeD was changed from NodeA to NodeC.

See Also

Lines (see page 6288)

68.1.3.4.39 TCustomOutline.SetUpdateState Method

Enables or disables the maintenance of the node indexes.

Pascal

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

C++

```
__fastcall SetUpdateState(Boolean Value);
```

Description

Call SetUpdateState with Value set to true to block the recomputation of indexes. Call SetUpdateState with Value set to false to recompute all indexes in the outline and enable the maintenance of indexes with every change to the nodes in the outline.

Unlike calls to BeginUpdate (see page 6268) and EndUpdate (see page 6271), calls to SetUpdateState do not nest: Calling SetUpdateState with Value set to false always recomputes the indexes regardless of the number of previous calls to SetUpdateState with Value set to true.

Note: Calling SetUpdateState takes precedence over any enabling or disabling of index maintenance provided by BeginUpdate (see page 6268) or EndUpdate (see page 6271).

See Also

BeginUpdate (see page 6268)

EndUpdate (see page 6271)

68.1.3.5 TCustomOutline Properties

The properties of the TCustomOutline class are listed here.

Protected Properties

	Name	Description
	ItemHeight (see page 6286)	Specifies the height of each node in the outline.
	ItemSeparator (see page 6288)	Specifies a delimiting string for outline nodes.
	Lines (see page 6288)	Lists the Text property values of all nodes in the outline.
	Options (see page 6288)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 6289)	Specifies how the outline tree is displayed.
	PictureClosed (see page 6290)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 6291)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 6291)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 6291)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 6292)	Specifies the bitmap used to represent a collapsed item that has subitems.
	Style (see page 6294)	Determines whether the nodes and tree are drawn automatically.

Public Properties

	Name	Description
	Canvas (see page 6284)	Specifies the TCanvas object that presents a drawing surface for the control.
R	ItemCount (see page 6286)	Indicates the number of nodes in the outline.
R	Items (see page 6287)	Lists the nodes in the outline.
	Row (see page 6292)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 6294)	Specifies the index of the selected node.

68.1.3.5.1 TCustomOutline.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
```

```
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = dynamic_cast<TStringGrid *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
```

```
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint

68.1.3.5.2 TCustomOutline.ItemCount Property

Indicates the number of nodes in the outline.

Pascal

```
property ItemCount: LongInt;
```

C++

```
__property LongInt ItemCount;
```

Description

Read ItemCount to determine the number of nodes in the outline. Use ItemCount with the Items (see page 6287) property array to iterate over all nodes in the outline.

Note: Reading ItemCount can be computationally expensive, as it causes all the indexes used by the outline to be updated.

See Also

Items (see page 6287)

68.1.3.5.3 TCustomOutline.ItemHeight Property

Specifies the height of each node in the outline.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Set ItemHeight to the height, in pixels, of each node in the outline. Changing the ItemHeight property also changes the font used to display the nodes.

See Also

ItemCount (see page 6286)

Items (see page 6287)

68.1.3.5.4 TCustomOutline.Items Property

Lists the nodes in the outline.

Pascal

```
property Items [Index: LongInt]: TOutlineNode;
```

C++

```
__property TOutlineNode Items[LongInt Index];
```

Description

Use Items to access a node in the outline. Each item is a TOutlineNode (see page 6363) object.

Index is a number specifying the node to return, where Items[0] refers to the root, Items[1] is the first (topmost) node, and so on.

Note: Items is the default property of TCustomOutline (see page 6258). This means that the name of the property may be omitted in code. Thus the line

Note: Outline1.Items[1].Text := Edit1.Text;

Note: can also be written

Note: Outline1[1].Text := Edit1.Text;

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

ItemCount (see page 6286)

ItemHeight (see page 6286)

ItemSeparator (see page 6288)

Lines (see page 6288)

operator_[]

68.1.3.5.5 TCustomOutline.ItemSeparator Property

Specifies a delimiting string for outline nodes.

Pascal

```
property ItemSeparator: string;
```

C++

```
__property UnicodeString ItemSeparator;
```

Description

Set ItemSeparator to the string that should appear between node names in the path description of any individual node in the outline. ItemSeparator is used to implement the FullPath read method of TOutlineNode (see page 6363).

ItemSeparator is initialized to the backslash (\) character when the TCustomOutline (see page 6258) object is created.

See Also

Items (see page 6287)

FullPath (see page 6375)

68.1.3.5.6 TCustomOutline.Lines Property

Lists the Text property values of all nodes in the outline.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Read Lines to obtain a text representation of the outline. Set Lines to fill the outline from a set of text strings. Each string in the Lines property represents a node in the outline. Lines does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

The Lines property is primarily useful for filling an outline with items at design time. While Lines can be used to access the individual nodes at run time, it is much quicker to access an item with the Items (see page 6287) property.

See Also

Items (see page 6287)

Text (see page 6378)

68.1.3.5.7 TCustomOutline.Options Property

Determines how the nodes in the outline are drawn.

Pascal

```
property Options: TOutlineOptions;
```

C++

```
__property TOutlineOptions Options;
```

Description

Set Options to any combination of the following values:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf (see page 6291), PictureOpen (see page 6291), PictureClosed (see page 6290), PicturePlus (see page 6292), PictureMinus (see page 6291)) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

Items (see page 6287)

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

SelectedItem (see page 6294)

68.1.3.5.8 TCustomOutline.OutlineStyle Property

Specifies how the outline tree is displayed.

Pascal

```
property OutlineStyle: TOutlineStyle;
```

C++

```
__property TOutlineStyle OutlineStyle;
```

Description

Set OutlineStyle to one of the following values:

Value	Meaning
osText	Displays node text (specified in Text) only.
osPlusMinusText	Displays plus picture (specified in PicturePlus (see page 6292)), minus picture (specified in PictureMinus (see page 6291)), and node text (specified in Text).

osPictureText	Displays open picture (specified in PictureOpen (see page 6291)), closed picture (specified in PictureClosed (see page 6290)), leaf picture (specified in PictureLeaf (see page 6291)) and node text (specified in Text).
osPlusMinusPictureText	Displays plus picture (specified in PicturePlus (see page 6292)), minus picture (specified in PictureMinus (see page 6291)), open picture (specified in PictureOpen (see page 6291)), closed picture (specified in PictureClosed (see page 6290)), leaf picture (specified in PictureLeaf (see page 6291)), and node text (specified in Text).
osTreeText	Displays lines connecting outline nodes and node text (specified in Text).
osTreePictureText	Displays lines connecting outline nodes, open picture (specified in PictureOpen (see page 6291)), closed picture (specified in PictureClosed (see page 6290)), leaf picture (specified in PictureLeaf (see page 6291)), and node text (specified in Text).

Note: If the Style (see page 6294) property is otOwnerDraw, the value of OutlineStyle is ignored. The connecting lines, node bitmaps, and node text must all be drawn in an OnDrawItem (see page 6295) event handler.

See Also

Options (see page 6288)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

Style (see page 6294)

Text (see page 6378)

68.1.3.5.9 TCustomOutline.PictureClosed Property

Specifies the bitmap used to represent a collapsed node that has subitems.

Pascal

```
property PictureClosed: TBitmap;
```

C++

```
__property TBitmap PictureClosed;
```

Description

By default, PictureClosed is a closed file folder bitmap. Set PictureClosed to change the bitmap used for collapsed nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 6289) property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

68.1.3.5.10 TCustomOutline.PictureLeaf Property

Specifies the bitmap used to represent an item that does not have subitems.

Pascal

```
property PictureLeaf: TBitmap;
```

C++

```
__property TBitmap PictureLeaf;
```

Description

By default, PictureLeaf is a document bitmap. Set PictureLeaf to change the bitmap used for leaf nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 6289) property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureOpen (see page 6291)

68.1.3.5.11 TCustomOutline.PictureMinus Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureMinus: TBitmap;
```

C++

```
__property TBitmap PictureMinus;
```

Description

By default, PictureMinus is a minus sign (-). Set PictureMinus to change the bitmap used for collapsible nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 6289) property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 6289)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

68.1.3.5.12 TCustomOutline.PictureOpen Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureOpen: TBitmap;
```

C++

```
__property TBitmap PictureOpen;
```

Description

By default, the PictureOpen property is an open file folder. Set PictureOpen to change the bitmap used for expanded nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 6289) property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PicturePlus (see page 6292)

68.1.3.5.13 TCustomOutline.PicturePlus Property

Specifies the bitmap used to represent a collapsed item that has subitems.

Pascal

```
property PicturePlus: TBitmap;
```

C++

```
__property TBitmap PicturePlus;
```

Description

By default, the PicturePlus property is a plus sign (+). Set PicturePlus to change the bitmap used for collapsed nodes.

Note: The bitmap will not be displayed in the outline unless the OutlineStyle (see page 6289) property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureMinus (see page 6291)

68.1.3.5.14 TCustomOutline.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

C++ Examples:

```
/*
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items = StringGrid1->Rows[StringGrid1->Row];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{

  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";

}
```

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

Col (see page 3060)

68.1.3.5.15 TCustomOutline.SelectedItem Property

Specifies the index of the selected node.

Pascal

```
property SelectedItem: Longint;
```

C++

```
__property Longint SelectedItem;
```

Description

Read SelectedItem to determine the Index of the selected node. If no node is selected, SelectedItem is 0. Set SelectedItem to set focus to a particular node in the outline.

See Also

Items (see page 6287)

68.1.3.5.16 TCustomOutline.Style Property

Determines whether the nodes and tree are drawn automatically.

Pascal

```
property Style: TOutlineType;
```

C++

```
__property TOutlineType Style;
```

Description

By default, Style is osStandard, meaning that the outline displays items in the style determined by the OutlineStyle (see page 6289) property. Set Style to otOwnerDraw to create owner-draw outlines, meaning that items must be drawn on the canvas of the outline by code in an OnDrawItem (see page 6295) event handler.

Owner-draw outlines can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed (see page 6290), PictureOpen (see page 6291), PictureMinus (see page 6291), PicturePlus (see page 6292), and PictureLeaf (see page 6291) properties. Owner-draw outlines require more programming, however, as the application must specify how to render the image for each node in an OnDrawItem (see page 6295) event handler.

See Also

Canvas

OnDrawItem (see page 6295)

OutlineStyle (see page 6289)

68.1.3.6 TCustomOutline Events

The events of the TCustomOutline class are listed here.

Protected Events

	Name	Description
	OnCollapse (see page 6295)	Occurs immediately after a node in the outline is collapsed.
	OnDrawItem (see page 6295)	Occurs when a node must be drawn in an owner-draw outline.
	OnExpand (see page 6296)	Occurs immediately after a node in the outline is expanded.

68.1.3.6.1 TCustomOutline.OnCollapse Event

Occurs immediately after a node in the outline is collapsed.

Pascal

```
property OnCollapse: EOutlineChange;
```

C++

```
__property EOutlineChange OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a node in the outline is collapsed. The event handler supplies the index of the node that collapsed.

See Also

OnExpand (see page 6296)

68.1.3.6.2 TCustomOutline.OnDrawItem Event

Occurs when a node must be drawn in an owner-draw outline.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

When the Style (see page 6294) property is set to otOwnerDraw, specify an OnDrawItem event handler to render the image of each node in the outline. An OnDrawItem event handler can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed (see page 6290), PictureOpen (see page 6291), PictureMinus (see page 6291), PicturePlus (see page 6292), and PictureLeaf (see page 6291) properties.

Draw the image of the node specified by the Index parameter onto the canvas of the outline.

See Also

Canvas

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

Style (see page 6294)

Text (see page 6378)

68.1.3.6.3 TCustomOutline.OnExpand Event

Occurs immediately after a node in the outline is expanded.

Pascal

```
property OnExpand: EOutlineChange;
```

C++

```
__property EOutlineChange OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a node in the outline is expanded. The event handler identifies the node that was just expanded.

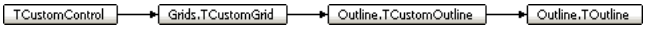
See Also

OnCollapse (see page 6295)

68.1.4 TOutline Class

TOutline is a multilevel outline that displays the values of TOutlineNode (see page 6363) objects.

Class Hierarchy



Pascal

```
TOutline = class(TCustomOutline);
```

C++

```
class TOutline : public TCustomOutline;
```

File

Outline

Description

Use an outline to visually organize information in a hierarchical tree. Each item in an outline is contained in a TOutlineNode (see page 6363) object. Outline (see page 6255) controls allow an application to manage a hierarchy of outline nodes, including the ability to control the appearance and layout of those nodes and the ability to expand or collapse branches of the outline tree.

TOutline is provided primarily for backward compatibility. New applications should use TTreeView instead.

See Also

TOutlineNode (see page 6363)

TTreeView

68.1.4.1 TOutline Members

The following tables list the members exposed by TOutline.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 3031)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 3031)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 3032)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 3032)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	AdjustSize (see page 3034)	Adds or deletes rows or columns from the grid.
	BadIndex (see page 6267)	Indicates whether a node's index is up to date.
	BeginColumnDrag (see page 3034)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 3034)	Starts the dragging of a row in the grid.
	BoxRect (see page 3035)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 3035)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 3036)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 3036)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 3036)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 3037)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 3037)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 3037)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 3038)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 3038)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 3039)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 3039)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
V	Click (see page 6268)	Updates the SelectedItem (see page 6294) property.
	Collapse (see page 6269)	Generates an OnCollapse (see page 6295) event.
	ColumnMoved (see page 3039)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 3040)	Responds when the column widths change.
V	CreateEditor (see page 3040)	Creates the inplace editor control.
V	CreateParams (see page 3041)	Initializes the window-creation parameter record.
V	DblClick (see page 6269)	Expands or collapses the selected node.
V	DefineProperties (see page 3041)	Adds the ColWidths (see page 3063) and RowHeights (see page 3076) properties to the values saved with the grid.
V	DefineProperties (see page 6269)	Creates a binary property for customized bitmap values.
V	DeleteColumn (see page 3041)	Deletes a specified column from the grid.
	DeleteNode (see page 6270)	Fixes up the outline after a node has been deleted.
V	DeleteRow (see page 3042)	Deletes a specified from the grid.
V	DoExit (see page 3042)	Hides the inplace editor when selection moves to another control in a form.
V	DoMouseWheelDown (see page 3042)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 3043)	Performs tasks when the mouse wheel is rotated upward.
A	DrawCell (see page 3043)	Provides the prototype for a method that draws the contents of a cell in the grid.
V	DrawCell (see page 6271)	Draws a node in the outline.
	EndColumnDrag (see page 3044)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 3044)	Ends the dragging of a row in the grid.
	Expand (see page 6272)	Generates an OnExpand (see page 6296) event.
	FocusCell (see page 3044)	Changes the current cell to a specified cell.
	GetEditLimit (see page 3045)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 3045)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 3046)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.

	GetEditText (see page 3046)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 3046)	Returns the value of the GridHeight (see page 3068) property.
	GetGridWidth (see page 3047)	Returns the value of the GridWidth (see page 3070) property.
	HideEdit (see page 3047)	Hides the inplace editor.
	HideEditor (see page 3048)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 3048)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCell (see page 6276)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 3048)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 3049)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 3049)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 3050)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
V	KeyDown (see page 3050)	Provides special processing when the user presses a key.
V	KeyDown (see page 6277)	Provides special processing when the user presses a key.
V	KeyPress (see page 3050)	Provides special processing when the user presses a key.
V	KeyPress (see page 6277)	Provides special processing when a key is pressed.
V	Loaded (see page 6278)	Sets up the outline after it has been streamed in.
V	MouseDown (see page 3051)	Provides special processing when the user presses a mouse button.
V	MouseDown (see page 6280)	Updates the SelectedItem (see page 6294) property.
V	MouseMove (see page 3052)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 3052)	Provides special processing when the user releases the mouse button.
	Move (see page 6280)	Repositions a node in the outline.
	MoveColRow (see page 3053)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 3053)	Moves a column to a new position.
	MoveRow (see page 3053)	Moves a row to a new position.
V	Paint (see page 3054)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 3054)	Responds when the row heights change.
	RowMoved (see page 3055)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 3055)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 3055)	Determines whether a particular cell in the grid can be selected.
	SetDisplayWidth (see page 6282)	Sets the display width of the outline.
	SetEditText (see page 3056)	Provides the interface for a method that updates the text associated with a cell.
	SetGoodIndex (see page 6282)	Returns the first node in the Items (see page 6287) array that must be reindexed to make the index of the Value parameter correct.
	SetLevel (see page 6283)	Sets the level of an outline node.
	ShowEditor (see page 3056)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 3057)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 3057)	Responds to changes in the size of the grid.
	Sizing (see page 3057)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 3058)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 3058)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 3059)	Notifies the designer when rows and columns of the grid are moved or resized.

Protected Properties

	Name	Description
	Col (see page 3060)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 3062)	Specifies the number of columns in the grid.
	ColWidths (see page 3063)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 3064)	Determines the width (in pixels) of all columns that have not been explicitly resized.

	DefaultDrawing (see page 3066)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 3066)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 3066)	Determines whether the current cell can be edited.
	FixedColor (see page 3067)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 3067)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 3068)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 3068)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 3069)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 3070)	Specifies the width of the grid in pixels.
R	HitTest (see page 3071)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 3071)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 3072)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 3074)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 3075)	Specifies the number of rows in the grid.
	RowHeights (see page 3076)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 3077)	Indicates the boundaries of the current selection.
	TabStops (see page 3079)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 3080)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 3082)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 3082)	Indicates the number of scrollable rows visible in the grid.

Public Methods

	Name	Description
	Add (see page 6264)	Adds a node to the outline.
	AddChild (see page 6265)	Adds a child node to a node in the outline.
	AddChildObject (see page 6266)	Adds a child node containing data to a node in the outline.
	AddObject (see page 6266)	Adds a node containing data to the outline.
	BeginUpdate (see page 6268)	Prevents the updating of the outline until the EndUpdate (see page 6271) method is called.
	Clear (see page 6268)	Removes all items from the outline.
V	Create (see page 3030)	Creates an instance of TCustomGrid (see page 3027).
V	Create (see page 6262)	Creates an instance of TCustomOutline (see page 6258).
	Delete (see page 6270)	Deletes a node from the outline.
V	Destroy (see page 3031)	Destroys an instance of TCustomGrid (see page 3027).
V	Destroy (see page 6263)	Destroys an instance of TCustomOutline (see page 6258).
	EndUpdate (see page 6271)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 6268).
	FullCollapse (see page 6272)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 6272)	Expands every node in the outline.
	GetDataItem (see page 6273)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetItem (see page 6273)	Returns the index of the node at the position indicated by the X and Y parameters.
	GetNodeDisplayWidth (see page 6274)	Returns the length of the row containing a specified node.
	GetTextItem (see page 6274)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetVisibleNode (see page 6274)	Returns the TOutlineNode (see page 6363) object for the node indicated by a position in the image of the outline.
	Insert (see page 6275)	Inserts a node at a specified position.
	InsertObject (see page 6275)	Inserts a node containing data at a specified position.
	LoadFromFile (see page 6278)	This is the overview for the LoadFromFile method overload.
	LoadFromStream (see page 6279)	This is the overview for the LoadFromStream method overload.
	MouseCoord (see page 3051)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	SaveToFile (see page 6280)	This is the overview for the SaveToFile method overload.
	SaveToStream (see page 6281)	This is the overview for the SaveToStream method overload.

	SetUpdateState (see page 6283)	Enables or disables the maintenance of the node indexes.

Public Properties

	Name	Description
	Canvas (see page 6284)	Specifies the TCanvas object that presents a drawing surface for the control.
R	ItemCount (see page 6286)	Indicates the number of nodes in the outline.
R	Items (see page 6287)	Lists the nodes in the outline.
	Row (see page 6292)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 6294)	Specifies the index of the selected node.

Published Events

	Name	Description
	OnClick (see page 6330)	Occurs when the user clicks the control.
	OnCollapse (see page 6332)	Occurs immediately after a node in the outline is collapsed.
	OnContextPopup (see page 6332)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 6333)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 6333)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 6336)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 6340)	Occurs when a node must be drawn in an owner-draw outline.
	OnEndDock (see page 6340)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 6341)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6342)	Occurs when a control receives the input focus.
	OnExit (see page 6344)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 6347)	Occurs immediately after a node in the outline is expanded.
	OnKeyDown (see page 6347)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6351)	Occurs when a key is pressed.
	OnKeyUp (see page 6352)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 6354)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 6355)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 6357)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 6357)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 6357)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 6360)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 6362)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 6362)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 6302)	Determines how the control aligns within its container (parent control).
	BorderStyle (see page 6305)	Determines whether a single line border is drawn around the grid.
	Color (see page 6306)	Specifies the background color of the control.
	Ctl3D (see page 6306)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 6307)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 6309)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 6309)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 6309)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 6311)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 6311)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 6313)	Specifies the height of each node in the outline.
	ItemSeparator (see page 6313)	Specifies a delimiting string for outline nodes.
	Lines (see page 6313)	Lists the Text property values of all nodes in the outline.
	Options (see page 6314)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 6315)	Specifies how the outline tree is displayed.
	ParentColor (see page 6315)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6316)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 6316)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 6316)	Determines where a control looks for its font information.
	ParentShowHint (see page 6317)	Determines where a control looks to find out if its Help Hint should be shown.
	PictureClosed (see page 6318)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 6318)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 6318)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 6319)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 6319)	Specifies the bitmap used to represent a collapsed item that has subitems.
	PopupMenu (see page 6320)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 6325)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 6325)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 6327)	Determines whether the nodes and tree are drawn automatically.
	TabOrder (see page 6327)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6328)	Determines if the user can tab to a control.
	Visible (see page 6328)	Determines whether the component appears onscreen.

68.1.4.2 TOutline Properties

The properties of the TOutline class are listed here.

Published Properties

	Name	Description
	Align (see page 6302)	Determines how the control aligns within its container (parent control).
	BorderStyle (see page 6305)	Determines whether a single line border is drawn around the grid.
	Color (see page 6306)	Specifies the background color of the control.
	Ctl3D (see page 6306)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 6307)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 6309)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 6309)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 6309)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 6311)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 6311)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 6313)	Specifies the height of each node in the outline.
	ItemSeparator (see page 6313)	Specifies a delimiting string for outline nodes.
	Lines (see page 6313)	Lists the Text property values of all nodes in the outline.
	Options (see page 6314)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 6315)	Specifies how the outline tree is displayed.
	ParentColor (see page 6315)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6316)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 6316)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 6316)	Determines where a control looks for its font information.
	ParentShowHint (see page 6317)	Determines where a control looks to find out if its Help Hint should be shown.
	PictureClosed (see page 6318)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 6318)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 6318)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 6319)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 6319)	Specifies the bitmap used to represent a collapsed item that has subitems.
	PopupMenu (see page 6320)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 6325)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 6325)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 6327)	Determines whether the nodes and tree are drawn automatically.
	TabOrder (see page 6327)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6328)	Determines if the user can tab to a control.
	Visible (see page 6328)	Determines whether the component appears onscreen.

68.1.4.2.1 TOutline.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
```

```
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  */

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
  (Form1).
    TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
  objects.
    TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

    // Line up the directory list box on the left of the form.
    pdlb->Parent = Form1;
    pdlb->Align = alLeft;
    pdlb->Width = Form1->ClientWidth/3;

    // Now use the splitter to divide the directory pane from the file pane.
    ps->Parent = Form1;
    ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
    ps->Align = pdlb->Align; // Give same alignment as directory list.

    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
  }
  /*
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  */

  #include <ComCtrls.hpp>
  #include <Controls.hpp>

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
    ProgressBar->Parent = this;
    ProgressBar->Align = alBottom;
    TButton *Button1 = new TButton(this);  // The owner will clean this up.
    Button1->Parent = this;
  }
  /*
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  */
  #include <Comctrls.hpp>
  TPageControl* ppc;
  const int MAXTABS = 3;
  TTabSheet* pts[MAXTABS];
  const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
  int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
      ppc = new TPageControl(this); // The owner (this) will clean this up.
      ppc->Parent = this;
      ppc->Align = alClient;
```

```
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
```

```
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

68.1.4.2.2 TOutline.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

68.1.4.2.3 TOutline.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

68.1.4.2.4 TOutline.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

68.1.4.2.5 TOutline.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
```

```
end;
```

See Also

DockSite

DockManager

PaintWindow

68.1.4.2.6 TOutline.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

68.1.4.2.7 TOutline.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

68.1.4.2.8 TOutline.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

68.1.4.2.9 TOutline.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

68.1.4.2.10 TOutline.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

68.1.4.2.11 TOutline.ItemHeight Property

Specifies the height of each node in the outline.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Set ItemHeight to the height, in pixels, of each node in the outline. Changing the ItemHeight property also changes the font used to display the nodes.

See Also

ItemCount (see page 6286)

Items (see page 6287)

68.1.4.2.12 TOutline.ItemSeparator Property

Specifies a delimiting string for outline nodes.

Pascal

```
property ItemSeparator: string;
```

C++

```
__property UnicodeString ItemSeparator;
```

Description

Set ItemSeparator to the string that should appear between node names in the path description of any individual node in the outline. ItemSeparator is used to implement the FullPath read method of TOutlineNode.

ItemSeparator is initialized to the backslash (\) character when the TCustomOutline object is created.

See Also

Items (see page 6287)

FullPath (see page 6375)

68.1.4.2.13 TOutline.Lines Property

Lists the Text property values of all nodes in the outline.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Read Lines to obtain a text representation of the outline. Set Lines to fill the outline from a set of text strings. Each string in the Lines property represents a node in the outline. Lines does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items.

For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

The Lines property is primarily useful for filling an outline with items at design time. While Lines can be used to access the individual nodes at run time, it is much quicker to access an item with the Items property.

See Also

Items (see page 6287)

Text (see page 6378)

68.1.4.2.14 TOutline.Options Property

Determines how the nodes in the outline are drawn.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TOutlineOptions Options;__property TGridOptions Options;
```

Description

Set Options to any combination of the following values:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

Items (see page 6287)

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

SelectedItem (see page 6294)

68.1.4.2.15 TOutline.OutlineStyle Property

Specifies how the outline tree is displayed.

Pascal

```
property OutlineStyle: TOutlineStyle;
```

C++

```
__property TOutlineStyle OutlineStyle;
```

Description

Set OutlineStyle to one of the following values:

Value	Meaning
osText	Displays node text (specified in Text) only.
osPlusMinusText	Displays plus picture (specified in PicturePlus), minus picture (specified in PictureMinus), and node text (specified in Text).
osPictureText	Displays open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf) and node text (specified in Text).
osPlusMinusPictureText	Displays plus picture (specified in PicturePlus), minus picture (specified in PictureMinus), open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf), and node text (specified in Text).
osTreeText	Displays lines connecting outline nodes and node text (specified in Text).
osTreePictureText	Displays lines connecting outline nodes, open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf), and node text (specified in Text).

Note: If the Style property is otOwnerDraw, the value of OutlineStyle is ignored. The connecting lines, node bitmaps, and node text must all be drawn in an OnDrawItem event handler.

See Also

Options (see page 6288)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

Style (see page 6294)

Text (see page 6378)

68.1.4.2.16 TOutline.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

68.1.4.2.17 TOutline.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

68.1.4.2.18 TOutline.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

68.1.4.2.19 TOutline.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

68.1.4.2.20 TOutline.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

68.1.4.2.21 TOutline.PictureClosed Property

Specifies the bitmap used to represent a collapsed node that has subitems.

Pascal

```
property PictureClosed: TBitmap;
```

C++

```
__property TBitmap PictureClosed;
```

Description

By default, PictureClosed is a closed file folder bitmap. Set PictureClosed to change the bitmap used for collapsed nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

68.1.4.2.22 TOutline.PictureLeaf Property

Specifies the bitmap used to represent an item that does not have subitems.

Pascal

```
property PictureLeaf: TBitmap;
```

C++

```
__property TBitmap PictureLeaf;
```

Description

By default, PictureLeaf is a document bitmap. Set PictureLeaf to change the bitmap used for leaf nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureOpen (see page 6291)

68.1.4.2.23 TOutline.PictureMinus Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureMinus: TBitmap;
```

C++

```
__property TBitmap PictureMinus;
```

Description

By default, PictureMinus is a minus sign (-). Set PictureMinus to change the bitmap used for collapsible nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 6289)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

68.1.4.2.24 TOutline.PictureOpen Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureOpen: TBitmap;
```

C++

```
__property TBitmap PictureOpen;
```

Description

By default, the PictureOpen property is an open file folder. Set PictureOpen to change the bitmap used for expanded nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PicturePlus (see page 6292)

68.1.4.2.25 TOutline.PicturePlus Property

Specifies the bitmap used to represent a collapsed item that has subitems.

Pascal

```
property PicturePlus: TBitmap;
```

C++

```
__property TBitmap PicturePlus;
```

Description

By default, the PicturePlus property is a plus sign (+). Set PicturePlus to change the bitmap used for collapsed nodes.

Note: The bitmap will not be displayed in the outline unless the OutlineStyle property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 6289)

PictureClosed (see page 6290)

PictureMinus (see page 6291)

68.1.4.2.26 TOutline.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
```

```
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
```

```
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
```

```
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation
```

```
{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

68.1.4.2.27 TOutline.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight

ClientWidth

68.1.4.2.28 TOutline.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

68.1.4.2.29 TOutline.Style Property

Determines whether the nodes and tree are drawn automatically.

Pascal

```
property Style: TOutlineType;
```

C++

```
__property TOutlineType Style;
```

Description

By default, Style is osStandard, meaning that the outline displays items in the style determined by the OutlineStyle property. Set Style to otOwnerDraw to create owner-draw outlines, meaning that items must be drawn on the canvas of the outline by code in an OnDrawItem event handler.

Owner-draw outlines can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed, PictureOpen, PictureMinus, PicturePlus, and PictureLeaf properties. Owner-draw outlines require more programming, however, as the application must specify how to render the image for each node in an OnDrawItem event handler.

See Also

Canvas

OnDrawItem (see page 6295)

OutlineStyle (see page 6289)

68.1.4.2.30 TOutline.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

68.1.4.2.31 TOutline.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

68.1.4.2.32 TOutline.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

68.1.4.3 TOutline Events

The events of the TOutline class are listed here.

Published Events

	Name	Description
	OnClick (see page 6330)	Occurs when the user clicks the control.
	OnCollapse (see page 6332)	Occurs immediately after a node in the outline is collapsed.
	OnContextPopup (see page 6332)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 6333)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 6333)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 6336)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 6340)	Occurs when a node must be drawn in an owner-draw outline.
	OnEndDock (see page 6340)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 6341)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6342)	Occurs when a control receives the input focus.
	OnExit (see page 6344)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 6347)	Occurs immediately after a node in the outline is expanded.
	OnKeyDown (see page 6347)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6351)	Occurs when a key is pressed.
	OnKeyUp (see page 6352)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 6354)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 6355)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 6357)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 6357)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 6357)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 6360)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 6362)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 6362)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.4.3.1 TOutline.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
```

```
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

68.1.4.3.2 TOutline.OnCollapse Event

Occurs immediately after a node in the outline is collapsed.

Pascal

```
property OnCollapse: EOutlineChange;
```

C++

```
__property EOutlineChange OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a node in the outline is collapsed. The event handler supplies the index of the node that collapsed.

See Also

OnExpand (see page 6296)

68.1.4.3.3 TOutline.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

68.1.4.3.4 TOutline.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

68.1.4.3.5 TOutline.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

68.1.4.3.6 TOutline.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

68.1.4.3.7 TOutline.OnDrawItem Event

Occurs when a node must be drawn in an owner-draw outline.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

When the Style property is set to otOwnerDraw, specify an OnDrawItem event handler to render the image of each node in the outline. An OnDrawItem event handler can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed, PictureOpen, PictureMinus, PicturePlus, and PictureLeaf properties.

Draw the image of the node specified by the Index parameter onto the canvas of the outline.

See Also

Canvas

PictureClosed (see page 6290)

PictureLeaf (see page 6291)

PictureMinus (see page 6291)

PictureOpen (see page 6291)

PicturePlus (see page 6292)

Style (see page 6294)

Text (see page 6378)

68.1.4.3.8 TOutline.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

68.1.4.3.9 TOutline.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

68.1.4.3.10 TOutline.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
```

```
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
```

```
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

68.1.4.3.11 TOutline.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

68.1.4.3.12 TOutline.OnExpand Event

Occurs immediately after a node in the outline is expanded.

Pascal

```
property OnExpand: EOutlineChange;
```

C++

```
__property EOutlineChange OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a node in the outline is expanded. The event handler identifies the node that was just expanded.

See Also

OnCollapse (see page 6295)

68.1.4.3.13 TOutline.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
```

```
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

68.1.4.3.14 TOutline.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

68.1.4.3.15 TOutline.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

68.1.4.3.16 TOutline.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

68.1.4.3.17 TOutline.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

68.1.4.3.18 TOutline.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.4.3.19 TOutline.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.4.3.20 TOutline.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

68.1.4.3.21 TOutline.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

68.1.4.3.22 TOutline.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

68.1.4.3.23 TOutline.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

68.1.5 TOutlineNode Class

TOutlineNode represents a node in a TCustomOutline (see page 6258) object.

Class Hierarchy

Pascal

```
TOutlineNode = class(TPersistent);
```

C++

```
class TOutlineNode : public TPersistent;
```

File

Outline

Description

Use TOutlineNode to describe a node in a Win 3.1 outline. Each outline node appears as a line, or row, of the outline. An outline node can have from 0 to 16368 subnodes, which are subordinate to the parent node in the outline structure.

See Also

TOutline (see page 6296)

68.1.5.1 TOutlineNode Members

The following tables list the members exposed by TOutlineNode.

Protected Methods

	Name	Description
	AddNode (see page 6366)	This is AddNode, a member of class TOutlineNode.
	GetDataItem (see page 6368)	This is GetDataItem, a member of class TOutlineNode.
	GetNodeAtIndex (see page 6370)	This is GetNodeAtIndex, a member of class TOutlineNode.
	GetRowOfNode (see page 6371)	This is GetRowOfNode, a member of class TOutlineNode.
	GetTextItem (see page 6371)	This is GetTextItem, a member of class TOutlineNode.
	GetVisibleNode (see page 6371)	This is GetVisibleNode, a member of class TOutlineNode.
	HasAsParent (see page 6372)	This is HasAsParent, a member of class TOutlineNode.
	InsertNode (see page 6372)	This is InsertNode, a member of class TOutlineNode.
	InternalRemove (see page 6372)	This is InternalRemove, a member of class TOutlineNode.
	Remove (see page 6373)	This is Remove, a member of class TOutlineNode.
	WriteNode (see page 6373)	This is the overview for the WriteNode method overload.

Protected Properties

	Name	Description
R	ExpandCount (see page 6374)	This is ExpandCount, a member of class TOutlineNode.
R	Items (see page 6377)	This is Items, a member of class TOutlineNode.
R	List (see page 6378)	This is List, a member of class TOutlineNode.
R	Outline (see page 6378)	This is Outline, a member of class TOutlineNode.

Public Methods

	Name	Description
	ChangeLevelBy (see page 6367)	Changes the level of an outline node.
	Collapse (see page 6367)	Collapses an outline item.
	Create (see page 6365)	Creates a new TOutlineNode (see page 6363) object.
V	Destroy (see page 6365)	Frees the memory associated with the TOutlineNode (see page 6363) object.
	Expand (see page 6368)	Expands an outline item.
	FullExpand (see page 6368)	Expands an outline node and all its descendants.
	GetDisplayWidth (see page 6369)	Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.
	getFirstChild (see page 6369)	This is getFirstChild, a member of class TOutlineNode.
	GetLastChild (see page 6369)	Returns the Index (see page 6376) value of the last subnode.
	GetNextChild (see page 6370)	Returns the Index (see page 6376) value of the next child of an outline node.
	GetPrevChild (see page 6370)	Returns the Index (see page 6376) value of the previous child of an outline node.
	MoveTo (see page 6372)	Moves an outline node from one location within the outline to another.

Public Properties

	Name	Description
	Data (see page 6374)	Points to application-specific data associated with the node.
	Expanded (see page 6375)	Determines whether descendants of the outline node are displayed.
R	FullPath (see page 6375)	Describes an outline node, using its text and the text of all ancestor nodes in the outline.
R	HasItems (see page 6376)	Indicates whether the outline node has any subnodes.
R	Index (see page 6376)	Identifies the node in the Items (see page 6377) array maintained by the outline.

R	IsVisible (see page 6376)	Indicates whether the outline node is visible in the outline.
	Level (see page 6377)	Indicates the level of indentation of a node within the outline.
R	Parent (see page 6378)	Identifies the immediate ancestor of the node.
	Text (see page 6378)	Specifies the string that identifies the outline node.
R	TopItem (see page 6379)	Indicates which outline node is the topmost ancestor of a node.

68.1.5.2 TOutlineNode.Create Constructor

Creates a new TOutlineNode (see page 6363) object.

Pascal

```
constructor Create(AOwner: TCustomOutline);
```

C++

```
__fastcall TOutlineNode(TCustomOutline AOwner);
```

Description

Do not call the constructor for TOutlineNode (see page 6363) directly. Instead, use one of the methods of the outline component in which the node should appear. These methods are as follows:

Method	Action
Add	Creates the node as the last sibling of a specified node.
AddObject	Same as Add, but also initializing the node's Data (see page 6374) property.
AddChild	Creates the node as the last child of a specified node.
AddChildObject	Same as AddChild, but also initializing the node's Data (see page 6374) property.
Insert	Creates the node in the current position of an existing node.
InsertObject	Same as Insert, but also initializing the node's Data (see page 6374) property.

See Also

Data (see page 6374)

Add (see page 6264)

AddObject (see page 6266)

AddChild (see page 6265)

Insert (see page 6275)

AddChildObject (see page 6266)

InsertObject (see page 6275)

68.1.5.3 TOutlineNode.Destroy Destructor

Frees the memory associated with the TOutlineNode (see page 6363) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOutlineNode();
```

Description

Do not call Destroy directly. The preferred method of deleting nodes is to use the Delete method of the containing outline component.

See Also

System::TObject::Free

Delete (see page 6270)

68.1.5.4 TOutlineNode Methods

The methods of the TOutlineNode class are listed here.

Protected Methods

	Name	Description
	AddNode (see page 6366)	This is AddNode, a member of class TOutlineNode.
	GetDataItem (see page 6368)	This is GetDataItem, a member of class TOutlineNode.
	GetNodeAtIndex (see page 6370)	This is GetNodeAtIndex, a member of class TOutlineNode.
	GetRowOfNode (see page 6371)	This is GetRowOfNode, a member of class TOutlineNode.
	GetTextItem (see page 6371)	This is GetTextItem, a member of class TOutlineNode.
	GetVisibleNode (see page 6371)	This is GetVisibleNode, a member of class TOutlineNode.
	HasAsParent (see page 6372)	This is HasAsParent, a member of class TOutlineNode.
	InsertNode (see page 6372)	This is InsertNode, a member of class TOutlineNode.
	InternalRemove (see page 6372)	This is InternalRemove, a member of class TOutlineNode.
	Remove (see page 6373)	This is Remove, a member of class TOutlineNode.
	WriteNode (see page 6373)	This is the overview for the WriteNode method overload.

Public Methods

	Name	Description
	ChangeLevelBy (see page 6367)	Changes the level of an outline node.
	Collapse (see page 6367)	Collapses an outline item.
	Expand (see page 6368)	Expands an outline item.
	FullExpand (see page 6368)	Expands an outline node and all its descendants.
	GetDisplayWidth (see page 6369)	Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.
	getFirstChild (see page 6369)	This is getFirstChild, a member of class TOutlineNode.
	GetLastChild (see page 6369)	Returns the Index (see page 6376) value of the last subnode.
	GetNextChild (see page 6370)	Returns the Index (see page 6376) value of the next child of an outline node.
	GetPrevChild (see page 6370)	Returns the Index (see page 6376) value of the previous child of an outline node.
	MoveTo (see page 6372)	Moves an outline node from one location within the outline to another.

68.1.5.4.1 TOutlineNode.AddNode Method

Pascal

```
function AddNode(Value: TOutlineNode): LongInt;
```

C++

```
__fastcall LongInt AddNode(TOutlineNode Value);
```

Description

This is AddNode, a member of class TOutlineNode.

68.1.5.4.2 TOutlineNode.ChangeLevelBy Method

Changes the level of an outline node.

Pascal

```
procedure ChangeLevelBy(Value: TChangeRange);
```

C++

```
__fastcall ChangeLevelBy(TChangeRange Value);
```

Description

Call ChangeLevelBy to move a node up or down a branch in the outline. Specify a Value parameter value of -1 to move up (toward the root) one level. Specify a Value parameter value of 1 to move down (away from the root) one level.

When moving up one level, an item becomes the next sibling of its former parent. When moving down one level, an item becomes the last child of its former prior sibling.

Nodes at the first level can not be moved up, as they have no parent. The first node in the outline can not be moved down either, as it has no prior sibling.

Note: Before calling ChangeLevelBy, be sure that the node has the appropriate number of parents or that the appropriate prior siblings exist.

See Also

Level (see page 6377)

MoveTo (see page 6372)

Parent (see page 6378)

68.1.5.4.3 TOutlineNode.Collapse Method

Collapses an outline item.

Pascal

```
procedure Collapse;
```

C++

```
__fastcall Collapse();
```

Description

Collapse sets the Expanded (see page 6375) property to false. When an outline node is collapsed, its descendants are hidden and, depending on the OutlineStyle property of the outline in which the node is contained, the plus picture or closed picture is displayed.

Collapse does nothing to nodes that have no descendants.

See Also

Expand (see page 6368)

Expanded (see page 6375)

OutlineStyle (see page 6289)

68.1.5.4.4 TOutlineNode.Expand Method

Expands an outline item.

Pascal

```
procedure Expand;
```

C++

```
__fastcall Expand();
```

Description

Expand sets the Expanded (see page 6375) property to true. When an outline node is expanded, its descendants are displayed and, depending on the OutlineStyle property of the outline in which the node is contained, the minus picture or open picture is displayed.

Expand does nothing to nodes that have no descendants.

Note: Expand only ensures that the immediate children of a node are visible. To expand all descendants of a node, use the FullExpand (see page 6368) method.

See Also

Collapse (see page 6367)

Expanded (see page 6375)

FullExpand (see page 6368)

OutlineStyle (see page 6289)

68.1.5.4.5 TOutlineNode.FullExpand Method

Expands an outline node and all its descendants.

Pascal

```
procedure FullExpand;
```

C++

```
__fastcall FullExpand();
```

Description

Use FullExpand to expand all the nodes in the subtree of the outline rooted at the outline node. FullExpand sets the Expanded (see page 6375) property of the node and all its descendants to true.

See Also

Collapse (see page 6367)

Expand (see page 6368)

Expanded (see page 6375)

68.1.5.4.6 TOutlineNode.GetDataItem Method

Pascal

```
function GetDataItem(Value: TCustomData): LongInt;
```

C++

```
__fastcall LongInt GetDataItem(TCustomData Value);
```

Description

This is GetDataItem, a member of class TOutlineNode.

68.1.5.4.7 TOutlineNode.GetDisplayWidth Method

Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

Call GetDisplayWidth to determine the length of the line in the outline that contains the outline node. GetDisplayWidth adjusts for the lines and bitmaps specified by the OutlineStyle property of the outline.

See Also

GetNodeDisplayWidth (see page 6274)

OutlineStyle (see page 6289)

68.1.5.4.8 TOutlineNode.getFirstChild Method

Pascal

```
function getFirstChild: LongInt;
```

C++

```
__fastcall LongInt getFirstChild();
```

Description

This is getFirstChild, a member of class TOutlineNode.

68.1.5.4.9 TOutlineNode.GetLastChild Method

Returns the Index (see page 6376) value of the last subnode.

Pascal

```
function GetLastChild: LongInt;
```

C++

```
__fastcall LongInt GetLastChild();
```

Description

Call GetLastChild to locate the last subnode of a node. If the node has no descendants, GetLastChild returns -1.

See Also

GetFirstChild (see page 6369)

GetNextChild (see page 6370)

GetPrevChild (see page 6370)

HasItems (see page 6376)

Index (see page 6376)

68.1.5.4.10 TOutlineNode.GetNextChild Method

Returns the Index (see page 6376) value of the next child of an outline node.

Pascal

```
function GetNextChild(Value: LongInt): LongInt;
```

C++

```
__fastcall LongInt GetNextChild(LongInt Value);
```

Description

Call GetNextChild to obtain the index of the next immediate descendant of a node, following the child node with the index given by the Value parameter. If the child node indicated by Value has no next sibling, or if the child node indicated by Value is not an immediate descendant of the outline node, GetNextChild returns -1.

Index (see page 6376) values reflect a depth-first traversal of the outline. Thus, the node returned by GetNextChild is not simply the node with an index of Value+1, unless the node with index Value has no descendants. GetNextChild allows a breadth-first traversal of the outline.

See Also

GetFirstChild (see page 6369)

GetLastChild (see page 6369)

GetPrevChild (see page 6370)

Index (see page 6376)

68.1.5.4.11 TOutlineNode.GetNodeAtIndex Method

Pascal

```
function GetNodeAtIndex(TargetIndex: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetNodeAtIndex(LongInt TargetIndex);
```

Description

This is GetNodeAtIndex, a member of class TOutlineNode.

68.1.5.4.12 TOutlineNode.GetPrevChild Method

Returns the Index (see page 6376) value of the previous child of an outline node.

Pascal

```
function GetPrevChild(Value: LongInt): LongInt;
```

C++

```
__fastcall LongInt GetPrevChild(LongInt Value);
```

Description

Call GetPrevChild to obtain the index of the previous immediate descendant of a node, preceding the child node with the index given by the Value parameter. If the child node indicated by Value has no prior sibling, or if the child node indicated by Value is not an immediate descendant of the outline node, GetPrevChild returns -1.

Index (see page 6376) values reflect a depth-first traversal of the outline. Thus, the previous sibling of the child node indicated by Value is not simply one less than Value, unless that previous sibling has no descendants.

See Also

GetFirstChild (see page 6369)

GetLastChild (see page 6369)

GetNextChild (see page 6370)

Index (see page 6376)

68.1.5.4.13 TOutlineNode.GetRowOfNode Method

Pascal

```
function GetRowOfNode(TargetNode: TOutlineNode; var RowCount: Longint): Boolean;
```

C++

```
__fastcall Boolean GetRowOfNode(TOutlineNode TargetNode, Longint RowCount);
```

Description

This is GetRowOfNode, a member of class TOutlineNode.

68.1.5.4.14 TOutlineNode.GetTextItem Method

Pascal

```
function GetTextItem(const Value: string): LongInt;
```

C++

```
__fastcall LongInt GetTextItem(const UnicodeString Value);
```

Description

This is GetTextItem, a member of class TOutlineNode.

68.1.5.4.15 TOutlineNode.GetVisibleNode Method

Pascal

```
function GetVisibleNode(TargetCount: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetVisibleNode(LongInt TargetCount);
```

Description

This is GetVisibleNode, a member of class TOutlineNode.

68.1.5.4.16 TOutlineNode.HasAsParent Method

Pascal

```
function HasAsParent(Value: TOutlineNode): Boolean;
```

C++

```
__fastcall Boolean HasAsParent(TOutlineNode Value);
```

Description

This is HasAsParent, a member of class TOutlineNode.

68.1.5.4.17 TOutlineNode.InsertNode Method

Pascal

```
function InsertNode(Index: LongInt; Value: TOutlineNode): LongInt;
```

C++

```
__fastcall LongInt InsertNode(LongInt Index, TOutlineNode Value);
```

Description

This is InsertNode, a member of class TOutlineNode.

68.1.5.4.18 TOutlineNode.InternalRemove Method

Pascal

```
procedure InternalRemove(Value: TOutlineNode; Index: Integer);
```

C++

```
__fastcall InternalRemove(TOutlineNode Value, int Index);
```

Description

This is InternalRemove, a member of class TOutlineNode.

68.1.5.4.19 TOutlineNode.MoveTo Method

Moves an outline node from one location within the outline to another.

Pascal

```
procedure MoveTo(Destination: LongInt; AttachMode: TAttachMode);
```

C++

```
__fastcall MoveTo(LongInt Destination, TAttachMode AttachMode);
```

Description

Call MoveTo to reposition the node to the position indicated by the Destination parameter. Destination is the Index (see page 6376) value of another outline node. The AttachMode parameter specifies how to attach the moved node to the destination position.

Note: To change only the level of a node, use the ChangeLevelBy (see page 6367) method.

See Also

ChangeLevelBy (see page 6367)

Index (see page 6376)

Add (see page 6264)

AddChild (see page 6265)

Insert (see page 6275)

68.1.5.4.20 TOutlineNode.Remove Method

Pascal

```
procedure Remove(Value: TOutlineNode);
```

C++

```
__fastcall Remove(TOutlineNode Value);
```

Description

This is Remove, a member of class TOutlineNode.

68.1.5.4.21 WriteNode Method

This is the overview for the WriteNode method overload.

Overload List

	Name	Description
	TOutlineNode.WriteNode (PChar, TStream) (see page 6373)	This is WriteNode, a member of class TOutlineNode.
	TOutlineNode.WriteNode (TStream, TEncoding) (see page 6373)	This is WriteNode, a member of class TOutlineNode.

68.1.5.4.21.1 TOutlineNode.WriteNode Method (PChar, TStream)

Pascal

```
procedure WriteNode(Buffer: PChar; Stream: TStream); overload; deprecated;
```

C++

```
__fastcall WriteNode(const char * Buffer, TStream Stream);
```

Description

This is WriteNode, a member of class TOutlineNode.

68.1.5.4.21.2 TOutlineNode.WriteNode Method (TStream, TEncoding)

Pascal

```
procedure WriteNode(Stream: TStream; Encoding: TEncoding); overload;
```

C++

```
__fastcall WriteNode(TStream Stream, TEncoding Encoding);
```

Description

This is WriteNode, a member of class TOutlineNode.

68.1.5.5 TOutlineNode Properties

The properties of the TOutlineNode class are listed here.

Protected Properties

	Name	Description
	ExpandCount (see page 6374)	This is ExpandCount, a member of class TOutlineNode.
	Items (see page 6377)	This is Items, a member of class TOutlineNode.
	List (see page 6378)	This is List, a member of class TOutlineNode.
	Outline (see page 6378)	This is Outline, a member of class TOutlineNode.

Public Properties

	Name	Description
	Data (see page 6374)	Points to application-specific data associated with the node.
	Expanded (see page 6375)	Determines whether descendants of the outline node are displayed.
	FullPath (see page 6375)	Describes an outline node, using its text and the text of all ancestor nodes in the outline.
	HasItems (see page 6376)	Indicates whether the outline node has any subnodes.
	Index (see page 6376)	Identifies the node in the Items (see page 6377) array maintained by the outline.
	IsVisible (see page 6376)	Indicates whether the outline node is visible in the outline.
	Level (see page 6377)	Indicates the level of indentation of a node within the outline.
	Parent (see page 6378)	Identifies the immediate ancestor of the node.
	Text (see page 6378)	Specifies the string that identifies the outline node.
	TopItem (see page 6379)	Indicates which outline node is the topmost ancestor of a node.

68.1.5.5.1 TOutlineNode.Data Property

Points to application-specific data associated with the node.

Pascal

```
property Data: TCustomData;
```

C++

```
__property TCustomData Data;
```

Description

Use Data to associate an arbitrary value with a node in an outline. For example, if the outline represents the table of contents to an electronic document, Data could point to an object that provides jump information for accessing the section of the document named by the node.

The Data property is usually described by the Text (see page 6378) property, which is displayed in the rendered image of the node.

See Also

Text (see page 6378)

68.1.5.5.2 TOutlineNode.ExpandCount Property

Pascal

```
property ExpandCount: LongInt;
```

C++

```
__property LongInt ExpandCount;
```

Description

This is ExpandCount, a member of class TOutlineNode.

68.1.5.5.3 TOutlineNode.Expanded Property

Determines whether descendants of the outline node are displayed.

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

Set Expanded to true to display all subnodes of a node. When an outline node is expanded, its descendants are displayed and, depending on the OutlineStyle property of the outline in which the node is contained, the minus picture or open picture is displayed.

Set Expanded to false to collapse the node. When an outline node is collapsed, its descendants are hidden and, depending on the OutlineStyle property of the outline in which the node is contained, the plus picture or closed picture is displayed.

Expanded is not meaningful for nodes that have no descendants.

See Also

HasItems (see page 6376)

OutlineStyle (see page 6289)

68.1.5.5.4 TOutlineNode.FullPath Property

Describes an outline node, using its text and the text of all ancestor nodes in the outline.

Pascal

```
property FullPath: string;
```

C++

```
__property UnicodeString FullPath;
```

Description

Read FullPath to obtain a description of the entire branch of the outline leading to the outline node. FullPath consists of the values of the Text (see page 6378) properties of all outline items, starting with the text of the node at the first level and traversing the outline down to the outline node. The string specified as the ItemSeparator property of the outline separates the text for each node in the path.

See Also

Level (see page 6377)

Parent (see page 6378)

ItemSeparator (see page 6288)

Text (see page 6378)

TopItem (see page 6379)

68.1.5.5.5 TOutlineNode.HasItems Property

Indicates whether the outline node has any subnodes.

Pascal

```
property HasItems: Boolean;
```

C++

```
__property Boolean HasItems;
```

Description

Read HasItem to determine whether the outline node is a leaf node. Depending on the OutlineStyle property of the outline that contains the node, leaf nodes may appear with the leaf bitmap. If the node is not a leaf, subnodes appear below and indented from the node when it is expanded.

HasItem returns true if the node has any descendants, even if the node is collapsed and the descendants are not visible. HasItem returns false if the node is a leaf node.

See Also

Expanded (see page 6375)

Parent (see page 6378)

OutlineStyle (see page 6289)

68.1.5.5.6 TOutlineNode.Index Property

Identifies the node in the Items (see page 6377) array maintained by the outline.

Pascal

```
property Index: LongInt;
```

C++

```
__property LongInt Index;
```

Description

Read Index to determine the row of the outline where the node appears. The first node has an Index value of 1, and subsequent nodes are indexed sequentially. If a node has subnodes, the Index value of the parent node is one less than the Index value of its first descendant.

When nodes in the outline are added, moved, or deleted, the values of the Index properties of all subsequent nodes must be recalculated to be valid. Sometimes, the recalculation of indexes is deferred until all changes to the nodes are complete. To ensure that the value of Index is valid, call the SetUpdateState method of the outline that contains the node.

See Also

Items (see page 6287)

SetUpdateState (see page 6283)

68.1.5.5.7 TOutlineNode.IsVisible Property

Indicates whether the outline node is visible in the outline.

Pascal

```
property IsVisible: Boolean;
```

C++

```
__property Boolean IsVisible;
```

Description

Read IsVisible to determine whether the node is part of the current image of the outline. A node is visible if it is on level 1 or if all its parents are expanded.

Note: Do not confuse IsVisible with the Visible property of TControl objects. Outline (see page 6378) nodes are not controls, but merely helper objects that describe nodes for an outline control. IsVisible indicates whether a node is visible because its parents are expanded or collapsed. It does not provide information about whether the node has been scrolled out of view, or even if the outline is not visible.

See Also

Expanded (see page 6375)

Visible

68.1.5.5.8 TOutlineNode.Items Property

Pascal

```
property Items [Index: LongInt]: TOutlineNode;
```

C++

```
__property TOutlineNode Items[LongInt Index];
```

Description

This is Items, a member of class TOutlineNode.

68.1.5.5.9 TOutlineNode.Level Property

Indicates the level of indentation of a node within the outline.

Pascal

```
property Level: Cardinal;
```

C++

```
__property unsigned Level;
```

Description

The value of Level is 1 for nodes that directly descend from the root of the outline (the top level). The value of Level is 2 for their children, and so on.

See Also

ChangeLevelBy (see page 6367)

Parent (see page 6378)

TopItem (see page 6379)

68.1.5.5.10 TOutlineNode.List Property

Pascal

```
property List: TList;
```

C++

```
__property TList List;
```

Description

This is List, a member of class TOutlineNode.

68.1.5.5.11 TOutlineNode.Outline Property

Pascal

```
property Outline: TCustomOutline;
```

C++

```
__property TCustomOutline Outline;
```

Description

This is Outline, a member of class TOutlineNode.

68.1.5.5.12 TOutlineNode.Parent Property

Identifies the immediate ancestor of the node.

Pascal

```
property Parent: TOutlineNode;
```

C++

```
__property TOutlineNode Parent;
```

Description

Read Parent to find the parent of an outline node. A parent node is one level higher and contains the child node as a subnode.

See Also

Level (see page 6377)

TopItem (see page 6379)

68.1.5.5.13 TOutlineNode.Text Property

Specifies the string that identifies the outline node.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text to determine the string that is displayed in the outline to represent the node.

See Also

FullPath (see page 6375)

68.1.5.5.14 TOutlineNode.TopItem Property

Indicates which outline node is the topmost ancestor of a node.

Pascal

```
property TopItem: Longint;
```

C++

```
__property Longint TopItem;
```

Description

Read TopItem to determine the first node in the branch of the outline tree that contains the outline node. The TopItem property specifies the Index (see page 6376) value of the level 1 ancestor of the outline node. For a node on level 1, TopItem has the same value as the Index (see page 6376) property.

See Also

Index (see page 6376)

Level (see page 6377)

Parent (see page 6378)

68.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAttachMode (see page 6380)	TAttachMode indicates how an outline node should be inserted into an outline.
	TOutlineBitmap (see page 6381)	This is record Outline.TOutlineBitmap.
	TOutlineNodeCompare (see page 6381)	This is record Outline.TOutlineNodeCompare.
	TOutlineOption (see page 6382)	TOutlineOptions (see page 6387) represents the style in which an outline is drawn.
	TOutlineStyle (see page 6382)	TOutlineStyle determines how the nodes of an outline are drawn.
	TOutlineType (see page 6383)	TOutlineType indicates whether an outline is owner-drawn.

68.2.1 Outline.TAttachMode Enumeration

TAttachMode indicates how an outline node should be inserted into an outline.

Pascal

```
TAttachMode = (
  oaAdd,
  oaAddChild,
  oaInsert
);
```

C++

```
enum TAttachMode {
  oaAdd,
  oaAddChild,
  oaInsert
};
```

File

Outline

Description

TAttachMode is used by methods that move outline nodes within an outline. It includes the following values:

Value	Description
oaAdd	The item is attached as if added with the Add method. The moved item becomes the last sibling of the item specified by the Destination parameter, sharing the same parent as the Destination item.
oaAddChild	The item is attached as if added with the AddChild method. The moved item becomes the last child of the item specified by the Destination parameter. The Destination item becomes the parent of the moved item.
oaInsert	The item is attached as if inserted with the Insert method. The moved item replaces the Destination item in the outline, while the Destination item and all other following items are moved down one row.

See Also

TCustomOutline

TOutlineNode

68.2.2 Outline.TOutlineBitmap Enumeration

Pascal

```
TOutlineBitmap = (
  obPlus,
  obMinus,
  obOpen,
  obClose,
  obLeaf
);
```

C++

```
enum TOutlineBitmap {
  obPlus,
  obMinus,
  obOpen,
  obClose,
  obLeaf
};
```

File

Outline

Description

This is record Outline.TOutlineBitmap.

68.2.3 Outline.TOutlineNodeCompare Enumeration

Pascal

```
TOutlineNodeCompare = (
  ocLess,
  ocSame,
  ocGreater,
  ocInvalid
);
```

C++

```
enum TOutlineNodeCompare {
  ocLess,
  ocSame,
  ocGreater,
  ocInvalid
};
```

File

Outline

Description

This is record Outline.TOutlineNodeCompare.

68.2.4 Outline.TOutlineOption Enumeration

TOutlineOptions (see page 6387) represents the style in which an outline is drawn.

Pascal

```
TOutlineOption = (
  ooDrawTreeRoot,
  ooDrawFocusRect,
  ooStretchBitmaps
);
```

C++

```
enum TOutlineOption {
  ooDrawTreeRoot,
  ooDrawFocusRect,
  ooStretchBitmaps
};
```

File

Outline

Description

TOutlineOptions (see page 6387) is a set of TOutlineOption values. Together they influence the way an outline is drawn. Each TOutlineOption value is described below:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

TCustomOutline (see page 6258)

68.2.5 Outline.TOutlineStyle Enumeration

TOutlineStyle determines how the nodes of an outline are drawn.

Pascal

```
TOutlineStyle = (
  osText,
  osPlusMinusText,
  osPictureText,
  osPlusMinusPictureText,
  osTreeText,
  osTreePictureText
);
```

C++

```
enum TOutlineStyle {
  osText,
  osPlusMinusText,
  osPictureText,
  osPlusMinusPictureText,
  osTreeText,
  osTreePictureText
};
```

File

Outline

Description

TOutlineStyle is the type of TCustomOutline (see page 6258).::OutlineStyle. It includes the following values:

Value	Meaning
osText	Only the text of nodes is drawn.
osPlusMinusText	In addition to the text, each node displays plus picture (when it can be expanded) or minus picture (when it can be collapsed).
osPictureText	In addition to the text, each node displays the open picture (when it is expanded), closed picture (when it is collapsed), or leaf picture (when it has no subnodes).
osPlusMinusPictureText	In addition to the text, each node displays the plus picture or minus picture (as in osPlusMinusText), and the open, closed or leaf picture (as in osPictureText).
osTreeText	Visible lines connect the outline nodes.
osTreePictureText	In addition to the text, each node displays the open picture (when it is expanded), closed picture (when it is collapsed), or leaf picture (when it has no subnodes). Visible lines connect the outline nodes.

See Also

TCustomOutline

68.2.6 Outline.TOutlineType Enumeration

TOutlineType indicates whether an outline is owner-drawn.

Pascal

```
TOutlineType = (
  otStandard,
  otOwnerDraw
);
```

C++

```
enum TOutlineType {
  otStandard,
  otOwnerDraw
};
```

File

Outline

Description

TOutlineType determines whether an outline is drawn automatically or by user code in an OnDrawItem event.

Value	Meaning
otStandardItems	Items are drawn automatically.
otOwnerDraw	Items are drawn on the Canvas by application code.

See Also

TCustomOutline

TCustomOutline

68.3 Types

The following table lists types in this documentation.

Types

Name	Description
EOutlineChange (see page 6385)	EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.
TBitmapArray (see page 6385)	This is type Outline.TBitmapArray.
TBitmapArrayRange (see page 6386)	This is type Outline.TBitmapArrayRange.
TChangeRange (see page 6386)	TChangeRange indicates the direction in which a node is moved.
TOutlineBitmaps (see page 6386)	This is type Outline.TOutlineBitmaps.
TOutlineOptions (see page 6387)	TOutlineOptions represents the style in which an outline is drawn.

68.3.1 Outline.EOutlineChange Type

EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.

Pascal

```
EOutlineChange = procedure (Sender: TObject; Index: LongInt) of object;
```

C++

```
(Sender: TObject; Index: LongInt) ( EOutlineChange)();
```

File

Outline

Description

Sender is the outline control whose node just changed.

Index is the index of the node that just changed.

68.3.2 Outline.TBitmapArray Type

Pascal

```
TBitmapArray = array[TBitmapArrayRange] of TBitmap;
```

C++

```
array[TBitmapArrayRange] of TBitmap TBitmapArray;
```

File

Outline

Description

This is type Outline.TBitmapArray.

68.3.3 Outline.TBitmapArrayRange Type

Pascal

```
TBitmapArrayRange = 0..4;
```

C++

```
0..4 TBitmapArrayRange;
```

File

Outline

Description

This is type Outline.TBitmapArrayRange.

68.3.4 Outline.TChangeRange Type

TChangeRange indicates the direction in which a node is moved.

Pascal

```
TChangeRange = -1..1;
```

C++

```
-1..1 TChangeRange;
```

File

Outline

Description

TChangeRange is the type of the Value parameter of an outline node's ChangeLevelBy method. It can take one of the following values:

Value	Meaning
-1	Move the node toward the root by one level.
1	Move the node toward the leaf by one level.

See Also

TOutlineNode (see page 6363)

68.3.5 Outline.TOutlineBitmaps Type

Pascal

```
TOutlineBitmaps = set of TOutlineBitmap;
```

C++

```
TOutlineBitmap TOutlineBitmaps;
```

File

Outline

Description

This is type Outline.TOutlineBitmaps.

68.3.6 Outline.TOutlineOptions Type

TOutlineOptions represents the style in which an outline is drawn.

Pascal

```
TOutlineOptions = set of TOutlineOption;
```

C++

```
TOutlineOption TOutlineOptions;
```

File

Outline

Description

TOutlineOptions is a set of TOutlineOption (see page 6382) values. Together they influence the way an outline is drawn. Each TOutlineOption (see page 6382) value is described below:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

TCustomOutline

69 PagItems Namespace

This is namespace PagItems.

Topics

Name	Description
Classes (see page 6390)	The following table lists classes in this documentation.

Classes

	Name	Description
	TBasePageItemsProducer (see page 6390)	TBasePageItemsProducer produces a string of HTML commands based on an input template.
	TPageItemsProducer (see page 6394)	TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

69.1 Classes

The following table lists classes in this documentation.

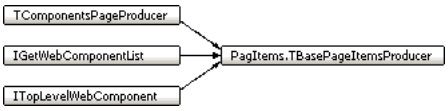
Classes

	Name	Description
	TBasePageItemsProducer (see page 6390)	TBasePageItemsProducer produces a string of HTML commands based on an input template.
	TPageItemsProducer (see page 6394)	TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

69.1.1 TBasePageItemsProducer Class

TBasePageItemsProducer produces a string of HTML commands based on an input template.

Class Hierarchy



Pascal

```
TBasePageItemsProducer = class(TComponentsPageProducer, IGetWebComponentList,
ITopLevelWebComponent);
```

C++

```
class TBasePageItemsProducer : public TComponentsPageProducer, public IGetWebComponentList,
public ITopLevelWebComponent;
```

File

PagItems

Description

Use TBasePageItemsProducer to convert an HTML template that includes content generated by other components into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the components page producer in an OnHTMLTag event handler.

TBasePageItemsProducer differs from other page producer components in that it replaces special HTML-transparent tags of the format

```
<#COMPONENT Name=WebContentComponent>
```

with the HTML produced by the components named WebContentComponent. In addition, TBasePageItemsProducer stores a default value for WebContentComponent as the value of its WebPageItems (see page 6393) property.

See Also

IWebContent

TPageItemsProducer (see page 6394)

TInetXPageProducer (see page 5730)

TAdapterPageProducer

69.1.1.1 TBasePageItemsProducer Members

The following tables list the members exposed by TBasePageItemsProducer.

Protected Methods

	Name	Description
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.

Public Methods

	Name	Description
	Create (see page 6391)	Creates an instance of a TBasePageItemsProducer (see page 6390).
	Destroy (see page 6391)	Destroys the instance of the TBasePageItemsProducer (see page 6390).

Public Properties

	Name	Description
	WebPageItems (see page 6393)	Maintains a list of components that generate HTML content for the producer.

69.1.1.2 TBasePageItemsProducer.Create Constructor

Creates an instance of a TBasePageItemsProducer (see page 6390).

Pascal

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

C++

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

Description

Call Create to instantiate a TBasePageItemsProducer (see page 6390) component at runtime if it was not created at design time.

AOwner is the component, typically the Web module, that becomes the value of this page producer's Owner property. The Owner is responsible for freeing the TBasePageItemsProducer (see page 6390) instance.

See Also

Owner

69.1.1.3 TBasePageItemsProducer.Destroy Destructor

Destroys the instance of the TBasePageItemsProducer (see page 6390).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasePageItemsProducer();
```

Description

Do not call Destroy directly. Instead call Free, which checks that the TBasePageItemsProducer (see page 6390) reference is not nil before calling Destroy.

See Also

Create (see page 6391)

System::TObject::Free

69.1.1.4 TBasePageItemsProducer Methods

The methods of the TBasePageItemsProducer class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.

69.1.1.4.1 TBasePageItemsProducer.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TBasePageItemsProducer.

69.1.1.4.2 TBasePageItemsProducer.GetComponentList Method

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

This is GetComponentList, a member of class TBasePageItemsProducer.

69.1.1.4.3 TBasePageItemsProducer.GetDefaultComponentList Method

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

This is GetDefaultComponentList, a member of class TBasePageItemsProducer.

69.1.1.4.4 TBasePageItemsProducer.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

This is SetChildOrder, a member of class TBasePageItemsProducer.

69.1.1.5 TBasePageItemsProducer Properties

The properties of the TBasePageItemsProducer class are listed here.

Public Properties

	Name	Description
	WebPageItems (see page 6393)	Maintains a list of components that generate HTML content for the producer.

69.1.1.5.1 TBasePageItemsProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems: TWebComponentList;
```

C++

```
__property TWebComponentList WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer (see page 6390). Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent

Creating Web Pages with an InternetExpress Page Producer

69.1.2 TPageItemsProducer Class

TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

Class Hierarchy

Pascal

```
TPageItemsProducer = class(TBasePageItemsProducer);
```

C++

```
class TPageItemsProducer : public TBasePageItemsProducer;
```

File

PagItems

Description

Use TPageItemsProducer as the base class for components that convert an HTML template that includes content generated by other components into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page items producer in an OnHTMLTag event handler.

Page item producers differ from other page producer components in that they replace special HTML-transparent tags of the format

```
<#COMPONENT Name=WebContentComponent>
```

with the HTML produced by the component named WebContentComponent. WebContentComponent must have the same Owner as the page item producer and support the IWebContent interface, which the page item producer uses to obtain the replacement HTML for the COMPONENT tag.

TPageItemsProducer introduces the WebPageItems (see page 6393) property to maintain a hierarchical list of these Web content components, where each component produces content for its parent component.

See Also

IWebContent

TInetXPageProducer (see page 5730)

69.1.2.1 TPageItemsProducer Members

The following tables list the members exposed by TPageItemsProducer.

Protected Methods

	Name	Description
	GetChildren (see page 6392)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 6392)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetDefaultComponentList (see page 6392)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.

	GetTemplateStream (see page 6396)	This is GetTemplateStream, a member of class TPageItemsProducer.
	SetChildOrder (see page 6393)	This is SetChildOrder, a member of class TBasePageItemsProducer.

Public Methods

	Name	Description
	Create (see page 6395)	Creates and initializes an instance of TPageItemsProducer (see page 6394).
	Destroy (see page 6395)	Frees an instance of TPageItemsProducer (see page 6394).

Public Properties

	Name	Description
	HTMLDoc (see page 6396)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 6397)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	WebPageItems (see page 6393)	Maintains a list of components that generate HTML content for the producer.

69.1.2.2 TPageItemsProducer.Create Constructor

Creates and initializes an instance of TPageItemsProducer (see page 6394).

Pascal

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

C++

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

Description

Call Create to instantiate a page items producer at runtime.

69.1.2.3 TPageItemsProducer.Destroy Destructor

Frees an instance of TPageItemsProducer (see page 6394).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPageItemsProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the TPageItemsProducer (see page 6394) reference is not nil before calling Destroy.

See Also

System::TObject::Free

69.1.2.4 TPageItemsProducer Methods

The methods of the TPageItemsProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateStream (see page 6396)	This is GetTemplateStream, a member of class TPageItemsProducer.

69.1.2.4.1 TPageItemsProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TPageItemsProducer.

69.1.2.5 TPageItemsProducer Properties

The properties of the TPageItemsProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 6396)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 6397)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

69.1.2.5.1 TPageItemsProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile (see page 6397) property.

See Also

Content (see page 3466)

HTMLFile (see page 6397)

69.1.2.5.2 TPageItemsProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 6396) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 3466)

HTMLDoc (see page 6396)

70 PlatformDefaultStyleActnCtrls Namespace

This is namespace PlatformDefaultStyleActnCtrls.

Topics

Name	Description
Classes (see page 6400)	The following table lists classes in this documentation.
Variables (see page 6403)	The following table lists variables in this documentation.

Classes

	Name	Description
	TPlatformDefaultStyleActionBars (see page 6400)	This is class PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.

Variables

Name	Description
PlatformDefaultStyle (see page 6403)	This is variable PlatformDefaultStyleActnCtrls.PlatformDefaultStyle.

70.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TPlatformDefaultStyleActionBars (see page 6400)	This is class PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.

70.1.1 TPlatformDefaultStyleActionBars Class

Class Hierarchy

Pascal

```
TPlatformDefaultStyleActionBars = class(TActionBarStyleEx);
```

C++

```
class TPlatformDefaultStyleActionBars : public TActionBarStyleEx;
```

File

PlatformDefaultStyleActnCtrls

Description

This is class PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.

70.1.1.1 TPlatformDefaultStyleActionBars Members

The following tables list the members exposed by TPlatformDefaultStyleActionBars.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 6401)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 6401)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 6401)	Returns the control class type on an action bar item.
	GetPopupClass (see page 6402)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 6402)	This is GetScrollBtnClass, a member of class TPlatformDefaultStyleActionBars.
	GetStyleName (see page 6402)	Returns the style associated with action manager.

70.1.1.2 TPlatformDefaultStyleActionBars Methods

The methods of the TPlatformDefaultStyleActionBars class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 6401)	Returns the base class for ActionBand menus for customizing action bands.

	GetColorMapClass (see page 6401)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 6401)	Returns the control class type on an action bar item.
	GetPopupClass (see page 6402)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 6402)	This is GetScrollBtnClass, a member of class TPlatformDefaultStyleActionBars.
	GetStyleName (see page 6402)	Returns the style associated with action manager.

70.1.1.2.1 TPlatformDefaultStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

70.1.1.2.2 TPlatformDefaultStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

70.1.1.2.3 TPlatformDefaultStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

70.1.1.2.4 TPlatformDefaultStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

70.1.1.2.5 TPlatformDefaultStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TPlatformDefaultStyleActionBars.

70.1.1.2.6 TPlatformDefaultStyleActionBars.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetStyleName();
```

Description

Returns the value of the Style property.

70.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
PlatformDefaultStyle (see page 6403)	This is variable PlatformDefaultStyleActnCtrls.PlatformDefaultStyle.

70.2.1 PlatformDefaultStyleActnCtrls.PlatformDefaultStyle Variable

Pascal

```
PlatformDefaultStyle: TPlatformDefaultStyleActionBars;
```

C++

```
TPlatformDefaultStyleActionBars PlatformDefaultStyle;
```

File

PlatformDefaultStyleActnCtrls

Description

This is variable PlatformDefaultStyleActnCtrls.PlatformDefaultStyle.

71 Printers Namespace

This is namespace Printers.

Topics

Name	Description
Classes (see page 6406)	The following table lists classes in this documentation.
Functions (see page 6434)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6438)	The following table lists structs, records, enums in this documentation.
Types (see page 6440)	The following table lists types in this documentation.

Classes

	Name	Description
	EPrinter (see page 6406)	EPrinter is the exception class for printing errors.
	TPrinter (see page 6406)	TPrinter encapsulates the Windows printer interface.

Functions

	Name	Description
	AssignPrn (see page 6434)	Assigns a text file variable to the printer.
	Printer (see page 6434)	Returns a global instance of TPrinter (see page 6406) to manage interaction with the printer.
	SetPrinter (see page 6436)	Replaces the global instance of TPrinter (see page 6406) that manages interaction with the printer.

Structs, Records, Enums

	Name	Description
	TPrinterCapability (see page 6438)	TPrinterCapability and TPrinterCapabilities (see page 6440) specify the capabilities of a printer device driver.
	TPrinterOrientation (see page 6438)	TPrinterOrientation indicates the orientation of pages when printing.
	TPrinterState (see page 6439)	This is record Printers.TPrinterState.

Types

Name	Description
TPrinterCapabilities (see page 6440)	This is type Printers.TPrinterCapabilities.

71.1 Classes

The following table lists classes in this documentation.

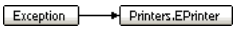
Classes

	Name	Description
	EPrinter (see page 6406)	EPrinter is the exception class for printing errors.
	TPrinter (see page 6406)	TPrinter encapsulates the Windows printer interface.

71.1.1 EPrinter Class

EPrinter is the exception class for printing errors.

Class Hierarchy



Pascal

```
EPrinter = class(Exception);
```

C++

```
class EPrinter : public Exception;
```

File

Printers

Description

EPrinter is raised when a printing error occurs. For example, if an application attempts to use a printer that doesn't exist, or if a job can't be sent to the printer for some other reason, an EPrinter exception is raised.

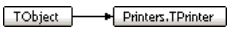
See Also

TPrinter (see page 6406)

71.1.2 TPrinter Class

TPrinter encapsulates the Windows printer interface.

Class Hierarchy



Pascal

```
TPrinter = class(TObject);
```

C++

```
class TPrinter : public TObject;
```

File

Printers

Description

Use TPrinter to manage any printing performed by an application. Obtain an instance of TPrinter by calling the Printer (see page 6434) function in the Printers (see page 6430) unit.

To determine how the printed image of the form appears, use the PrintScale property of a TForm (see page 2241) component.

A print job is started by a call to BeginDoc (see page 6411). The application sends commands by rendering through a Text variable or the printer's canvas. You can move to a new page by calling the NewPage (see page 6415) method. The job stays open until the application calls EndDoc (see page 6413). If a problem occurs and you need to terminate a print job that was not sent to the printer successfully, call the Abort (see page 6409) method.

Use the TPrinter properties to configure the print job. For example, the title displayed in the Print Manager (and on network header pages) is determined by the Title (see page 6433) property. Copies (see page 6422) determines the number of copies to print, and Orientation (see page 6424) lets you specify whether to print in portrait or landscape mode.

TPrinter includes several read-only properties as well, that let you determine which page is currently being printed, the fonts available on the printer, the paper size, and so on.

When creating a TPrinter descendant, you must call the SetPrinter (see page 6418) routine in order for the TPrinter descendant object to work correctly.

See Also

Printer (see page 6434)

TCustomForm (see page 2103)

SetPrinter Routine

71.1.2.1 TPrinter Members

The following tables list the members exposed by TPrinter.

Public Methods

	Name	Description
	Abort (see page 6409)	Terminates the printing of a print job, dropping all unprinted data.
	BeginDoc (see page 6411)	Sends a print job to the printer.
	Create (see page 6408)	Creates a printer.
V	Destroy (see page 6408)	Destroys an instance of a printer.
	EndDoc (see page 6413)	Ends the current print job and closes the text file variable.
	GetPrinter (see page 6415)	Retrieves information about the current printer.
	NewPage (see page 6415)	Starts a new page and increments the PageNumber (see page 6427) property.
	Refresh (see page 6417)	Updates the Fonts (see page 6422) and Printers (see page 6430) properties.
	SetPrinter (see page 6418)	Specifies the current printer.

Public Properties

	Name	Description
R	Aborted (see page 6419)	Determines if the user aborted the print job.
R	Canvas (see page 6421)	Represents the surface of the currently printing page.
R	Capabilities (see page 6421)	Indicates the current settings of a printer device driver.
	Copies (see page 6422)	Specifies the number of copies to print.
R	Fonts (see page 6422)	Lists the fonts supported by the current printer.

R	Handle (see page 6422)	Provides access to the handle of the printer object.
	Orientation (see page 6424)	Determines if the print job prints vertically or horizontally on a page.
R	PageHeight (see page 6425)	Indicates the height (in pixels) of the currently printing page.
R	PageNumber (see page 6427)	Indicates the number of the page currently printing.
R	PageWidth (see page 6428)	Indicates the value of width (in pixels) of the currently printing page.
	PrinterIndex (see page 6430)	Specifies which printer listed in the Printers (see page 6430) property is the currently selected printer.
R	Printers (see page 6430)	Lists all printers installed in Windows.
R	Printing (see page 6430)	Indicates whether a print job is printing.
	Title (see page 6433)	Determines the text that is listed in the Print Manager and on network header pages.

71.1.2.2 TPrinter.Create Constructor

Creates a printer.

Pascal

```
constructor Create;
```

C++

```
__fastcall TPrinter();
```

Description

Never call Create directly. Create is called automatically by the Printer (see page 6434) function. Create allocates memory for a printer and calls the inherited Create. Then it sets the correct driver, device, and port.

See Also

System::TObject::Create

71.1.2.3 TPrinter.Destroy Destructor

Destroys an instance of a printer.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPrinter();
```

Description

Do not call Destroy directly. The printer object is automatically created and destroyed.

When Destroy is called, it verifies that any currently printing job has ended, then frees resources such as the handle, fonts, and canvas, closes the printer, and calls the inherited Destroy method.

See Also

System::TObject::Destroy

System::TObject::Free

71.1.2.4 TPrinter Methods

The methods of the TPrinter class are listed here.

Public Methods

	Name	Description
	Abort (see page 6409)	Terminates the printing of a print job, dropping all unprinted data.
	BeginDoc (see page 6411)	Sends a print job to the printer.
	EndDoc (see page 6413)	Ends the current print job and closes the text file variable.
	GetPrinter (see page 6415)	Retrieves information about the current printer.
	NewPage (see page 6415)	Starts a new page and increments the PageNumber (see page 6427) property.
	Refresh (see page 6417)	Updates the Fonts (see page 6422) and Printers (see page 6430) properties.
	SetPrinter (see page 6418)	Specifies the current printer.

71.1.2.4.1 TPrinter.Abort Method

Terminates the printing of a print job, dropping all unprinted data.

Pascal

```
procedure Abort;
```

C++

```
__fastcall Abort();
```

Description

Use Abort to terminate the print job before it completes; otherwise, use the EndDoc (see page 6413) method.

When Abort is called, the device is then set for the next print job.

C++ Examples:

```
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

Aborted (see page 6419)

BeginDoc (see page 6411)

EndDoc (see page 6413)

Printing (see page 6430)

71.1.2.4.2 TPrinter.BeginDoc Method

Sends a print job to the printer.

Pascal

```
procedure BeginDoc;
```

C++

```
__fastcall BeginDoc();
```

Description

Call BeginDoc to initiate a print job. If the print job is sent successfully, the application calls EndDoc (see page 6413) to end the print job. The print job won't actually start printing until EndDoc (see page 6413) is called.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
```

```
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
```

```
        BeginDoc;
        for I := Start to Stop do
        begin
          PageControl1.Pages[I].PaintTo(Handle, 10, 10);
          if I <> Stop then
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

Abort (see page 6409)

EndDoc (see page 6413)

Printing (see page 6430)

71.1.2.4.3 TPrinter.EndDoc Method

Ends the current print job and closes the text file variable.

Pascal

```
procedure EndDoc;
```

C++

```
__fastcall EndDoc();
```

Description

EndDoc terminates the print job (and closes the currently open Text). The print job will begin printing on the printer after a call to EndDoc.

After the application calls EndDoc, the printer begins printing. Use EndDoc after successfully sending a print job to the printer. If the print job isn't successful, use the Abort (see page 6409) method.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
```

```
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
```

```
      { now, print the pages }
      with Printer do
      begin
        BeginDoc;
        for I := Start to Stop do
        begin
          PageControl1.Pages[I].PaintTo(Handle, 10, 10);
          if I <> Stop then
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

Abort (see page 6409)

BeginDoc (see page 6411)

Printing (see page 6430)

71.1.2.4.4 TPrinter.GetPrinter Method

Retrieves information about the current printer.

Pascal

```
procedure GetPrinter(ADevice: PChar; ADriver: PChar; APort: PChar; var ADeviceMode: THandle);
```

C++

```
__fastcall GetPrinter(const char * ADevice, const char * ADriver, const char * APort, THandle
ADeviceMode);
```

Description

Call GetPrinter to obtain information about the current printer.

ADevice is a buffer that receives the device name of the current printer. This is the name that appears in the Print Manager (for example, "Epson FX-80"). It is not the printer model name.

ADriver is a buffer that receives the driver name for the current printer. Some versions of Windows do not use printer driver names, in which case this buffer is unused. On versions of Windows that support printer driver names, this is the name of a printer driver (usually 'WINSPOOL').

APort is a buffer that receives the port used by the current printer. It is not used on 32-bit versions of Windows, but is included for backward compatibility.

DeviceMode returns the handle to a memory buffer containing device-specific initialization data for the device driver. Use the GlobalLock function to convert this handle into a pointer to a DeviceMode record (Delphi) or DEVMODE structure (C++).

See Also

Printers (see page 6430)

71.1.2.4.5 TPrinter.NewPage Method

Starts a new page and increments the PageNumber (see page 6427) property.

Pascal

```
procedure NewPage;
```

C++

```
__fastcall NewPage();
```

Description

NewPage forces the current print job to begin printing on a new page in the printer.

NewPage increments the value of the PageNumber (see page 6427) property. It also resets the value of the Pen property of the Canvas (see page 6421) back to (0, 0).

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
```

```
  print dialog is displayed.  The user can select any subset
  of the pages in the page control for printing.  The selected
  pages are then printed.  To run this example successfully,
  you must add the Printers unit to the uses clause of your
  unit.  Right click on the PageControl to add at least one
  page to the PageControl.
  }
  procedure TForm1.Button1Click(Sender:TObject);
  var
    I, Start, Stop: Integer;
  begin
  PrintDialog1.Options := [poPageNums, poSelection];
  PrintDialog1.FromPage := 1;
  PrintDialog1.MinPage := 1;
  PrintDialog1.ToPage := PageControl1.PageCount;
  PrintDialog1.MaxPage := PageControl1.PageCount;

  if PrintDialog1.Execute() then
    begin
      { determine the range the user wants to print }
      with PrintDialog1 do
      begin
        if PrintRange = prAllPages then
          begin
          Start := MinPage - 1;
          Stop := MaxPage - 1;
        end
        else if PrintRange = prSelection then
        begin
          Start := PageControl1.ActivePage.PageIndex;
          Stop := Start;
          end
        else  { PrintRange = prPageNums }
        begin
          Start := FromPage - 1;
          Stop := ToPage - 1;
        end;
      end;
      { now, print the pages }
      with Printer do
      begin
        BeginDoc;
        for I := Start to Stop do
        begin
          PageControl1.Pages[I].PaintTo(Handle, 10, 10);
          if I <> Stop then
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

PageNumber (see page 6427)

71.1.2.4.6 TPrinter.Refresh Method

Updates the Fonts (see page 6422) and Printers (see page 6430) properties.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the installed fonts or printers on the current system may have changed. Refresh clears the internal cache of installed fonts and printers, so that the Fonts (see page 6422) and Printers (see page 6430) properties are updated when they are next read.

See Also

Fonts (see page 6422)

Printers (see page 6430)

71.1.2.4.7 TPrinter.SetPrinter Method

Specifies the current printer.

Pascal

```
procedure SetPrinter(ADevice: PChar; ADriver: PChar; APort: PChar; ADeviceMode: THandle);
```

C++

```
__fastcall SetPrinter(const char * ADevice, const char * ADriver, const char * APort, THandle
ADeviceMode);
```

Description

Call SetPrinter to change the current printer.

ADevice specifies the device name of the new printer. This is the name that appears in the Print Manager (for example, "Epson FX-80"). It is not the printer model name.

ADriver specifies the driver name for the new printer (usually 'WINSPOOL').

APort is included for backward compatibility. It should be set to an empty string.

DeviceMode is the handle to a memory buffer device-specific initialization data for the device driver. This handle can be obtained by calling the GlobalAlloc function to allocate a DeviceMode record (Delphi) or DEVMODE structure (C++).

See Also

Printers (see page 6430)

71.1.2.5 TPrinter Properties

The properties of the TPrinter class are listed here.

Public Properties

	Name	Description
R	Aborted (see page 6419)	Determines if the user aborted the print job.
R	Canvas (see page 6421)	Represents the surface of the currently printing page.
R	Capabilities (see page 6421)	Indicates the current settings of a printer device driver.
	Copies (see page 6422)	Specifies the number of copies to print.
R	Fonts (see page 6422)	Lists the fonts supported by the current printer.
R	Handle (see page 6422)	Provides access to the handle of the printer object.
	Orientation (see page 6424)	Determines if the print job prints vertically or horizontally on a page.
R	PageHeight (see page 6425)	Indicates the height (in pixels) of the currently printing page.

R	PageNumber (see page 6427)	Indicates the number of the page currently printing.
R	PageWidth (see page 6428)	Indicates the value of width (in pixels) of the currently printing page.
	PrinterIndex (see page 6430)	Specifies which printer listed in the Printers (see page 6430) property is the currently selected printer.
R	Printers (see page 6430)	Lists all printers installed in Windows.
R	Printing (see page 6430)	Indicates whether a print job is printing.
	Title (see page 6433)	Determines the text that is listed in the Print Manager and on network header pages.

71.1.2.5.1 TPrinter.Aborted Property

Determines if the user aborted the print job.

Pascal

```
property Aborted: Boolean;
```

C++

```
__property Boolean Aborted;
```

Description

If Aborted is true, the print job was aborted. If it is false, the user did not abort the print job.

The Abort (see page 6409) method sets the Aborted property to true.

C++ Examples:

```
/*
The following code displays a dialog box if the print job
was aborted.  This code aborts a print job if the user
presses Esc. Note that you should set KeyPreview to true
to ensure that the OnKeyDown event handler of Form1 is called.
Also notice that the Button1Click routine continues to run
even after the print job has been aborted.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 140; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
  if (Printer()->Aborted)
    MessageDlg(
```

```
      "The print job did not finish printing",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  KeyPreview = True;
}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }

}
```

Delphi Examples:

```
{
The following code displays a dialog box if the print job
was aborted.  This code aborts a print job if the user
presses Esc. Note that you should set KeyPreview to true
to ensure that the OnKeyDown event handler of Form1 is called.
Also notice that the Button1Click routine continues to run
even after the print job has been aborted.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
  if Printer.Aborted then
    MessageDlg('The print job did not finish printing', mtInformation, [mbOK], 0);
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
//  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

Abort (see page 6409)

71.1.2.5.2 TPrinter.Canvas Property

Represents the surface of the currently printing page.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas represents the printing surface of a page. Use the Brush, Font, and Pen properties of the Canvas object to determine how drawing or text appears on the page.

Note:

See Also

Brush (see page 2716)

CopyRect (see page 2683)

Draw (see page 2685)

Font (see page 2721)

Pen (see page 2725)

StretchDraw (see page 2704)

Handle (see page 2722)

71.1.2.5.3 TPrinter.Capabilities Property

Indicates the current settings of a printer device driver.

Pascal

```
property Capabilities: TPrinterCapabilities;
```

C++

```
__property TPrinterCapabilities Capabilities;
```

Description

Capabilities contains the capabilities for the printer, indicating the orientation, number of copies, and whether or not collated is selected.

71.1.2.5.4 TPrinter.Copies Property

Specifies the number of copies to print.

Pascal

```
property Copies: Integer;
```

C++

```
__property int Copies;
```

Description

Set Copies to indicate the number of copies to print.

Copies uses the device mode to determine the number of copies to print.

71.1.2.5.5 TPrinter.Fonts Property

Lists the fonts supported by the current printer.

Pascal

```
property Fonts: TStrings;
```

C++

```
__property TStrings Fonts;
```

Description

Read Fonts to get a list of the fonts supported by the printer. Each string in Fonts is the name of a typeface.

Fonts contains TrueType fonts even if the printer doesn't support them natively because the Windows Graphics (see page 2619) Device Interface (GDI) can draw TrueType fonts accurately when a print job uses them.

See Also

Name (see page 2743)

71.1.2.5.6 TPrinter.Handle Property

Provides access to the handle of the printer object.

Pascal

```
property Handle: HDC;
```

C++

```
__property HDC Handle;
```

Description

Use Handle to gain access to the handle of the printer object.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
```

```
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
```

```
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

71.1.2.5.7 TPrinter.Orientation Property

Determines if the print job prints vertically or horizontally on a page.

Pascal

```
property Orientation: TPrinterOrientation;
```

C++

```
__property TPrinterOrientation Orientation;
```

Description

Use Orientation to determine if a print job prints in landscape or portrait.

C++ Examples:

```
/*
This example uses two radio buttons on a form named
Landscape and Portrait. The form also includes a button.
When the user selects an orientation by clicking one of the
radio buttons and then clicks the button to print one line
of text, the print job prints using the selected orientation:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Printer()->BeginDoc();
  Printer()->Canvas->TextOut(100,100,"Hi there!");
```

```
  Printer()->EndDoc();
}

void __fastcall TForm1::PortraitClick(TObject *Sender)
{
  Printer()->Orientation = poPortrait;
}

void __fastcall TForm1::LandscapeClick(TObject *Sender)
{
  Printer()->Orientation = poLandscape;
}
```

Delphi Examples:

```
{
This example uses two radio buttons on a form named
Landscape and Portrait. The form also includes a button.
When the user selects an orientation by clicking one of the
radio buttons and then clicks the button to print one line
of text, the print job prints using the selected orientation:
}
uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Printer.BeginDoc;
  Printer.Canvas.TextOut(100,100,'Hi there');
  Printer.EndDoc;
end;

procedure TForm1.LandscapeClick(Sender: TObject);
begin
  Printer.Orientation := poLandscape;
end;

procedure TForm1.PortraitClick(Sender: TObject);
begin
  Printer.Orientation := poPortrait;
end;
```

71.1.2.5.8 TPrinter.PageHeight Property

Indicates the height (in pixels) of the currently printing page.

Pascal

```
property PageHeight: Integer;
```

C++

```
__property int PageHeight;
```

Description

Use PageHeight to find the height in pixels of the currently printing page.

C++ Examples:

```
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

PageNumber (see page 6427)

PageWidth (see page 6428)

71.1.2.5.9 TPrinter.PageNumber Property

Indicates the number of the page currently printing.

Pascal

```
property PageNumber: Integer;
```

C++

```
__property int PageNumber;
```

Description

Use PageNumber to find the page number of the currently printing page.

PageNumber is incremented whenever the NewPage (see page 6415) method is called. It is also incremented when a Text variable is written and a CR is encountered on the last line of the page.

C++ Examples:

```
/*
This example uses a button and a status bar on a form. When
the user clicks the button, one line of text is printed on
six separate pages. As each page is printed, a message
indicating the number of the page being printed appears on
the status bar.  Notice that the SimplePanel property of the
```

```
status bar to must be set to true.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  StatusBar1->SimplePanel = True; // so that SimpleText will work
  Printer()->BeginDoc();
  for (int i = 1;i < 7;i++)
  {
    Printer()->Canvas->TextOut(100, 100, "C++Builder rules!");
    StatusBar1->SimpleText = AnsiString("Printing page ") + Printer()->PageNumber;
    Printer()->NewPage();
  }
  Printer()->EndDoc();
}
```

Delphi Examples:

```
{
This example uses a button and a status bar on a form. When
the user clicks the button, one line of text is printed on
six separate pages. As each page is printed, a message
indicating the number of the page being printed appears on
the status bar.  To run this example successfully, you must
add Printers to the uses clause of your unit and set the
SimplePanel property of the status bar to true.
}
uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  StatusBar1.SimplePanel := True; { so that SimpleText will work }
  Printer.BeginDoc;
  for I := 1 to 6 do
  begin
    Printer.Canvas.TextOut(100, 100, 'Object Pascal is great');
    StatusBar1.SimpleText := 'Now printing page ' + IntToStr(Printer.PageNumber);
    Printer.NewPage;
  end;
  Printer.EndDoc;

end;
```

See Also

NewPage (see page 6415)

PageHeight (see page 6425)

PageWidth (see page 6428)

71.1.2.5.10 TPrinter.PageWidth Property

Indicates the value of width (in pixels) of the currently printing page.

Pascal

```
property PageWidth: Integer;
```

C++

```
__property int PageWidth;
```

Description

Use PageWidth to find the width in pixels of the currently printing page.

C++ Examples:

```
/*
This example uses a button and a memo on a form. When the
user clicks the button, the content of the memo is printed,
with a 200-pixel border around the page.
Note: The TMemo will wrap the lines of the text files, so to
see the TextRect clip, the memo width must be large and the
lines in the text file must be long.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Memo1->Width = 1000;
  Memo1->Lines->LoadFromFile("..\\readme.txt");
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Printer()->BeginDoc();
  int margin = 1000;
  int X = margin;
  int Y = margin;
  int I = 0;
  while(Y < Printer()->PageHeight)
  {
    Canvas->TextRect(Rect(X, Y, Printer()->PageWidth-margin, Printer()->PageHeight-margin),
                     X, Y, Memo1->Lines->Strings[I]);
    Y = Y + 100;
    I = I + 1;
  }
  Printer()->EndDoc();
}
```

Delphi Examples:

```
{
This example uses a button and a memo on a form. When the
user clicks the button, the content of the memo is printed,
with a 200-pixel border around the page. To run this example
successfully, include <Printers.hpp> in your unit file.
Note: The TMemo will wrap the lines of the text files, so to
see the TextRect clip, the memo width must be large and the
lines in the text file must be long.
}
uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I, margin: Integer;
begin
  with Printer do
  begin
    BeginDoc;
    margin := 1000;
    X := margin;
    Y := margin;
    I := 0;
    while(Y < PageHeight) do
    begin
      Canvas.TextRect(Rect(X, Y, PageWidth-margin, PageHeight-margin),
                      X, Y, Memo1.Lines[I]);
      Y := Y + 100;
      I := I + 1;
    end;
    EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Memo1.Width := 1000;
  Memo1.Lines.LoadFromFile('readme.txt');
end;
```

See Also

PageHeight (see page 6425)

PageNumber (see page 6427)

71.1.2.5.11 TPrinter.PrinterIndex Property

Specifies which printer listed in the Printers (see page 6430) property is the currently selected printer.

Pascal

```
property PrinterIndex: Integer;
```

C++

```
__property int PrinterIndex;
```

Description

The value of PrinterIndex is the currently selected printer. If this value is changed, EndDoc (see page 6413) is called automatically. To select the default printer, set the value of PrinterIndex to -1.

The list of installed printers is found in the Printers (see page 6430) property. The list of fonts supported by the current printer is found in the Fonts (see page 6422) property.

See Also

EndDoc (see page 6413)

Fonts (see page 6422)

Printers (see page 6430)

71.1.2.5.12 TPrinter.Printers Property

Lists all printers installed in Windows.

Pascal

```
property Printers: TStrings;
```

C++

```
__property TStrings Printers;
```

Description

The list of installed printers is found in the Printers property. The value of the PrinterIndex (see page 6430) property is the currently selected printer. The list of fonts supported by the current printer is found in the Fonts (see page 6422) property.

See Also

GetPrinter (see page 6415)

71.1.2.5.13 TPrinter.Printing Property

Indicates whether a print job is printing.

Pascal

```
property Printing: Boolean;
```

C++

```
__property Boolean Printing;
```

Description

Use Printing to determine if a job is printing.

Printing is true when the application has called the BeginDoc (see page 6411) method but the EndDoc (see page 6413) method (or the Abort (see page 6409) method) hasn't yet been called.

C++ Examples:

```
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
```

```
  }
}
```

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

Abort (see page 6409)

BeginDoc (see page 6411)

EndDoc (see page 6413)

71.1.2.5.14 TPrinter.Title Property

Determines the text that is listed in the Print Manager and on network header pages.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Title is used by Windows in the Print Manager and for network title pages.

71.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AssignPrn (see page 6434)	Assigns a text file variable to the printer.
	Printer (see page 6434)	Returns a global instance of TPrinter (see page 6406) to manage interaction with the printer.
	SetPrinter (see page 6436)	Replaces the global instance of TPrinter (see page 6406) that manages interaction with the printer.

71.2.1 Printers.AssignPrn Function

Assigns a text file variable to the printer.

Pascal

```
procedure AssignPrn(var F: Text);
```

C++

```
AssignPrn(Text F);
```

File

Printers

Description

In Delphi code, Call AssignPrn to cause all information written to the text file variable F to be directed to the printer. After the variable is assigned, the application must call the Rewrite procedure.

See Also

Rewrite

71.2.2 Printers.Printer Function

Returns a global instance of TPrinter (see page 6406) to manage interaction with the printer.

Pascal

```
function Printer: TPrinter;
```

C++

```
TPrinter Printer();
```

File

Printers

Description

Printer creates an instance of a global TPrinter (see page 6406) object the first time it is called. Use the Printer function to print

using the TPrinter (see page 6406) object.

Printer is declared in the Printers (see page 6405) unit. To use the Printer function and the TPrinter (see page 6406) object, add Printers (see page 6405) to the uses clause of your unit (Delphi) or include Printers.hpp in the source file (C++).

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
```

```
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

SetPrinter (see page 6436)

71.2.3 Printers.SetPrinter Function

Replaces the global instance of TPrinter (see page 6406) that manages interaction with the printer.

Pascal

```
function SetPrinter(NewPrinter: TPrinter): TPrinter;
```

C++

```
TPrinter SetPrinter(TPrinter NewPrinter);
```

File

Printers

Description

SetPrinter replaces the global TPrinter (see page 6406) object with another TPrinter (see page 6406) object. This allows applications to change the way they handle printing by substituting a TPrinter (see page 6406) descendant for the default TPrinter (see page 6406) object.

NewPrinter is the new TPrinter (see page 6406) object that replaces the current global TPrinter (see page 6406) object.

SetPrinter returns the previous global TPrinter (see page 6406) object.

Note: The global TPrinter (see page 6406) object is freed automatically when the application shuts down. After a call to SetPrinter, the printer that is returned is not automatically freed. It is the caller's responsibility to either free the return value, or replace it using another call to SetPrinter and to free the substitute printer that the second SetPrinter call returns.

SetPrinter is declared in the Printers (see page 6405) unit. To use the SetPrinter function, add Printers (see page 6405) to the uses clause of your unit (Delphi) or include Printers.hpp in the source file (C++) .

See Also

Printer (see page 6434)

71.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPrinterCapability (see page 6438)	TPrinterCapability and TPrinterCapabilities (see page 6440) specify the capabilities of a printer device driver.
	TPrinterOrientation (see page 6438)	TPrinterOrientation indicates the orientation of pages when printing.
	TPrinterState (see page 6439)	This is record Printers.TPrinterState.

71.3.1 Printers.TPrinterCapability Enumeration

TPrinterCapability and TPrinterCapabilities (see page 6440) specify the capabilities of a printer device driver.

Pascal

```
TPrinterCapability = (
  pcCopies,
  pcOrientation,
  pcCollation
);
```

C++

```
enum TPrinterCapability {
  pcCopies,
  pcOrientation,
  pcCollation
};
```

File

Printers

Description

TPrinterCapabilities (see page 6440) is a set of TPrinterCapability values. The possible values of TPrinterCapability are

Value	Meaning
pcCopies	The printer driver allows the user to specify the number of copies.
pcOrientation	The printer driver allows the user to specify portrait or landscape orientation.
pcCollation	The printer driver allows the user to specify whether copies should be collated.

71.3.2 Printers.TPrinterOrientation Enumeration

TPrinterOrientation indicates the orientation of pages when printing.

Pascal

```
TPrinterOrientation = (
```

```
    poPortrait,
    poLandscape
  );
```

C++

```
  enum TPrinterOrientation {
    poPortrait,
    poLandscape
  };
```

File

Printers

Description

These are the possible values for TPrinterOrientation:

Value	Meaning
poPortrait	The job prints vertically on the page.
poLandscape	The job prints horizontally on the page.

71.3.3 Printers.TPrinterState Enumeration

Pascal

```
  TPrinterState = (
    psNoHandle,
    psHandleIC,
    psHandleDC
  );
```

C++

```
  enum TPrinterState {
    psNoHandle,
    psHandleIC,
    psHandleDC
  };
```

File

Printers

Description

This is record Printers.TPrinterState.

71.4 Types

The following table lists types in this documentation.

Types

Name	Description
TPrinterCapabilities (see page 6440)	This is type Printers.TPrinterCapabilities.

71.4.1 Printers.TPrinterCapabilities Type

Pascal

```
TPrinterCapabilities = set of TPrinterCapability;
```

C++

```
TPrinterCapability TPrinterCapabilities;
```

File

Printers

Description

This is type Printers.TPrinterCapabilities.

72 Provider Namespace

This is namespace Provider.

Topics

Name	Description
Classes (see page 6443)	The following table lists classes in this documentation.
Interfaces (see page 6580)	The following table lists interfaces in this documentation.
Functions (see page 6582)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6584)	The following table lists structs, records, enums in this documentation.
Types (see page 6589)	The following table lists types in this documentation.
Variables (see page 6595)	The following table lists variables in this documentation.
Constants (see page 6596)	The following table lists constants in this documentation.

Classes

	Name	Description
	EDSWriter (see page 6443)	EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.
	TBaseProvider (see page 6445)	TBaseProvider is a base class for all providers that represent the data in a dataset.
	TCustomPacketWriter (see page 6460)	This is class Provider.TCustomPacketWriter.
	TCustomProvider (see page 6464)	TCustomProvider is the base class for data providers.
	TCustomResolver (see page 6491)	TCustomResolver is the base class for components that apply updates on behalf of a provider.
	TDataPacketWriter (see page 6501)	This is class Provider.TDataPacketWriter.
	TDataSetProvider (see page 6510)	TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.
	TDataSetResolver (see page 6540)	TDataSetResolver applies updates to a dataset on behalf of a provider.
	TLocalAppServer (see page 6546)	This is class Provider.TLocalAppServer.
	TPacketDataSet (see page 6555)	TPacketDataSet represents a data packet or delta packet that is used by a provider.
	TProvider (see page 6563)	TProvider provides data to and applies updates from a client dataset.
	TSQLResolver (see page 6564)	TSQLResolver applies updates to a database server on behalf of a provider.
	TUpdateTree (see page 6572)	TUpdateTree stores information about a delta packet on behalf of a resolver.

Constants

Name	Description
MetaDataOption (see page 6596)	Specifies the options available when retrieving metadata from a target database.

ResetOption (see page 6597)	This is constant Provider.ResetOption.
XMLOption (see page 6597)	This is constant Provider.XMLOption.
XMLUTF8Option (see page 6598)	This is constant Provider.XMLUTF8Option.

Functions

	Name	Description
	GetObjectProperty (see page 6582)	This is function Provider.GetObjectProperty.
	GetStringProperty (see page 6582)	This is function Provider.GetStringProperty.
	VarArrayFromStrings (see page 6582)	Creates a Variant array that contains the strings in a TStrings object.
	VarArrayFromWideStrings (see page 6583)	This is function Provider.VarArrayFromWideStrings.

Interfaces

	Name	Description
	IProviderContainer (see page 6580)	This is class Provider.IProviderContainer.

Structs, Records, Enums

	Name	Description
	TGetRecordOption (see page 6584)	TGetRecordOption and TGetRecordOptions (see page 6591) influence what type of information is added to data packets.
	TProviderOption (see page 6585)	TProviderOption and TProviderOptions (see page 6593) influence what type of information is communicated between client datasets and provider components.
	TPutFieldInfo (see page 6587)	This is record Provider.TPutFieldInfo.
	TResolverResponse (see page 6587)	TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Types

Name	Description
PPutFieldInfo (see page 6589)	This is type Provider.PPutFieldInfo.
TAfterUpdateRecordEvent (see page 6589)	This is type Provider.TAfterUpdateRecordEvent.
TBeforeCommitEvent (see page 6590)	This is type Provider.TBeforeCommitEvent.
TBeforeUpdateRecordEvent (see page 6590)	This is type Provider.TBeforeUpdateRecordEvent.
TDataRequestEvent (see page 6590)	This is type Provider.TDataRequestEvent.
TGetDSProps (see page 6591)	This is type Provider.TGetDSProps.
TGetParamsEvent (see page 6591)	This is type Provider.TGetParamsEvent.
TGetRecordOptions (see page 6591)	This is type Provider.TGetRecordOptions.
TGetTableNameEvent (see page 6592)	This is type Provider.TGetTableNameEvent.
TInfoArray (see page 6592)	This is type Provider.TInfoArray.
TOnValidate (see page 6592)	This is type Provider.TOnValidate.
TProviderDataEvent (see page 6593)	This is type Provider.TProviderDataEvent.
TProviderOptions (see page 6593)	This is type Provider.TProviderOptions.
TPutFieldProc (see page 6593)	This is type Provider.TPutFieldProc.
TResolverErrorEvent (see page 6594)	This is type Provider.TResolverErrorEvent.

Variables

Name	Description
InformixLob (see page 6595)	This is variable Provider.InformixLob.

72.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDSWriter (see page 6443)	EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.
	TBaseProvider (see page 6445)	TBaseProvider is a base class for all providers that represent the data in a dataset.
	TCustomPacketWriter (see page 6460)	This is class Provider.TCustomPacketWriter.
	TCustomProvider (see page 6464)	TCustomProvider is the base class for data providers.
	TCustomResolver (see page 6491)	TCustomResolver is the base class for components that apply updates on behalf of a provider.
	TDataPacketWriter (see page 6501)	This is class Provider.TDataPacketWriter.
	TDataSetProvider (see page 6510)	TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.
	TDataSetResolver (see page 6540)	TDataSetResolver applies updates to a dataset on behalf of a provider.
	TLocalAppServer (see page 6546)	This is class Provider.TLocalAppServer.
	TPacketDataSet (see page 6555)	TPacketDataSet represents a data packet or delta packet that is used by a provider.
	TProvider (see page 6563)	TProvider provides data to and applies updates from a client dataset.
	TSQLResolver (see page 6564)	TSQLResolver applies updates to a database server on behalf of a provider.
	TUpdateTree (see page 6572)	TUpdateTree stores information about a delta packet on behalf of a resolver.

72.1.1 EDSWriter Class

EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.

Class Hierarchy

Pascal

```
EDSWriter = class(Exception);
```

C++

```
class EDSWriter : public Exception;
```

File

Provider

Description

EDSWriter is raised whenever a provider component encounters an error while trying to create a data packet from a dataset.

See Also

TDataSetProvider (see page 6510)

TCustomProvider (see page 6464)

72.1.1.1 EDSWriter Members

The following tables list the members exposed by EDSWriter.

Public Methods

	Name	Description
	Create (see page 6444)	Creates an instance of EDSWriter (see page 6443) using a specified error code and message string.

Public Properties

	Name	Description
	ErrorCode (see page 6444)	Indicates the error code that this exception represents.

72.1.1.2 EDSWriter.Create Constructor

Creates an instance of EDSWriter (see page 6443) using a specified error code and message string.

Pascal

```
constructor Create(ErrMsg: string; Status: Integer);
```

C++

```
__fastcall EDSWriter(UnicodeString ErrMsg, int Status);
```

Description

Create instantiates EDSWriter (see page 6443), setting the Message property to ErrMsg and the ErrorCode (see page 6444) property to Status.

See Also

Create

ErrorCode (see page 6444)

72.1.1.3 EDSWriter Properties

The properties of the EDSWriter class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 6444)	Indicates the error code that this exception represents.

72.1.1.3.1 EDSWriter.ErrorCode Property

Indicates the error code that this exception represents.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

Use ErrorCode to determine what error gave rise to this exception. ErrorCode is the value passed to the canonical EDSWriter (

see page 6443) constructor (C++) or Create (see page 6444) method (Delphi).

See Also

Create (see page 6444)

72.1.2 TBaseProvider Class

TBaseProvider is a base class for all providers that represent the data in a dataset.

Class Hierarchy

Pascal

```
TBaseProvider = class(TCustomProvider);
```

C++

```
class TBaseProvider : public TCustomProvider;
```

File

Provider

Description

TBaseProvider is the immediate ancestor of TDataSetProvider (see page 6510), which provides data from any dataset and applies updates back to that dataset or directly to an SQL server. Use TBaseProvider to derive other customized data provider objects, such as providers that apply updates directly to a database without using SQL.

Unlike its immediate ancestor (TCustomProvider (see page 6464)) TBaseProvider includes no abstract (Delphi) or pure virtual (C++) methods. However, applications should not create instances of TBaseProvider, as many of the methods are merely stubs that provide a backstop for specialized descendants that do not implement all the provider methods. To add a provider component to a form or data module, use TDataSetProvider (see page 6510) (or TXMLTransformProvider).

A provider derived from TBaseProvider is an intermediate link between a TClientDataSet or TXMLBroker object and a source dataset. The source dataset can reside in the same application as the client dataset or XML broker, or it can be part of a separate application server. The provider packages data from the source dataset and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database or source dataset.

Client datasets and XML brokers communicate with a provider through the IAppServer (see page 5222) interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 6482) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet

TDataSetProvider (see page 6510)

IAppServer (see page 5222)

TCustomProvider (see page 6464)

TXMLTransformProvider

TSoapDataModule

72.1.2.1 TBaseProvider Members

The following tables list the members exposed by TBaseProvider.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 6483)	Occurs after the provider applies updates.
	AfterExecute (see page 6484)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 6485)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 6485)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 6486)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 6487)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 6487)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 6488)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 6488)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 6490)	Occurs before the provider creates a delta packet with information about the current record.
	OnValidate (see page 6491)	This is OnValidate, a member of class TCustomProvider.

Protected Methods

	Name	Description
	CheckResolver (see page 6449)	Instantiates a resolver for the Resolver (see page 6456) property if it does not already exist.
V	CreateDataPacket (see page 6449)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 6450)	Creates a resolver component for handling the application of updates.
V	DoAfterApplyUpdates (see page 6469)	Generates an AfterApplyUpdates (see page 6483) event.
V	DoAfterExecute (see page 6469)	Generates an AfterExecute (see page 6484) event.
V	DoAfterGetParams (see page 6470)	Generates an AfterGetParams (see page 6485) event.
V	DoAfterGetRecords (see page 6470)	Generates an AfterGetRecords (see page 6485) event.
V	DoAfterRowRequest (see page 6471)	Generates an AfterRowRequest (see page 6486) event.
V	DoAfterUpdateRecord (see page 6450)	Generates an AfterUpdateRecord (see page 6458) event.
V	DoApplyUpdates (see page 6471)	This is DoApplyUpdates, a member of class TCustomProvider.
V	DoBeforeApplyUpdates (see page 6471)	Generates a BeforeApplyUpdates (see page 6487) event.
V	DoBeforeCommit (see page 6450)	This is DoBeforeCommit, a member of class TBaseProvider.
V	DoBeforeExecute (see page 6472)	Generates a BeforeExecute (see page 6487) event.
V	DoBeforeGetParams (see page 6472)	Generates a BeforeGetParams (see page 6488) event.
V	DoBeforeGetRecords (see page 6473)	Generates a BeforeGetRecords (see page 6488) event.
V	DoBeforeRowRequest (see page 6474)	Generates a BeforeRowRequest (see page 6490) event.
V	DoBeforeUpdateRecord (see page 6451)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.
V	DoGetRecords (see page 6474)	This is DoGetRecords, a member of class TCustomProvider.
V	DoOnGetData (see page 6451)	Generates an OnGetData (see page 6459) event.
V	DoOnUpdateData (see page 6451)	Generates an OnUpdateData (see page 6459) event.
V	FetchDetails (see page 6451)	Fills a delta dataset with all nested dataset field values.
	FreeResolver (see page 6452)	Frees the resolver specified by the Resolver (see page 6456) property.

	InternalApplyUpdates (see page 6452)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.
	InternalExecute (see page 6478)	Provides the underlying implementation of the Execute (see page 6474) method.
	InternalGetParams (see page 6478)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 6453)	Provides the underlying implementation of the GetRecords (see page 6476) method.
	InternalRowRequest (see page 6454)	Provides the underlying implementation of the RowRequest (see page 6480) method.
	LocateRecord (see page 6454)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	SetActiveUpdateException (see page 6481)	This is SetActiveUpdateException, a member of class TCustomProvider.
	UpdateRecord (see page 6455)	Retrieves specified data from a source dataset into a delta packet.

Protected Properties

	Name	Description
	ActiveUpdateException (see page 6482)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 6483)	This is OwnerData, a member of class TCustomProvider.
	PacketDataSet (see page 6456)	This is PacketDataSet, a member of class TBaseProvider.

Public Events

	Name	Description
	AfterUpdateRecord (see page 6458)	Occurs when the associated resolver has just updated a record.
	BeforeUpdateRecord (see page 6458)	This is BeforeUpdateRecord, a member of class TBaseProvider.
	OnDataRequest (see page 6458)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 6459)	Occurs after a provider retrieves data but before it is sent to the client.
	OnUpdateData (see page 6459)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 6460)	Occurs when a provider can't update a record because of an error condition.

Public Methods

	Name	Description
	ApplyUpdates (see page 6468)	This is the overview for the ApplyUpdates method overload.
	Create (see page 6447)	Creates and initializes a TBaseProvider (see page 6445) object.
	DataRequest (see page 6468)	Generates an OnDataRequest (see page 6490) event.
	Destroy (see page 6448)	Frees an instance of a provider object.
	Execute (see page 6474)	Executes a query, stored procedure, or custom SQL statement.
	GetParams (see page 6475)	Returns the current parameter values.
	GetRecords (see page 6476)	This is the overview for the GetRecords method overload.
	RowRequest (see page 6480)	Returns information from a specified record.

Public Properties

	Name	Description
	Data (see page 6482)	Denotes a data packet created by the provider.
	Exported (see page 6482)	Allows client applications to pass calls to the provider when it resides on an application server.
	Options (see page 6455)	Influences what is included in data packets and how the information in data packets is used.
	Resolver (see page 6456)	Provides access to the resolver component that applies updates and resolves update errors.
	UpdateMode (see page 6457)	Controls how the provider locates records when applying updates.

72.1.2.2 TBaseProvider.Create Constructor

Creates and initializes a TBaseProvider (see page 6445) object.

Pascal

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

C++

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

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

AOwner specifies the component, typically a form or data module, that is responsible for freeing the provider.

See Also

Owner

72.1.2.3 TBaseProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the provider is not nil before calling the Destroy method.

See Also

System::TObject::Free

Resolver (see page 6456)

72.1.2.4 TBaseProvider Methods

The methods of the TBaseProvider class are listed here.

Protected Methods

	Name	Description
	CheckResolver (see page 6449)	Instantiates a resolver for the Resolver (see page 6456) property if it does not already exist.
V	CreateDataPacket (see page 6449)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 6450)	Creates a resolver component for handling the application of updates.
V	DoAfterUpdateRecord (see page 6450)	Generates an AfterUpdateRecord (see page 6458) event.
V	DoBeforeCommit (see page 6450)	This is DoBeforeCommit, a member of class TBaseProvider.
V	DoBeforeUpdateRecord (see page 6451)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.
V	DoOnGetData (see page 6451)	Generates an OnGetData (see page 6459) event.
V	DoOnUpdateData (see page 6451)	Generates an OnUpdateData (see page 6459) event.
V	FetchDetails (see page 6451)	Fills a delta dataset with all nested dataset field values.
	FreeResolver (see page 6452)	Frees the resolver specified by the Resolver (see page 6456) property.
V	InternalApplyUpdates (see page 6452)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.

	InternalGetRecords (see page 6453)	Provides the underlying implementation of the GetRecords (see page 6476) method.
	InternalRowRequest (see page 6454)	Provides the underlying implementation of the RowRequest (see page 6480) method.
	LocateRecord (see page 6454)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	UpdateRecord (see page 6455)	Retrieves specified data from a source dataset into a delta packet.

72.1.2.4.1 TBaseProvider.CheckResolver Method

Instantiates a resolver for the Resolver (see page 6456) property if it does not already exist.

Pascal

```
procedure CheckResolver;
```

C++

```
__fastcall CheckResolver();
```

Description

CheckResolver is called internally when the provider needs to use its resolver component. It checks whether Resolver (see page 6456) is nil (Delphi) or NULL (C++), and if so, calls CreateResolver (see page 6450) to assign a value.

See Also

Resolver (see page 6456)

CreateResolver (see page 6450)

72.1.2.4.2 TBaseProvider.CreateDataPacket Method

Generates a data packet for the data associated with the provider.

Pascal

```
procedure CreateDataPacket(PacketOpts: TGetRecordOptions; ProvOpts: TProviderOptions; var
RecsOut: Integer; var Data: OleVariant); virtual;
```

C++

```
virtual __fastcall CreateDataPacket(TGetRecordOptions PacketOpts, TProviderOptions ProvOpts,
int RecsOut, OleVariant Data);
```

Description

CreateDataPacket is used internally to generate data packets. As implemented in TBaseProvider (see page 6445), CreateDataPacket sets RecsOut to 0 and Data (see page 6482) to NULL. Descendants override this method to create the data packets that are returned by GetRecords (see page 6476).

The PacketOpts parameter indicates what information (other than record values) is included in the data packet. It reflects the Options (see page 6455) passed to the GetRecords (see page 6476) method.

The ProvOpts parameter reflects the current value of the Options (see page 6455) property.

The RecsOut parameter specifies the number of records desired in the packet and returns the actual number of records in the generated data packet.

The Data (see page 6482) parameter returns the generated data packet.

See Also

GetRecords (see page 6476)

Options (see page 6455)

72.1.2.4.3 TBaseProvider.CreateResolver Method

Creates a resolver component for handling the application of updates.

Pascal

```
function CreateResolver: TCustomResolver; virtual;
```

C++

```
virtual __fastcall TCustomResolver CreateResolver();
```

Description

CreateResolver is called automatically when a resolver is needed to implement one of TBaseProvider (see page 6445)'s methods. As implemented in TBaseProvider (see page 6445), CreateResolver always returns nil (Delphi) or NULL (C++). Descendants of TBaseProvider (see page 6445) override this method to create a TCustomResolver (see page 6491) descendant that applies updates in the desired manner.

See Also

Resolver (see page 6456)

FreeResolver (see page 6452)

72.1.2.4.4 TBaseProvider.DoAfterUpdateRecord Method

Generates an AfterUpdateRecord (see page 6458) event.

Pascal

```
procedure DoAfterUpdateRecord(SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
TUpdateKind); virtual;
```

C++

```
virtual __fastcall DoAfterUpdateRecord(TDataSet SourceDS, TCustomClientDataSet DeltaDS,
TUpdateKind UpdateKind);
```

Description

DoAfterUpdateRecord is the protected implementation of the AfterUpdateRecord (see page 6458) event. Override this method to perform class-specific actions in response to data requests.

See Also

AfterUpdateRecord (see page 6458)

72.1.2.4.5 TBaseProvider.DoBeforeCommit Method

Pascal

```
procedure DoBeforeCommit(Delta: TPacketDataSet; ErrorCount: Integer; MaxErrors: Integer; const
ReturnVar: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeCommit(TPacketDataSet Delta, int ErrorCount, int MaxErrors, const
OleVariant ReturnVar);
```

Description

This is DoBeforeCommit, a member of class TBaseProvider.

72.1.2.4.6 TBaseProvider.DoBeforeUpdateRecord Method

Pascal

```
procedure DoBeforeUpdateRecord(SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
TUpdateKind; var Applied: Boolean); virtual;
```

C++

```
virtual __fastcall DoBeforeUpdateRecord(TDataSet SourceDS, TCustomClientDataSet DeltaDS,
TUpdateKind UpdateKind, Boolean Applied);
```

Description

This is DoBeforeUpdateRecord, a member of class TBaseProvider.

72.1.2.4.7 TBaseProvider.DoOnGetData Method

Generates an OnGetData (see page 6459) event.

Pascal

```
procedure DoOnGetData(var Data: OleVariant); virtual;
```

C++

```
virtual __fastcall DoOnGetData(OleVariant Data);
```

Description

DoOnGetData is the protected implementation of the OnGetData (see page 6459) event. Override this method to perform class-specific actions in response to data requests.

See Also

OnGetData (see page 6459)

72.1.2.4.8 TBaseProvider.DoOnUpdateData Method

Generates an OnUpdateData (see page 6459) event.

Pascal

```
procedure DoOnUpdateData(Delta: TPacketDataSet); virtual;
```

C++

```
virtual __fastcall DoOnUpdateData(TPacketDataSet Delta);
```

Description

DoOnUpdateData is the protected implementation of the OnUpdateData (see page 6459) event. The Delta property contains the delta packet of records to update. It is passed to the OnUpdateData (see page 6459) event handler as a client dataset.

Override this method to perform class-specific actions in response to update requests.

See Also

OnUpdateData (see page 6459)

72.1.2.4.9 TBaseProvider.FetchDetails Method

Fills a delta dataset with all nested dataset field values.

Pascal

```
procedure FetchDetails(Source: TDataSet; Delta: TDataSet); virtual;
```

C++

```
virtual __fastcall FetchDetails(TDataSet Source, TDataSet Delta);
```

Description

FetchDetails is used internally to fetch the current values of any nested datasets when the provider is executing its RowRequest (see page 6480) method. As implemented in TBaseProvider (see page 6445), FetchDetails does nothing. Descendants override this method to add nested detail datasets to the delta packet specified by Delta.

Source is a source dataset that has the current values for all fields. Delta is a delta dataset that must be updated so that its nested detail fields match the values in Source.

See Also

RowRequest (see page 6480)

72.1.2.4.10 TBaseProvider.FreeResolver Method

Frees the resolver specified by the Resolver (see page 6456) property.

Pascal

```
procedure FreeResolver;
```

C++

```
__fastcall FreeResolver();
```

Description

Call FreeResolver to safely dispose of the component specified by the Resolver (see page 6456) property. FreeResolver sets the Resolver (see page 6456) property to nil (Delphi) or NULL (C++) after calling the resolver's Free method.

See Also

Resolver (see page 6456)

CreateResolver (see page 6450)

System::TObject::Free

72.1.2.4.11 TBaseProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 6468) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates (see page 6468) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 6487) event and before generating the AfterApplyUpdates (see page 6483) event.

If the Options (see page 6455) property includes poReadOnly, InternalApplyUpdates raises an exception. Otherwise, it calls the ApplyUpdates (see page 6468) method of its Resolver (see page 6456) component to handle the actual update and resolving process. If the Resolver (see page 6456) property is nil (Delphi) or NULL (C++), ApplyUpdates (see page 6468) calls the CreateResolver (see page 6450) method and assigns the resulting resolver to the Resolver (see page 6456) property.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

Resolver (see page 6456)

ApplyUpdates (see page 6494)

CreateResolver (see page 6450)

Options (see page 6455)

applyupdates (see page 6468)

72.1.2.4.12 TBaseProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 6476) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 6476) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 6473) to apply the CommandText and Params and generate a BeforeGetRecords (see page 6488) event. InternalGetRecords calls CreateDataPacket (see page 6449) to create a data packet that reflects the current property settings of the provider. It then generates an OnGetData (see page 6459) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options (see page 6455) indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options (see page 6455) property does not include poAllowCommandText.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure.

If the provider is associated with a table, Params specifies field values that filter the result set. On exit from GetRecords (see page 6476), Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

Data (see page 6482)

CreateDataPacket (see page 6449)

GetRecords (see page 6476)

OnGetData (see page 6459)

72.1.2.4.13 TBaseProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 6480) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; RequestType: TFetchOptions): OleVariant;
override;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions
RequestType);
```

Description

The RowRequest (see page 6480) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 6490) event and before generating an AfterRowRequest (see page 6486) event. InternalRowRequest passes this request on to its associated Resolver (see page 6456).

The Row parameter is an OleVariant that describes the record for which information is requested. The provider uses Row to locate the record before assembling the requested information.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 6480)

Resolver (see page 6456)

RowRequest (see page 6500)

72.1.2.4.14 TBaseProvider.LocateRecord Method

Positions a data set on the record that corresponds to the current update in a specified delta packet.

Pascal

```
procedure LocateRecord(Source: TDataSet; Delta: TDataSet); virtual;
```

C++

```
virtual __fastcall LocateRecord(TDataSet Source, TDataSet Delta);
```

Description

LocateRecord is called indirectly by the RowRequest (see page 6480) method to locate a record before fetching information about a specified record. The Source parameter is the dataset containing current record values. The Delta parameter is a delta packet that identifies the desired type of information and information identifying the desired record.

As implemented in TBaseProvider (see page 6445), LocateRecord does nothing. Descendants override this method to locate the specified record in Source and navigate to it.

See Also

RowRequest (see page 6480)

72.1.2.4.15 TBaseProvider.UpdateRecord Method

Retrieves specified data from a source dataset into a delta packet.

Pascal

```
procedure UpdateRecord(Source: TDataSet; Delta: TDataSet; BlobsOnly: Boolean; KeyOnly:
Boolean); virtual;
```

C++

```
virtual __fastcall UpdateRecord(TDataSet Source, TDataSet Delta, Boolean BlobsOnly, Boolean
KeyOnly);
```

Description

UpdateRecord is called indirectly by the RowRequest (see page 6480) method to fetch BLOB fields or nested detail sets. The Source parameter is the dataset containing the BLOB fields or the detail dataset. The Delta parameter is a delta packet that returns the requested data. BlobsOnly indicates whether all field values should be provided (BlobsOnly == false) or only Blob field values (BlobsOnly == true).

As implemented in TBaseProvider (see page 6445), UpdateRecord does nothing. Descendants override this method to fill the Delta parameter with the requested information.

See Also

RowRequest (see page 6480)

72.1.2.5 TBaseProvider Properties

The properties of the TBaseProvider class are listed here.

Protected Properties

	Name	Description
	PacketDataSet (see page 6456)	This is PacketDataSet, a member of class TBaseProvider.

Public Properties

	Name	Description
	Options (see page 6455)	Influences what is included in data packets and how the information in data packets is used.
R	Resolver (see page 6456)	Provides access to the resolver component that applies updates and resolves update errors.
	UpdateMode (see page 6457)	Controls how the provider locates records when applying updates.

72.1.2.5.1 TBaseProvider.Options Property

Influences what is included in data packets and how the information in data packets is used.

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

Set Options to customize how the provider communicates with client datasets. Options is a set that specifies

Whether nested detail sets and BLOB data are included in data packets or fetched separately.

Whether data packets include field component properties such as formats, display names, and maximum & minimum values.

Whether data packets are provided on a read-only basis, and whether specific types of changes (insertions, edits, or deletions) are disabled.

Whether changes to master table fields should cause corresponding updates to detail tables.

Whether a single update can affect multiple server records.

Whether the client's records are refreshed when it applies updates.

Whether the client can send SQL statements that override the provider's dataset.

Note: Not all options are supported by an InternetExpress client. Specifically, InternetExpress clients don't support poFetchBlobsOnDemand, poFetchDetailsOnDemand, poDisableInserts, poDisableEdits, psDisableDeletes, poPropogateChanges, and poAllowCommandText.

See Also

Data (see page 6482)

applyupdates (see page 6468)

Overriding the Dataset On the Application Server

Setting Options That Influence the Data Packets

72.1.2.5.2 TBaseProvider.PacketDataSet Property

Pascal

```
property PacketDataSet: TPacketDataSet;
```

C++

```
__property TPacketDataSet PacketDataSet;
```

Description

This is PacketDataSet, a member of class TBaseProvider.

72.1.2.5.3 TBaseProvider.Resolver Property

Provides access to the resolver component that applies updates and resolves update errors.

Pascal

```
property Resolver: TCustomResolver;
```

C++

```
__property TCustomResolver Resolver;
```

Description

Provider (see page 6441) components use a resolver to handle the details of applying updates and resolving update errors. Most applications do not need to use the resolver directly.

When any method that uses a resolver is called, the provider automatically creates a resolver component (using the CreateResolver (see page 6450) method) if the value of Resolver is nil (Delphi) or NULL (C++). Thus, this property is nil (Delphi) or NULL (C++) until the first method that requires a resolver is called.

See Also

applyupdates (see page 6468)

CreateResolver (see page 6450)

72.1.2.5.4 TBaseProvider.UpdateMode Property

Controls how the provider locates records when applying updates.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria for locating a record in the dataset when applying updates. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

Note: To control how the provider locates records by specifying individual fields, use the field components' ProviderFlags property.

See Also

applyupdates (see page 6468)

ProviderFlags

Influencing How Updates Are Applied

72.1.2.6 TBaseProvider Events

The events of the TBaseProvider class are listed here.

Public Events

	Name	Description
	AfterUpdateRecord (see page 6458)	Occurs when the associated resolver has just updated a record.
	BeforeUpdateRecord (see page 6458)	This is BeforeUpdateRecord, a member of class TBaseProvider.
	OnDataRequest (see page 6458)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 6459)	Occurs after a provider retrieves data but before it is sent to the client.
	OnUpdateData (see page 6459)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 6460)	Occurs when a provider can't update a record because of an error condition.

72.1.2.6.1 TBaseProvider.AfterUpdateRecord Event

Occurs when the associated resolver has just updated a record.

Pascal

```
property AfterUpdateRecord: TAfterUpdateRecordEvent;
```

C++

```
__property TAfterUpdateRecordEvent AfterUpdateRecord;
```

Description

Write an AfterUpdateRecord event handler to provide special processing immediately after a record has been successfully updated. For example, use AfterUpdateRecord to maintain a log of successful updates.

See Also

beforeupdaterecord (see page 6458)

AfterApplyUpdates (see page 6483)

72.1.2.6.2 TBaseProvider.BeforeUpdateRecord Event

Pascal

```
property BeforeUpdateRecord: TBeforeUpdateRecordEvent;
```

C++

```
__property TBeforeUpdateRecordEvent BeforeUpdateRecord;
```

Description

This is BeforeUpdateRecord, a member of class TBaseProvider.

72.1.2.6.3 TBaseProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 6468)

IAppServer (see page 5222)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

72.1.2.6.4 TBaseProvider.OnGetData Event

Occurs after a provider retrieves data but before it is sent to the client.

Pascal

```
property OnGetData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnGetData;
```

Description

Write an OnGetData event handler to perform any custom data processing the provider should handle before returning data to a client. The OnGetData event occurs after a provider fetches data from its dataset but before the data is sent on to the client. Using the methods of the DataSet parameter, data can be edited in the OnGetData event handler before sending it to the client dataset or XML broker.

Note: OnGetData occurs when the Data (see page 6482) property is read, as well as when the GetRecords (see page 6476) method is called.

See Also

GetRecords (see page 6476)

OnUpdateData (see page 6459)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

Responding to Client Data Requests

72.1.2.6.5 TBaseProvider.OnUpdateData Event

Occurs when a provider begins to apply updates.

Pascal

```
property OnUpdateData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnUpdateData;
```

Description

Write an OnUpdateData event to handle special processing before the provider begins applying updates received from the client dataset or XML broker. Using the methods belonging to the DataSet parameter, an application can

Examine data (for example, for values or data changes that should not permitted), and raise exceptions that cancel applying of updates before they occur.

Change data (for example encrypting or decrypting values) before it is sent on to the source dataset or database server.

See Also

applyupdates (see page 6468)

beforeupdaterecord (see page 6458)

Adding Custom Information to Data Packets

Editing Delta Packets Before Updating the Database

Influencing How Updates Are Applied

72.1.2.6.6 TBaseProvider.OnUpdateError Event

Occurs when a provider can't update a record because of an error condition.

Pascal

```
property OnUpdateError: TResolverErrorEvent;
```

C++

```
__property TResolverErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to error conditions that arise when the provider applies a client dataset's updates. The OnUpdateError event handler is called for each record that can't be updated.

Warning: The provider can't detect when an update conflicts with another user's changes to a memo field. This situation will not generate an OnUpdateError event.

See Also

applyupdates (see page 6468)

OnReconcileError

NewValue

OldValue

CurValue

Resolving Update Errors On the Provider

72.1.3 TCustomPacketWriter Class

Class Hierarchy

Pascal

```
TCustomPacketWriter = class(TObject);
```

C++

```
class TCustomPacketWriter : public TObject;
```

File

Provider

Description

This is class Provider.TCustomPacketWriter.

72.1.3.1 TCustomPacketWriter Members

The following tables list the members exposed by TCustomPacketWriter.

Protected Fields

	Name	Description
	FBuffer (see page 6462)	This is FBuffer, a member of class TCustomPacketWriter.

Protected Methods

	Name	Description
V	AddAttribute (see page 6463)	This is AddAttribute, a member of class TCustomPacketWriter.
	Check (see page 6463)	This is Check, a member of class TCustomPacketWriter.
	SetDSWriter (see page 6463)	This is SetDSWriter, a member of class TCustomPacketWriter.

Protected Properties

	Name	Description
R	DSWriter (see page 6464)	This is DSWriter, a member of class TCustomPacketWriter.

Public Methods

	Name	Description
V	Create (see page 6461)	Constructs an object and initializes its data before the object is first used.
V	Destroy (see page 6461)	Disposes of an object instance.

72.1.3.2 TCustomPacketWriter.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TCustomPacketWriter();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy

System::TObject::Free

72.1.3.3 TCustomPacketWriter.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomPacketWriter();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

72.1.3.4 TCustomPacketWriter Fields

The fields of the TCustomPacketWriter class are listed here.

Protected Fields

	Name	Description
	FBuffer (see page 6462)	This is FBuffer, a member of class TCustomPacketWriter.

72.1.3.4.1 TCustomPacketWriter.FBuffer Field

Pascal

```
FBuffer: TBlobByteData;
```

C++

```
TBlobByteData FBuffer;
```

Description

This is FBuffer, a member of class TCustomPacketWriter.

72.1.3.5 TCustomPacketWriter Methods

The methods of the TCustomPacketWriter class are listed here.

Protected Methods

	Name	Description
	AddAttribute (see page 6463)	This is AddAttribute, a member of class TCustomPacketWriter.
	Check (see page 6463)	This is Check, a member of class TCustomPacketWriter.
	SetDSWriter (see page 6463)	This is SetDSWriter, a member of class TCustomPacketWriter.

72.1.3.5.1 TCustomPacketWriter.AddAttribute Method

Pascal

```
procedure AddAttribute(Area: TPcktAttrArea; const ParamName: string; const Value: OleVariant;
IncludeInDelta: Boolean); virtual;
```

C++

```
virtual __fastcall AddAttribute(TPcktAttrArea Area, const UnicodeString ParamName, const
OleVariant Value, Boolean IncludeInDelta);
```

Description

This is AddAttribute, a member of class TCustomPacketWriter.

72.1.3.5.2 TCustomPacketWriter.Check Method

Pascal

```
procedure Check(Status: Integer);
```

C++

```
__fastcall Check(int Status);
```

Description

This is Check, a member of class TCustomPacketWriter.

72.1.3.5.3 TCustomPacketWriter.SetDSWriter Method

Pascal

```
procedure SetDSWriter(const Value: IDSWriter);
```

C++

```
__fastcall SetDSWriter(const IDSWriter Value);
```

Description

This is SetDSWriter, a member of class TCustomPacketWriter.

72.1.3.6 TCustomPacketWriter Properties

The properties of the TCustomPacketWriter class are listed here.

Protected Properties

	Name	Description
	DSWriter (see page 6464)	This is DSWriter, a member of class TCustomPacketWriter.

72.1.3.6.1 TCustomPacketWriter.DSWriter Property

Pascal

```
property DSWriter: IDSWriter;
```

C++

```
__property IDSWriter DSWriter;
```

Description

This is DSWriter, a member of class TCustomPacketWriter.

72.1.4 TCustomProvider Class

TCustomProvider is the base class for data providers.

Class Hierarchy

Pascal

```
TCustomProvider = class(TComponent);
```

C++

```
class TCustomProvider : public TComponent;
```

File

Provider

Description

TCustomProvider defines the fundamental properties, methods, and events required of all provider objects, either through abstract (in C++ terminology, pure virutal) method declarations, or through virtual methods that do nothing but provide a backstop.

Do not create instances of TCustomProvider. To add a provider component to a form or data module, use a descendant of TCustomProvider such as TDataSetProvider (see page 6510) or TXMLTransformProvider.

A provider derived from TCustomProvider is an intermediate link between TClientDataSet or TXMLBroker object and a source dataset or source XML document. The source dataset or XML document can reside in the same application as the client dataset or XML broker, or it can be part of a separate application server. The provider packages data from the source dataset or XML document and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database, to the source dataset, or to the source XML document.

If the provider can encode data packets in XML, it can supply data to an XML broker, which adds the data packet to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database, to the source dataset, or to the source XML document.

Client datasets and XML brokers communicate with a provider through the IAppServer (see page 5222) interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 6482) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet

TDataSetProvider (see page 6510)

IAppServer (see page 5222)

TXMLTransformProvider

72.1.4.1 TCustomProvider Members

The following tables list the members exposed by TCustomProvider.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 6483)	Occurs after the provider applies updates.
	AfterExecute (see page 6484)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 6485)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 6485)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 6486)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 6487)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 6487)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 6488)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 6488)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 6490)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 6490)	Occurs when a provider calls its DataRequest (see page 6468) method.
	OnValidate (see page 6491)	This is OnValidate, a member of class TCustomProvider.

Protected Methods

	Name	Description
	DoAfterApplyUpdates (see page 6469)	Generates an AfterApplyUpdates (see page 6483) event.
	DoAfterExecute (see page 6469)	Generates an AfterExecute (see page 6484) event.
	DoAfterGetParams (see page 6470)	Generates an AfterGetParams (see page 6485) event.
	DoAfterGetRecords (see page 6470)	Generates an AfterGetRecords (see page 6485) event.
	DoAfterRowRequest (see page 6471)	Generates an AfterRowRequest (see page 6486) event.
	DoApplyUpdates (see page 6471)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 6471)	Generates a BeforeApplyUpdates (see page 6487) event.
	DoBeforeExecute (see page 6472)	Generates a BeforeExecute (see page 6487) event.
	DoBeforeGetParams (see page 6472)	Generates a BeforeGetParams (see page 6488) event.
	DoBeforeGetRecords (see page 6473)	Generates a BeforeGetRecords (see page 6488) event.
	DoBeforeRowRequest (see page 6474)	Generates a BeforeRowRequest (see page 6490) event.
	DoGetRecords (see page 6474)	This is DoGetRecords, a member of class TCustomProvider.
	InternalApplyUpdates (see page 6477)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.
	InternalExecute (see page 6478)	Provides the underlying implementation of the Execute (see page 6474) method.
	InternalGetParams (see page 6478)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 6479)	Provides the underlying implementation of the GetRecords (see page 6476) method.

	InternalRowRequest (see page 6480)	Provides the underlying implementation of the RowRequest (see page 6480) method.
	SetActiveUpdateException (see page 6481)	This is SetActiveUpdateException, a member of class TCustomProvider.

Protected Properties

	Name	Description
	ActiveUpdateException (see page 6482)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 6483)	This is OwnerData, a member of class TCustomProvider.

Public Methods

	Name	Description
	ApplyUpdates (see page 6468)	This is the overview for the ApplyUpdates method overload.
	Create (see page 6466)	Creates a new TCustomProvider (see page 6464) object.
	DataRequest (see page 6468)	Generates an OnDataRequest (see page 6490) event.
	Destroy (see page 6467)	Frees an instance of a provider object.
	Execute (see page 6474)	Executes a query, stored procedure, or custom SQL statement.
	GetParams (see page 6475)	Returns the current parameter values.
	GetRecords (see page 6476)	This is the overview for the GetRecords method overload.
	RowRequest (see page 6480)	Returns information from a specified record.

Public Properties

	Name	Description
	Data (see page 6482)	Denotes a data packet created by the provider.
	Exported (see page 6482)	Allows client applications to pass calls to the provider when it resides on an application server.

72.1.4.2 TCustomProvider.Create Constructor

Creates a new TCustomProvider (see page 6464) object.

Pascal

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

C++

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

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

The constructor assigns the AOwner parameter as the owner of the provider. This owner is responsible for freeing the provider when it is destroyed. If the owner is a remote data module, the constructor registers the provider with that remote data module so that it is available to client applications through the IAppServer (see page 5222) interface.

See Also

IAppServer (see page 5222)

Providers

registerprovider

Owner

72.1.4.3 TCustomProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the provider is not nil before calling its Destroy method. If the provider's Owner is a remote data module, Destroy unregisters the provider from that remote data module.

72.1.4.4 TCustomProvider Methods

The methods of the TCustomProvider class are listed here.

Protected Methods

	Name	Description
	DoAfterApplyUpdates (see page 6469)	Generates an AfterApplyUpdates (see page 6483) event.
	DoAfterExecute (see page 6469)	Generates an AfterExecute (see page 6484) event.
	DoAfterGetParams (see page 6470)	Generates an AfterGetParams (see page 6485) event.
	DoAfterGetRecords (see page 6470)	Generates an AfterGetRecords (see page 6485) event.
	DoAfterRowRequest (see page 6471)	Generates an AfterRowRequest (see page 6486) event.
	DoApplyUpdates (see page 6471)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 6471)	Generates a BeforeApplyUpdates (see page 6487) event.
	DoBeforeExecute (see page 6472)	Generates a BeforeExecute (see page 6487) event.
	DoBeforeGetParams (see page 6472)	Generates a BeforeGetParams (see page 6488) event.
	DoBeforeGetRecords (see page 6473)	Generates a BeforeGetRecords (see page 6488) event.
	DoBeforeRowRequest (see page 6474)	Generates a BeforeRowRequest (see page 6490) event.
	DoGetRecords (see page 6474)	This is DoGetRecords, a member of class TCustomProvider.
	InternalApplyUpdates (see page 6477)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.
	InternalExecute (see page 6478)	Provides the underlying implementation of the Execute (see page 6474) method.
	InternalGetParams (see page 6478)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 6479)	Provides the underlying implementation of the GetRecords (see page 6476) method.
	InternalRowRequest (see page 6480)	Provides the underlying implementation of the RowRequest (see page 6480) method.
	SetActiveUpdateException (see page 6481)	This is SetActiveUpdateException, a member of class TCustomProvider.

Public Methods

	Name	Description
	ApplyUpdates (see page 6468)	This is the overview for the ApplyUpdates method overload.
	DataRequest (see page 6468)	Generates an OnDataRequest (see page 6490) event.
	Execute (see page 6474)	Executes a query, stored procedure, or custom SQL statement.
	GetParams (see page 6475)	Returns the current parameter values.
	GetRecords (see page 6476)	This is the overview for the GetRecords method overload.
	RowRequest (see page 6480)	Returns information from a specified record.

72.1.4.4.1 ApplyUpdates Method

This is the overview for the ApplyUpdates method overload.

Overload List

	Name	Description
	TCustomProvider.ApplyUpdates (OleVariant, Integer, Integer) (see page 6468)	This is ApplyUpdates, a member of class TCustomProvider.
	TCustomProvider.ApplyUpdates (OleVariant, Integer, Integer, OleVariant) (see page 6468)	This is ApplyUpdates, a member of class TCustomProvider.

72.1.4.4.1.1 TCustomProvider.ApplyUpdates Method (OleVariant, Integer, Integer)

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant; overload;
```

C++

```
__fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is ApplyUpdates, a member of class TCustomProvider.

72.1.4.4.1.2 TCustomProvider.ApplyUpdates Method (OleVariant, Integer, Integer, OleVariant)

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer;
var OwnerData: OleVariant): OleVariant; overload;
```

C++

```
__fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int ErrorCount,
OleVariant OwnerData);
```

Description

This is ApplyUpdates, a member of class TCustomProvider.

72.1.4.4.2 TCustomProvider.DataRequest Method

Generates an OnDataRequest (see page 6490) event.

Pascal

```
function DataRequest(Input: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DataRequest(OleVariant Input);
```

Description

Write an OnDataRequest (see page 6490) event handler and call the DataRequest function to invoke the event handler at runtime. This function and the event it invokes enable an application to perform custom processing in response to a request for data.

Note: Calling the client dataset's DataRequest method causes a call to the provider's DataRequest method.

See Also

OnDataRequest (see page 6490)

AS_DataRequest (see page 5224)

DataRequest

Communicating with the Client Dataset

72.1.4.4.3 TCustomProvider.DoAfterApplyUpdates Method

Generates an AfterApplyUpdates (see page 6483) event.

Pascal

```
procedure DoAfterApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterApplyUpdates(OleVariant OwnerData);
```

Description

The ApplyUpdates (see page 6468) method calls DoAfterApplyUpdates to generate the AfterApplyUpdates (see page 6483) event after applying updates and resolving any update errors that are handled by the provider.

OwnerData (see page 6483) supplies custom information to the AfterApplyUpdates (see page 6483) event handler. This is the same as the outgoing OwnerData (see page 6483) parameter of the BeforeApplyUpdates (see page 6487) event handler. The AfterApplyUpdates (see page 6483) event handler can change this value. The returned value is then passed to the client if it supplied an OwnerData (see page 6483) parameter when calling ApplyUpdates (see page 6468).

See Also

applyupdates (see page 6468)

AfterApplyUpdates (see page 6483)

BeforeApplyUpdates (see page 6487)

72.1.4.4.4 TCustomProvider.DoAfterExecute Method

Generates an AfterExecute (see page 6484) event.

Pascal

```
procedure DoAfterExecute(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterExecute(OleVariant OwnerData);
```

Description

The Execute (see page 6474) method calls DoAfterExecute to generate the AfterExecute (see page 6484) event after executing CommandText or the associated query or stored procedure but before fetching any output parameters.

OwnerData (see page 6483) supplies custom information to the AfterExecute (see page 6484) event handler. This is the same as the outgoing OwnerData (see page 6483) parameter of the BeforeExecute (see page 6487) event handler. The AfterExecute (see page 6484) event handler can change this value. The returned value is then passed to the client.

See Also

Execute (see page 6474)

AfterExecute (see page 6484)

beforeexecute (see page 6487)

72.1.4.4.5 TCustomProvider.DoAfterGetParams Method

Generates an AfterGetParams (see page 6485) event.

Pascal

```
procedure DoAfterGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetParams(OleVariant OwnerData);
```

Description

The GetParams (see page 6475) method calls DoAfterGetParams to generate the AfterGetParams (see page 6485) event after fetching the current parameter values of the associated dataset.

OwnerData (see page 6483) supplies custom information to the AfterGetParams (see page 6485) event handler. This is the same as the outgoing OwnerData (see page 6483) parameter of the BeforeGetParams (see page 6488) event handler. The AfterGetParams (see page 6485) event handler can change this value. The returned value is then passed to the client.

See Also

GetParams (see page 6475)

aftergetparams (see page 6485)

BeforeGetParams (see page 6488)

72.1.4.4.6 TCustomProvider.DoAfterGetRecords Method

Generates an AfterGetRecords (see page 6485) event.

Pascal

```
procedure DoAfterGetRecords(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetRecords(OleVariant OwnerData);
```

Description

The GetRecords (see page 6476) method calls DoAfterGetRecords to generate the AfterGetRecords (see page 6485) event after assembling a data packet but before updating the parameters to reflect any output parameters.

OwnerData (see page 6483) supplies custom information to the AfterGetRecords (see page 6485) event handler. This is the same as the outgoing OwnerData (see page 6483) parameter of the BeforeGetRecords (see page 6488) event handler. The AfterGetRecords (see page 6485) event handler can change this value. The returned value is then passed to the client.

See Also

GetRecords (see page 6476)

AfterGetRecords (see page 6485)

BeforeGetRecords (see page 6488)

72.1.4.4.7 TCustomProvider.DoAfterRowRequest Method

Generates an AfterRowRequest (see page 6486) event.

Pascal

```
procedure DoAfterRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterRowRequest(OleVariant OwnerData);
```

Description

The RowRequest (see page 6480) method calls DoAfterRowRequest to generate the AfterRowRequest (see page 6486) event after fetching the requested data.

OwnerData (see page 6483) supplies custom information to the AfterRowRequest (see page 6486) event handler. This is the same as the outgoing OwnerData (see page 6483) parameter of the BeforeRowRequest (see page 6490) event handler. The AfterRowRequest (see page 6486) event handler can change this value. The returned value is then passed to the client.

See Also

RowRequest (see page 6480)

AfterRowRequest (see page 6486)

BeforeRowRequest (see page 6490)

72.1.4.4.8 TCustomProvider.DoApplyUpdates Method

Pascal

```
function DoApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer;
var OwnerData: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DoApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount, OleVariant OwnerData);
```

Description

This is DoApplyUpdates, a member of class TCustomProvider.

72.1.4.4.9 TCustomProvider.DoBeforeApplyUpdates Method

Generates a BeforeApplyUpdates (see page 6487) event.

Pascal

```
procedure DoBeforeApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeApplyUpdates(OleVariant OwnerData);
```

Description

The ApplyUpdates (see page 6468) method calls DoBeforeApplyUpdates to generate the BeforeApplyUpdates (see page 6487) event before it applies the updates in Delta.

Delta is the delta packet containing the updates as received from the client.

MaxErrors is the number of errors tolerated before the update operation should aborted.

OwnerData (see page 6483) supplies custom information to the BeforeApplyUpdates (see page 6487) event handler. The BeforeApplyUpdates (see page 6487) event handler can change this value. The returned value is then passed to the AfterApplyUpdates (see page 6483) event handler after applying the updates in Delta.

As implemented in TCustomProvider (see page 6464), Delta and MaxErrors are ignored.

See Also

applyupdates (see page 6468)

AfterApplyUpdates (see page 6483)

BeforeApplyUpdates (see page 6487)

72.1.4.4.10 TCustomProvider.DoBeforeExecute Method

Generates a BeforeExecute (see page 6487) event.

Pascal

```
procedure DoBeforeExecute(const CommandText: WideString; var Params: OleVariant; var
OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeExecute(const BSTR CommandText, OleVariant Params, OleVariant
OwnerData);
```

Description

The Execute (see page 6474) method calls DoBeforeExecute to generate the BeforeExecute (see page 6487) event before it executes CommandText or executes an associated query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params contains any parameter values that should be applied to CommandText or to the query or stored procedure before execution.

OwnerData (see page 6483) supplies custom information to the BeforeExecute (see page 6487) event handler. The event handler can change this value. The returned value is then passed to an AfterExecute (see page 6484) event handler and then returned to the caller of the Execute (see page 6474) method.

As implemented in TCustomProvider (see page 6464), CommandText and Params are ignored. Descendant classes apply these values before generating the BeforeExecute (see page 6487) event.

See Also

Execute (see page 6474)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

Options (see page 6455)

72.1.4.4.11 TCustomProvider.DoBeforeGetParams Method

Generates a BeforeGetParams (see page 6488) event.

Pascal

```
procedure DoBeforeGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetParams(OleVariant OwnerData);
```

Description

The GetParams (see page 6475) method calls DoBeforeGetParams to generate the BeforeGetParams (see page 6488) event before fetching the current parameter values of the associated dataset.

OwnerData (see page 6483) supplies custom information to the BeforeGetParams (see page 6488) event handler. The event handler can change this value. The returned value is then passed to an AfterGetParams (see page 6485) event handler and then returned to the caller of the GetParams (see page 6475) method.

See Also

GetParams (see page 6475)

BeforeGetParams (see page 6488)

aftergetparams (see page 6485)

72.1.4.4.12 TCustomProvider.DoBeforeGetRecords Method

Generates a BeforeGetRecords (see page 6488) event.

Pascal

```
procedure DoBeforeGetRecords(Count: Integer; Options: Integer; const CommandText: WideString;
var Params: OleVariant; var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetRecords(int Count, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

The GetRecords (see page 6476) method calls DoBeforeGetRecords to generate the BeforeGetRecords (see page 6488) event before it fetches data and assembles it into a data packet.

Count is the number of records requested. If Count is –1, all records are requested.

Options indicates whether the data packet should include metadata, whether it should start with the first record, and whether the data packet should be encoded in XML. For details, see the GetRecords (see page 6476) method.

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params contains any parameter values that should be applied to CommandText or to a query or stored procedure before fetching data into a data packet.

OwnerData (see page 6483) supplies custom information to the BeforeGetRecords (see page 6488) event handler. The event handler can change this value. The returned value is then passed to an AfterGetRecords (see page 6485) event handler and then (optionally) returned to the caller of the GetRecords (see page 6476) method.

As implemented in TCustomProvider (see page 6464), Count, Options, CommandText, and Params are ignored. Descendant classes can apply these values before generating the BeforeExecute (see page 6487) event.

See Also

GetRecords (see page 6476)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

Options (see page 6455)

72.1.4.4.13 TCustomProvider.DoBeforeRowRequest Method

Generates a BeforeRowRequest (see page 6490) event.

Pascal

```
procedure DoBeforeRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeRowRequest(OleVariant OwnerData);
```

Description

The RowRequest (see page 6480) method calls DoBeforeRowRequest to generate the BeforeRowRequest (see page 6490) event before fetching the requested data.

OwnerData (see page 6483) supplies custom information to the BeforeRowRequest (see page 6490) event handler. The event handler can change this value. The returned value is then passed to an AfterRowRequest (see page 6486) event handler and then returned to the caller of the RowRequest (see page 6480) method.

See Also

RowRequest (see page 6480)

BeforeRowRequest (see page 6490)

AfterRowRequest (see page 6486)

72.1.4.4.14 TCustomProvider.DoGetRecords Method

Pascal

```
function DoGetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; var Params: OleVariant; var OwnerData: OleVariant): OleVariant;
virtual;
```

C++

```
virtual __fastcall OleVariant DoGetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is DoGetRecords, a member of class TCustomProvider.

72.1.4.4.15 TCustomProvider.Execute Method

Executes a query, stored procedure, or custom SQL statement.

Pascal

```
procedure Execute(const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant); virtual;
```

C++

```
virtual __fastcall Execute(const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Client datasets call Execute through the AS_Execute method of the IAppServer (see page 5222) interface. In response, the

provider executes an SQL command. If the provider represents a query or stored procedure that does not return a cursor, calling Execute causes the provider to execute the query or stored procedure. If the caller supplies a value for CommandText, Execute may execute that statement instead (see below).

Before executing the appropriate command, Execute generates a BeforeExecute (see page 6487) event. After executing the command, it generates an AfterExecute (see page 6484) event. Finally, it returns any output parameters as the value of the Params parameter.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

OwnerData (see page 6483) contains custom information passed to the provider from the client dataset's BeforeExecute (see page 6487) event handler. This information is passed in to the provider's BeforeExecute (see page 6487) event handler and is returned by the provider's AfterExecute (see page 6484) event handler.

See Also

AS_Execute (see page 5224)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

BeforeExecute

AfterExecute

Execute

CommandText

InternalExecute (see page 6478)

Options (see page 6455)

Communicating with the Client Dataset

72.1.4.4.16 TCustomProvider.GetParams Method

Returns the current parameter values.

Pascal

```
function GetParams(var OwnerData: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetParams(OleVariant OwnerData);
```

Description

GetParams returns a variant array of variant arrays that contains the current parameter values for the provider's data source. The first array dimension is an index into all the parameter values. Its value runs from 0 to one less than the number of parameters. The second array dimension has four values: The parameter name (a Delphi string or a C++ AnsiString), the parameter data type (a TFieldType), the parameter type (a TParamType), and the parameter value (a Variant). The values in this array can be applied to a TParams object using the global UnpackParams procedure.

Before fetching the current parameter values, GetParams generates a BeforeGetParams (see page 6488) event, passing the value supplied as the OwnerData (see page 6483) parameter to the event handler. After creating the OleVariant that contains

the parameter values, GetParams generates an AfterGetParams (see page 6485) event, passing in the current value of OwnerData (see page 6483). Finally, the value of OwnerData (see page 6483) returned by the AfterGetParams (see page 6485) is returned in the OwnerData (see page 6483) parameter.

See Also

FetchParams

ParamType

Name

DataType

Value

AS_GetParams (see page 5225)

UnpackParams

BeforeGetParams (see page 6488)

aftergetparams (see page 6485)

InternalGetParams (see page 6478)

Communicating with the Client Dataset

72.1.4.4.17 GetRecords Method

This is the overview for the GetRecords method overload.

Overload List

	Name	Description
	TCustomProvider.GetRecords (Integer, Integer, Integer) (see page 6476)	Retrieves a data packet that represents the provider's associated data.
	TCustomProvider.GetRecords (Integer, Integer, Integer, WideString, OleVariant, OleVariant) (see page 6476)	Retrieves a data packet that represents the provider's associated data.

72.1.4.4.17.1 TCustomProvider.GetRecords Method (Integer, Integer, Integer)

Retrieves a data packet that represents the provider's associated data.

Pascal

```
function GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer): OleVariant;
overload;
function GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const CommandText:
WideString; var Params: OleVariant; var OwnerData: OleVariant): OleVariant; overload;
```

C++

```
__fastcall OleVariant GetRecords(int Count, int RecsOut, int Options);
__fastcall OleVariant GetRecords(int Count, int RecsOut, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

Call GetRecords to retrieve a data packet containing a specified number of records. GetRecords generates a BeforeGetRecords (see page 6488) event, assembles a data packet, and then generates an AfterGetRecords (see page 6485) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records in the data packet.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable, Params specifies field values that filter the result set. On exit from GetRecords, Params returns any output parameters.

OwnerData (see page 6483) contains custom information that is supplied by a client dataset's BeforeGetRecords (see page 6488) event handler. This information is passed to the provider's BeforeGetRecords (see page 6488) event handler and is returned by the provider's AfterGetRecords (see page 6485) event handler.

Records are returned as a data packet in a Variant.

Note: GetRecords always generates the BeforeGetRecords (see page 6488) and AfterGetRecords (see page 6485) events, even if you use the syntax that does not include an OwnerData (see page 6483) parameter. In that case, the value of OwnerData (see page 6483) is nil (Delphi) or NULL (C++) on entry to the BeforeGetRecords (see page 6488) event handler and is discarded on exit from the AfterGetRecords (see page 6485) event handler.

Tip: If the provider is part of a stateless application server, there is no guarantee that the provider will be positioned on the desired record before it creates a data packet. Use a BeforeGetRecords (see page 6488) event handler to position the cursor before the call to GetRecords.

See Also

DataRequest (see page 6468)

Data (see page 6482)

BeforeGetRecords (see page 6488)

AS_GetRecords (see page 5227)

AfterGetRecords (see page 6485)

InternalGetRecords (see page 6479)

BeforeGetRecords

Communicating with the Client Dataset

Responding to Client Data Requests

72.1.4.4.18 TCustomProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 6468) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
```

```
Integer): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount) = 0;
```

Description

The ApplyUpdates (see page 6468) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 6487) event and before generating the AfterApplyUpdates (see page 6483) event. The InternalApplyUpdates method of TCustomProvider (see page 6464) is abstract (Delphi) or pure virtual (C++), meaning that it has no implementation. Descendant classes override this method to apply updates, resolve update errors, and return any updates that could not be applied.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

applyupdates (see page 6468)

72.1.4.4.19 TCustomProvider.InternalExecute Method

Provides the underlying implementation of the Execute (see page 6474) method.

Pascal

```
procedure InternalExecute(const CommandText: WideString; var Params: OleVariant); virtual;
```

C++

```
virtual __fastcall InternalExecute(const BSTR CommandText, OleVariant Params);
```

Description

The Execute (see page 6474) method calls InternalExecute to execute the appropriate SQL statement after calling DoBeforeExecute (see page 6472) to generate a BeforeExecute (see page 6487) event and apply the CommandText and Params. As Implemented in TCustomProvider (see page 6464), InternalExecute does nothing. Descendants override this method to pass the execute command on to an associated dataset.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

See Also

Execute (see page 6474)

DoBeforeExecute (see page 6472)

72.1.4.4.20 TCustomProvider.InternalGetParams Method

Returns the requested parameter values as a Variant array.

Pascal

```
function InternalGetParams(Types: TParamTypes = AllParamTypes): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant InternalGetParams(TParamTypes Types = AllParamTypes);
```

Description

TCustomProvider (see page 6464) uses InternalGetParams internally whenever it needs to assemble the current parameter values into an OleVariant. This provides the underlying implementation of the GetParams (see page 6475) method, as well as assembling parameter values for methods that return output parameters (such as Execute (see page 6474) or GetRecords (see page 6476)).

The Types parameter limits the parameter values that are returned to a subset of all possible parameter types. If this parameter is omitted, all parameter values are returned.

As implemented in TCustomProvider (see page 6464), InternalGetParams always returns NULL. Descendants override this method to return the requested parameter values.

See Also

GetParams (see page 6475)

Execute (see page 6474)

GetRecords (see page 6476)

72.1.4.4.21 TCustomProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 6476) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 6476) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 6473) to apply the CommandText and Params and generate a BeforeGetRecords (see page 6488) event. As implemented in TCustomProvider (see page 6464), InternalGetRecords always returns NULL. Descendant classes override this method to fetch the requested records and assemble them into a data packet.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure.

If the provider is associated with a TTable object, Params specifies field values that filter the result set. On exit from GetRecords (see page 6476), Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

GetRecords (see page 6476)

DoBeforeGetRecords (see page 6473)

72.1.4.4.22 TCustomProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 6480) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; RequestType: TFetchOptions): OleVariant;
virtual;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions
RequestType);
```

Description

The RowRequest (see page 6480) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 6490) event and before generating an AfterRowRequest (see page 6486) event. As implemented in TCustomProvider (see page 6464), InternalRowRequest does nothing. Descendant classes override this method to assemble the requested information into an OleVariant.

The Row parameter is an OleVariant that describes the current record on the client dataset. It enables the provider to locate the record of interest.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 6480)

72.1.4.4.23 TCustomProvider.RowRequest Method

Returns information from a specified record.

Pascal

```
function RowRequest(const Row: OleVariant; RequestType: Integer; var OwnerData: OleVariant):
OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant RowRequest(const OleVariant Row, int RequestType, OleVariant
OwnerData);
```

Description

RowRequest is called automatically in response to the client dataset's FetchBlobs, FetchDetails, or RefreshRecord method. It generates a BeforeRowRequest (see page 6490) event, assembles the requested information, and generates an AfterRowRequest (see page 6486) event.

The Row parameter is an OleVariant that describes the record about which information is requested. It enables the provider to locate the record of interest.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (Its an

integer because the value must be marshaled over an IAppServer (see page 5222) interface). In Delphi, to convert the TFetchOptions value to the appropriate integer, use the Ord function. In C++, to combine TFetchOption values, use the bitwise "or" operator. For example:

```
Ord(foBlobs) + Ord(foDetails);
foBlobs || foDetails
```

RowRequest returns the requested information in a delta packet.

As implemented in TCustomProvider (see page 6464), RowRequest always returns NULL. Descendant classes override this method to return the requested data.

See Also

FetchBlobs

FetchDetails

RefreshRecord

BeforeRowRequest (see page 6490)

AfterRowRequest (see page 6486)

AS_RowRequest (see page 5228)

InternalRowRequest (see page 6480)

Communicating with the Client Dataset

72.1.4.4.24 TCustomProvider.SetActiveUpdateException Method

Pascal

```
procedure SetActiveUpdateException(E: Exception);
```

C++

```
__fastcall SetActiveUpdateException(Exception E);
```

Description

This is SetActiveUpdateException, a member of class TCustomProvider.

72.1.4.5 TCustomProvider Properties

The properties of the TCustomProvider class are listed here.

Protected Properties

	Name	Description
	ActiveUpdateException (see page 6482)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 6483)	This is OwnerData, a member of class TCustomProvider.

Public Properties

	Name	Description
	Data (see page 6482)	Denotes a data packet created by the provider.
	Exported (see page 6482)	Allows client applications to pass calls to the provider when it resides on an application server.

72.1.4.5.1 TCustomProvider.ActiveUpdateException Property

Pascal

```
property ActiveUpdateException: Exception;
```

C++

```
__property Exception ActiveUpdateException;
```

Description

This is ActiveUpdateException, a member of class TCustomProvider.

72.1.4.5.2 TCustomProvider.Data Property

Denotes a data packet created by the provider.

Pascal

```
property Data: OleVariant;
```

C++

```
__property OleVariant Data;
```

Description

Data is a read-only property that provides access to a data packet containing all records. Reading Data calls GetRecords (see page 6476) with a parameter of -1.

As implemented in TCustomProvider (see page 6464), GetRecords (see page 6476) returns NULL. Descendants of TCustomProvider (see page 6464) reimplement the GetRecords (see page 6476) method to implement the Data property.

See Also

GetRecords (see page 6476)

72.1.4.5.3 TCustomProvider.Exported Property

Allows client applications to pass calls to the provider when it resides on an application server.

Pascal

```
property Exported: Boolean;
```

C++

```
__property Boolean Exported;
```

Description

Use Exported to indicate whether the provider is made available to clients of a remote data module that owns it. When Exported is true, client applications can specify the provider as a target for calls made using the IAppServer (see page 5222) interface. When Exported is false, IAppServer (see page 5222) calls that specify the provider cause an exception.

Note: In addition to the Exported property, the provider must be registered with the remote data module for clients to access it using the data module's IAppServer (see page 5222) interface. When the provider's Owner is the remote data module, it is automatically registered. If the provider has a different owner, the server application must explicitly register providers using the remote data module's RegisterProvider method.

See Also

IAppServer (see page 5222)

Providers

registerprovider

72.1.4.5.4 TCustomProvider.OwnerData Property

Pascal

```
property OwnerData: OleVariant;
```

C++

```
__property OleVariant OwnerData;
```

Description

This is OwnerData, a member of class TCustomProvider.

72.1.4.6 TCustomProvider Events

The events of the TCustomProvider class are listed here.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 6483)	Occurs after the provider applies updates.
	AfterExecute (see page 6484)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 6485)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 6485)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 6486)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 6487)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 6487)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 6488)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 6488)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 6490)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 6490)	Occurs when a provider calls its DataRequest (see page 6468) method.
	OnValidate (see page 6491)	This is OnValidate, a member of class TCustomProvider.

72.1.4.6.1 TCustomProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer (see page 5222) interface, the following events occur:

1.The provider receives a BeforeApplyUpdates (see page 6487) event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeApplyUpdates (see page 6487) event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates (see page 6468) is called without the OwnerData (see page 6483) parameter, on entry to the event handler the OwnerData (see page 6483) parameter contains the value set in the BeforeApplyUpdates (see page 6487) event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates (see page 6468)

AS_ApplyUpdates (see page 5223)

ApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates (see page 6487)

OnUpdateData (see page 6459)

Responding to Client Update Requests

Communicating with the Client Dataset

72.1.4.6.2 TCustomProvider.AfterExecute Event

Occurs after the provider passes an execute command to its dataset.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to return custom information to a client dataset after executing a query or stored procedure. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing queries or stored procedures that do not return a cursor. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_Execute method of the IAppServer (see page 5222) interface, the following events occur:

1.The provider receives a BeforeExecute (see page 6487) event, where it can respond to or change custom information that the client dataset sends.

2.The provider dispatches its Execute (see page 6474) method to execute a query or stored procedure.

3.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeExecute (see page 6487) event handler.

4.The client dataset receives an AfterExecute event, where it can respond to the custom information returned by the provider in its AfterExecute event handler.

See Also

Execute (see page 6474)

AS_Execute (see page 5224)

Execute

beforeexecute (see page 6487)

DataSet (see page 6525)

AfterExecute

Communicating with the Client Dataset

72.1.4.6.3 TCustomProvider.AfterGetParams Event

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

This is AfterGetParams, a member of class TCustomProvider.

72.1.4.6.4 TCustomProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer (see page 5222) interface, the following events occur:

1.The provider receives a BeforeGetRecords (see page 6488) event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeGetRecords (see page 6488) event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider

in its AfterGetRecords event handler.

See Also

AS_GetRecords (see page 5227)

OnGetData (see page 6459)

GetRecords (see page 6476)

AfterGetRecords

BeforeGetRecords (see page 6488)

Communicating with the Client Dataset

Responding to Client Data Requests

72.1.4.6.5 TCustomProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer (see page 5222) interface, the following events occur:

1.The provider receives a BeforeRowRequest (see page 6490) event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeRowRequest (see page 6490) event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 5228)

RowRequest (see page 6480)

AfterRowRequest

BeforeRowRequest (see page 6490)

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

72.1.4.6.6 TCustomProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates (see page 6468) method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData (see page 6483) parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates (see page 6483) event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates (see page 6483) event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates (see page 6483) event handler.

Note: If ApplyUpdates (see page 6468) is called without the OwnerData (see page 6483) parameter, the OwnerData (see page 6483) parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates (see page 6483) event handler but not returned to the caller.

See Also

applyupdates (see page 6468)

AS_ApplyUpdates (see page 5223)

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates (see page 6483)

Responding to Client Update Requests

Communicating with the Client Dataset

72.1.4.6.7 TCustomProvider.BeforeExecute Event

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

This is BeforeExecute, a member of class TCustomProvider.

72.1.4.6.8 TCustomProvider.BeforeGetParams Event

Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to respond to custom information from a client dataset. BeforeGetParams is part of the mechanism by which a provider and a client dataset communicate information during parameter fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches parameter values from the provider, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetParams event, where the OleVariant from the client dataset appears as the OwnerData (see page 6483) parameter. The provider can respond to or change that information before it fetches and encodes the parameter values.

3.The provider generates an OleVariant for the current parameter values of its dataset, if appropriate.

4.The provider receives an AfterGetParams (see page 6485) event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset receives an AfterGetParams (see page 6485) event, where it can respond to the custom information returned by the provider in its AfterGetParams (see page 6485) event handler.

See Also

AS_GetParams (see page 5225)

GetParams (see page 6475)

BeforeGetParams

aftergetparams (see page 6485)

Communicating with the Client Dataset

72.1.4.6.9 TCustomProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData (see page 6483) parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords (see page 6485) event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords (see page 6485) event, where it can respond to the custom information returned by the provider in its AfterGetRecords (see page 6485) event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords (see page 5227)

GetRecords (see page 6476)

BeforeGetRecords

AfterGetRecords (see page 6485)

Communicating with the Client Dataset

Responding to Client Data Requests

72.1.4.6.10 TCustomProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData (see page 6483) parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest (see page 6486) event, where it can encode custom information into its OwnerData (see page 6483) parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest (see page 6486) event, where it can respond to the custom information returned by the provider in its AfterRowRequest (see page 6486) event handler.

See Also

AS_RowRequest (see page 5228)

RowRequest (see page 6480)

AfterRowRequest (see page 6486)

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

72.1.4.6.11 TCustomProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest (see page 6468) method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer (see page 5222) interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest (see page 6468) return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 6468)

IAppServer (see page 5222)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

72.1.4.6.12 TCustomProvider.OnValidate Event

Pascal

```
property OnValidate: TOnValidate;
```

C++

```
__property TOnValidate OnValidate;
```

Description

This is OnValidate, a member of class TCustomProvider.

72.1.5 TCustomResolver Class

TCustomResolver is the base class for components that apply updates on behalf of a provider.

Class Hierarchy

Pascal

```
TCustomResolver = class(TComponent);
```

C++

```
class TCustomResolver : public TComponent;
```

File

Provider

Description

TCustomResolver defines the fundamental properties and methods used by resolver components for applying updates from a client dataset or XML broker and resolving errors that occur as part of that update process.

TCustomResolver has two descendants: TDataSetResolver (see page 6540), which applies updates to a dataset, and TSQLResolver (see page 6564), which applies updates directly to a database server.

Do not create resolver instances. Provider (see page 6501) components create their own TCustomResolver descendants for applying updates.

See Also

TDataSetProvider (see page 6510)

TBaseProvider (see page 6445)

TDataSetResolver (see page 6540)

TSQLResolver (see page 6564)

72.1.5.1 TCustomResolver Members

The following tables list the members exposed by TCustomResolver.

Protected Methods

	Name	Description
	BeginUpdate (see page 6494)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 6495)	Defines a method that deletes a record.
	DoInsert (see page 6495)	Defines a method that inserts a record.
	DoUpdate (see page 6496)	Defines a method that modifies a record.
	EndUpdate (see page 6496)	Provides the interface for a method that performs any finalization after applying updates.
	HandleUpdateError (see page 6497)	Manages the resolving process when errors occur while applying updates.
	InitializeConflictBuffer (see page 6497)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	LogUpdateError (see page 6499)	This is LogUpdateError, a member of class TCustomResolver.
	LogUpdateRecord (see page 6500)	Adds records to the error log if their data has been changed by another application.

Public Methods

	Name	Description
	ApplyUpdates (see page 6494)	Applies updates received from a provider.
	Create (see page 6493)	Creates and initializes a resolver.
	Destroy (see page 6493)	Frees an instance of a resolver object.
	FreeTreeData (see page 6497)	This is FreeTreeData, a member of class TCustomResolver.
	InitKeyFields (see page 6498)	Sets the provider flag pfInKey on all key fields in a delta data packet.
	InitTreeData (see page 6498)	This is InitTreeData, a member of class TCustomResolver.
	InternalBeforeResolve (see page 6498)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
	InternalUpdateRecord (see page 6499)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 6572) object.
	RowRequest (see page 6500)	Returns information from a specified record.

Public Properties

	Name	Description
	Provider (see page 6501)	Indicates the provider that uses this resolver to apply updates.
	UpdateTree (see page 6501)	This is UpdateTree, a member of class TCustomResolver.

72.1.5.2 TCustomResolver.Create Constructor

Creates and initializes a resolver.

Pascal

```
constructor Create(AProvider: TBaseProvider); reintroduce;
```

C++

```
__fastcall TCustomResolver(TBaseProvider AProvider);
```

Description

Provider (see page 6501) components call the TCustomResolver (see page 6491) constructor when they need a resolver component for applying updates. The AProvider parameter specifies the provider that creates and uses the resolver.

Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

See Also

Resolver (see page 6456)

72.1.5.3 TCustomResolver.Destroy Destructor

Frees an instance of a resolver object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomResolver();
```

Description

Applications rarely have to destroy resolvers explicitly. Once a resolver has been assigned as the Resolver property of a provider, the provider is responsible for freeing the resolver.

See Also

System::TObject::Free

Resolver (see page 6456)

72.1.5.4 TCustomResolver Methods

The methods of the TCustomResolver class are listed here.

Protected Methods

	Name	Description
	BeginUpdate (see page 6494)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 6495)	Defines a method that deletes a record.
	DoInsert (see page 6495)	Defines a method that inserts a record.
	DoUpdate (see page 6496)	Defines a method that modifies a record.
	EndUpdate (see page 6496)	Provides the interface for a method that performs any finalization after applying updates.

	HandleUpdateError (see page 6497)	Manages the resolving process when errors occur while applying updates.
	InitializeConflictBuffer (see page 6497)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	LogUpdateError (see page 6499)	This is LogUpdateError, a member of class TCustomResolver.
	LogUpdateRecord (see page 6500)	Adds records to the error log if their data has been changed by another application.

Public Methods

	Name	Description
	ApplyUpdates (see page 6494)	Applies updates received from a provider.
	FreeTreeData (see page 6497)	This is FreeTreeData, a member of class TCustomResolver.
	InitKeyFields (see page 6498)	Sets the provider flag pfInKey on all key fields in a delta data packet.
	InitTreeData (see page 6498)	This is InitTreeData, a member of class TCustomResolver.
	InternalBeforeResolve (see page 6498)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
	InternalUpdateRecord (see page 6499)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 6572) object.
	RowRequest (see page 6500)	Returns information from a specified record.

72.1.5.4.1 TCustomResolver.ApplyUpdates Method

Applies updates received from a provider.

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

Providers call ApplyUpdates to apply updates received from a client dataset or XML broker.

ApplyUpdates generates an OnUpdateData event on the associated provider, and then applies the updates passed in as the Delta parameter, returning an OleVariant that contains any updates that could not be applied.

The Delta parameter specifies a delta packet that contains the insertions, deletions, and modifications to be applied. MaxErrors indicates the maximum number of errors to permit before aborting the update operation. ErrorCount returns the actual number of errors encountered during the update operation. If ErrorCount is greater than MaxErrors, (or an OnUpdateError event returns rrAbort), all updates that have been successfully applied so far are rolled back, and the update process is aborted.

If the associated provider has an OnUpdateError event handler, it is called automatically whenever a record in Delta cannot be updated. In the OnUpdateError handler, applications can correct error conditions and try reapplying an update before returning unsuccessfully updated records back to the client dataset or XML broker.

See Also

OnUpdateData (see page 6459)

OnUpdateError (see page 6460)

applyupdates (see page 6468)

72.1.5.4.2 TCustomResolver.BeginUpdate Method

Provides the interface for a method that performs any initialization before applying updates.

Pascal

```
procedure BeginUpdate; virtual;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The ApplyUpdates (see page 6494) method calls BeginUpdate before applying updates in a delta packet.

As implemented in TCustomResolver (see page 6491), BeginUpdate does nothing. Descendants that need take specific action before applying updates override this method.

See Also

EndUpdate (see page 6496)

ApplyUpdates (see page 6494)

72.1.5.4.3 TCustomResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree) = 0;
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates (see page 6494) method. These helper methods perform the actual updates in the delta packet. ApplyUpdates (see page 6494) wraps the delta packet in a TUpdateTree (see page 6572) object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver (see page 6491) does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoInsert (see page 6495)

DoUpdate (see page 6496)

72.1.5.4.4 TCustomResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree) = 0;
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates (see page 6494) method. These helper methods perform the actual updates in the delta packet. ApplyUpdates (see page 6494) wraps the delta packet in a TUpdateTree (

see page 6572) object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver (see page 6491) does not implement the DoInsert method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoUpdate (see page 6496)

72.1.5.4.5 TCustomResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree) = 0;
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates (see page 6494) method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates (see page 6494) wraps the delta packet in a TUpdateTree (see page 6572) object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver (see page 6491) does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoInsert (see page 6495)

72.1.5.4.6 TCustomResolver.EndUpdate Method

Provides the interface for a method that performs any finalization after applying updates.

Pascal

```
procedure EndUpdate; virtual;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

The ApplyUpdates (see page 6494) method calls EndUpdate after applying updates.

As implemented in TCustomResolver (see page 6491), EndUpdate does nothing. Descendants that need take specific action after applying updates override this method.

See Also

BeginUpdate (see page 6494)

ApplyUpdates (see page 6494)

72.1.5.4.7 TCustomResolver.FreeTreeData Method

Pascal

```
procedure FreeTreeData(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall FreeTreeData(TUpdateTree Tree);
```

Description

This is FreeTreeData, a member of class TCustomResolver.

72.1.5.4.8 TCustomResolver.HandleUpdateError Method

Manages the resolving process when errors occur while applying updates.

Pascal

```
function HandleUpdateError(Tree: TUpdateTree; E: EUpdateError; var MaxErrors: Integer; var
ErrorCount: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandleUpdateError(TUpdateTree Tree, EUpdateError E, int MaxErrors,
int ErrorCount);
```

Description

When a resolver encounters errors in applying updates, it calls HandleUpdateError to determine how to proceed. HandleUpdateError generates an OnUpdateError event on the associated provider so that applications have an opportunity to correct update errors. Depending on the response provided by the OnUpdateError handler (if any), HandleUpdateError logs the unresolved error or signals that the resolver should try to apply the (corrected) record again.

The Tree parameter specifies the TUpdateTree (see page 6572) that wraps all the updates currently being applied. The MaxErrors parameter indicates the maximum number of errors that the resolver will tolerate before the update operation is rolled back. The ErrorCount parameter specifies how many errors have been encountered so far.

HandleUpdateError returns true if the record has been corrected and the resolver should try to reapply the update. It returns false if the record is added to the error log.

See Also

OnUpdateError (see page 6460)

ApplyUpdates (see page 6494)

72.1.5.4.9 TCustomResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree) = 0;
```

Description

The HandleUpdateError (see page 6497) method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with

appropriate values for field's CurValue property.

Descendants of TCustomResolver (see page 6491) implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 6497)

OnUpdateError (see page 6460)

72.1.5.4.10 TCustomResolver.InitKeyFields Method

Sets the provider flag pfInKey on all key fields in a delta data packet.

Pascal

```
procedure InitKeyFields(Tree: TUpdateTree; ADelta: TPacketDataSet); virtual;
```

C++

```
virtual __fastcall InitKeyFields(TUpdateTree Tree, TPacketDataSet ADelta);
```

Description

InitKeyFields is used internally to mark fields from a delta packet that can be used to uniquely identify records in the underlying dataset.

The Tree parameter is the internal TUpdateTree (see page 6572) that the resolver uses to store information about the delta packet. The ADelta parameter is the delta packet of changes to apply. For every field in this delta packet that acts as a key value to uniquely identify records, InitKeyFields adds pfInKey to its ProviderFlags property.

See Also

ProviderFlags

72.1.5.4.11 TCustomResolver.InitTreeData Method

Pascal

```
procedure InitTreeData(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall InitTreeData(TUpdateTree Tree);
```

Description

This is InitTreeData, a member of class TCustomResolver.

72.1.5.4.12 TCustomResolver.InternalBeforeResolve Method

Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

Pascal

```
procedure InternalBeforeResolve(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall InternalBeforeResolve(TUpdateTree Tree);
```

Description

The ApplyUpdates (see page 6494) method wraps delta packets in a TUpdateTree (see page 6572) object. Once this is

complete, InternalBeforeResolve performs any other initialization before the updates are applied.

As implemented in TCustomResolver (see page 6491), InternalBeforeResolve does nothing. Descendants override this method to perform class-specific initialization.

See Also

ApplyUpdates (see page 6494)

inittreedata (see page 6498)

72.1.5.4.13 TCustomResolver.InternalUpdateRecord Method

Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 6572) object.

Pascal

```
function InternalUpdateRecord(Tree: TUpdateTree): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean InternalUpdateRecord(TUpdateTree Tree);
```

Description

InternalUpdateRecord orchestrates the updating process, ensuring that updates are applied in the correct order and that errors are resolved or logged. For each record that must be updated, InternalUpdateRecord

1.Generates a BeforeUpdateRecord event.

2.Applies the update.

3.Generates an AfterUpdateRecord event if the update is successful, or calls HandleUpdateError (see page 6497) to generate an OnUpdateError event.

See Also

ApplyUpdates (see page 6494)

beforeupdaterecord (see page 6458)

AfterUpdateRecord (see page 6458)

OnUpdateError (see page 6460)

HandleUpdateError (see page 6497)

72.1.5.4.14 TCustomResolver.LogUpdateError Method

Pascal

```
procedure LogUpdateError(Tree: TUpdateTree; E: EUpdateError; Response: TResolverResponse);
virtual;
```

C++

```
virtual __fastcall LogUpdateError(TUpdateTree Tree, EUpdateError E, TResolverResponse
Response);
```

Description

This is LogUpdateError, a member of class TCustomResolver.

72.1.5.4.15 TCustomResolver.LogUpdateRecord Method

Adds records to the error log if their data has been changed by another application.

Pascal

```
procedure LogUpdateRecord(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall LogUpdateRecord(TUpdateTree Tree);
```

Description

When the InternalUpdateRecord (see page 6499) method detects that a record includes other changes than those specified by an update in the delta packet, it calls LogUpdateRecord to add the changed and updated record to the error log so that the client dataset or XML broker can refresh the updated record. This error log is eventually returned by the ApplyUpdates (see page 6494) method when the update operation ends.

The Tree parameter represents the set of updates being applied.

See Also

logupdateerror (see page 6499)

InternalUpdateRecord (see page 6499)

72.1.5.4.16 TCustomResolver.RowRequest Method

Returns information from a specified record.

Pascal

```
function RowRequest(Row: OleVariant; Options: TFetchOptions): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant RowRequest(OleVariant Row, TFetchOptions Options);
```

Description

RowRequest is called automatically in response to the associated provider's RowRequest method.

The Row parameter is an OleVariant that describes the record about which information is requested. The resolver uses this to locate the desired record.

The requested information is returned as a delta packet.

See Also

RowRequest (see page 6480)

72.1.5.5 TCustomResolver Properties

The properties of the TCustomResolver class are listed here.

Public Properties

	Name	Description
R	Provider (see page 6501)	Indicates the provider that uses this resolver to apply updates.
R	UpdateTree (see page 6501)	This is UpdateTree, a member of class TCustomResolver.

72.1.5.5.1 TCustomResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TBaseProvider;
```

C++

```
__property TBaseProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to generate events on the provider at appropriate points in the update process.

See Also

Create (see page 6493)

Resolver (see page 6456)

72.1.5.5.2 TCustomResolver.UpdateTree Property

Pascal

```
property UpdateTree: TUpdateTree;
```

C++

```
__property TUpdateTree UpdateTree;
```

Description

This is UpdateTree, a member of class TCustomResolver.

72.1.6 TDataPacketWriter Class

Class Hierarchy

Pascal

```
TDataPacketWriter = class(TCustomPacketWriter);
```

C++

```
class TDataPacketWriter : public TCustomPacketWriter;
```

File

Provider

Description

This is class Provider.TDataPacketWriter.

72.1.6.1 TDataPacketWriter Members

The following tables list the members exposed by TDataPacketWriter.

Protected Fields

	Name	Description
	FBuffer (see page 6462)	This is FBuffer, a member of class TCustomPacketWriter.

Protected Methods

	Name	Description
V	AddAttribute (see page 6463)	This is AddAttribute, a member of class TCustomPacketWriter.
V	AddColumn (see page 6504)	This is AddColumn, a member of class TDataPacketWriter.
V	AddConstraints (see page 6504)	This is AddConstraints, a member of class TDataPacketWriter.
V	AddDataSetAttributes (see page 6504)	This is AddDataSetAttributes, a member of class TDataPacketWriter.
	AddExtraFieldProps (see page 6504)	This is AddExtraFieldProps, a member of class TDataPacketWriter.
V	AddFieldLinks (see page 6505)	This is AddFieldLinks, a member of class TDataPacketWriter.
V	AddIndexDefs (see page 6505)	This is AddIndexDefs, a member of class TDataPacketWriter.
	Check (see page 6463)	This is Check, a member of class TCustomPacketWriter.
	FreeInfoRecords (see page 6505)	This is FreeInfoRecords, a member of class TDataPacketWriter.
	GetFieldIdx (see page 6506)	This is GetFieldIdx, a member of class TDataPacketWriter.
	InitPutProcs (see page 6506)	This is InitPutProcs, a member of class TDataPacketWriter.
	PutADTField (see page 6506)	This is PutADTField, a member of class TDataPacketWriter.
	PutArrayField (see page 6506)	This is PutArrayField, a member of class TDataPacketWriter.
	PutBlobField (see page 6506)	This is PutBlobField, a member of class TDataPacketWriter.
	PutCalcField (see page 6507)	This is PutCalcField, a member of class TDataPacketWriter.
	PutDataSetField (see page 6507)	This is PutDataSetField, a member of class TDataPacketWriter.
	PutField (see page 6507)	This is PutField, a member of class TDataPacketWriter.
	PutStringField (see page 6507)	This is PutStringField, a member of class TDataPacketWriter.
	PutVarBytesField (see page 6508)	This is PutVarBytesField, a member of class TDataPacketWriter.
	PutWideStringField (see page 6508)	This is PutWideStringField, a member of class TDataPacketWriter.
	RefreshPutProcs (see page 6508)	This is RefreshPutProcs, a member of class TDataPacketWriter.
	SetDSWriter (see page 6463)	This is SetDSWriter, a member of class TCustomPacketWriter.
V	WriteDataSet (see page 6508)	This is WriteDataSet, a member of class TDataPacketWriter.
V	WriteMetaData (see page 6509)	This is WriteMetaData, a member of class TDataPacketWriter.

Protected Properties

	Name	Description
R	DSWriter (see page 6464)	This is DSWriter, a member of class TCustomPacketWriter.

Public Events

	Name	Description
	OnGetParams (see page 6510)	This is OnGetParams, a member of class TDataPacketWriter.

Public Methods

	Name	Description
V	Create (see page 6461)	Constructs an object and initializes its data before the object is first used.
V	Destroy (see page 6503)	Disposes of an object instance.
V	GetDataPacket (see page 6505)	This is GetDataPacket, a member of class TDataPacketWriter.
	Reset (see page 6508)	This is Reset, a member of class TDataPacketWriter.

Public Properties

	Name	Description
	Constraints (see page 6509)	This is Constraints, a member of class TDataPacketWriter.

	Options (see page 6509)	This is Options, a member of class TDataPacketWriter.
	PacketOptions (see page 6510)	This is PacketOptions, a member of class TDataPacketWriter.

72.1.6.2 TDataPacketWriter.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataPacketWriter();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

72.1.6.3 TDataPacketWriter Methods

The methods of the TDataPacketWriter class are listed here.

Protected Methods

	Name	Description
V	AddColumn (see page 6504)	This is AddColumn, a member of class TDataPacketWriter.
V	AddConstraints (see page 6504)	This is AddConstraints, a member of class TDataPacketWriter.
V	AddDataSetAttributes (see page 6504)	This is AddDataSetAttributes, a member of class TDataPacketWriter.
	AddExtraFieldProps (see page 6504)	This is AddExtraFieldProps, a member of class TDataPacketWriter.
V	AddFieldLinks (see page 6505)	This is AddFieldLinks, a member of class TDataPacketWriter.
V	AddIndexDefs (see page 6505)	This is AddIndexDefs, a member of class TDataPacketWriter.
	FreeInfoRecords (see page 6505)	This is FreeInfoRecords, a member of class TDataPacketWriter.
	GetFieldIdx (see page 6506)	This is GetFieldIdx, a member of class TDataPacketWriter.
	InitPutProcs (see page 6506)	This is InitPutProcs, a member of class TDataPacketWriter.
	PutADTField (see page 6506)	This is PutADTField, a member of class TDataPacketWriter.
	PutArrayField (see page 6506)	This is PutArrayField, a member of class TDataPacketWriter.
	PutBlobField (see page 6506)	This is PutBlobField, a member of class TDataPacketWriter.

	PutCalcField (see page 6507)	This is PutCalcField, a member of class TDataPacketWriter.
	PutDataSetField (see page 6507)	This is PutDataSetField, a member of class TDataPacketWriter.
	PutField (see page 6507)	This is PutField, a member of class TDataPacketWriter.
	PutStringField (see page 6507)	This is PutStringField, a member of class TDataPacketWriter.
	PutVarBytesField (see page 6508)	This is PutVarBytesField, a member of class TDataPacketWriter.
	PutWideStringField (see page 6508)	This is PutWideStringField, a member of class TDataPacketWriter.
	RefreshPutProcs (see page 6508)	This is RefreshPutProcs, a member of class TDataPacketWriter.
	WriteDataSet (see page 6508)	This is WriteDataSet, a member of class TDataPacketWriter.
	WriteMetaData (see page 6509)	This is WriteMetaData, a member of class TDataPacketWriter.

Public Methods

	Name	Description
	GetDataPacket (see page 6505)	This is GetDataPacket, a member of class TDataPacketWriter.
	Reset (see page 6508)	This is Reset, a member of class TDataPacketWriter.

72.1.6.3.1 TDataPacketWriter.AddColumn Method

Pascal

```
procedure AddColumn(const Info: TPutFieldInfo); virtual;
```

C++

```
virtual __fastcall AddColumn(const TPutFieldInfo Info);
```

Description

This is AddColumn, a member of class TDataPacketWriter.

72.1.6.3.2 TDataPacketWriter.AddConstraints Method

Pascal

```
procedure AddConstraints(DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall AddConstraints(TDataSet DataSet);
```

Description

This is AddConstraints, a member of class TDataPacketWriter.

72.1.6.3.3 TDataPacketWriter.AddDataSetAttributes Method

Pascal

```
procedure AddDataSetAttributes(DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall AddDataSetAttributes(TDataSet DataSet);
```

Description

This is AddDataSetAttributes, a member of class TDataPacketWriter.

72.1.6.3.4 TDataPacketWriter.AddExtraFieldProps Method

Pascal

```
procedure AddExtraFieldProps(Field: TField);
```

C++

```
__fastcall AddExtraFieldProps(TField Field);
```

Description

This is AddExtraFieldProps, a member of class TDataPacketWriter.

72.1.6.3.5 TDataPacketWriter.AddFieldLinks Method

Pascal

```
procedure AddFieldLinks(const Info: TInfoArray); virtual;
```

C++

```
virtual __fastcall AddFieldLinks(const TInfoArray Info);
```

Description

This is AddFieldLinks, a member of class TDataPacketWriter.

72.1.6.3.6 TDataPacketWriter.AddIndexDefs Method

Pascal

```
procedure AddIndexDefs(DataSet: TDataSet; const Info: TInfoArray); virtual;
```

C++

```
virtual __fastcall AddIndexDefs(TDataSet DataSet, const TInfoArray Info);
```

Description

This is AddIndexDefs, a member of class TDataPacketWriter.

72.1.6.3.7 TDataPacketWriter.FreeInfoRecords Method

Pascal

```
procedure FreeInfoRecords(var Info: TInfoArray);
```

C++

```
__fastcall FreeInfoRecords(TInfoArray Info);
```

Description

This is FreeInfoRecords, a member of class TDataPacketWriter.

72.1.6.3.8 TDataPacketWriter.GetDataPacket Method

Pascal

```
procedure GetDataPacket(DataSet: TDataSet; var RecsOut: Integer; out Data: OleVariant);
virtual;
```

C++

```
virtual __fastcall GetDataPacket(TDataSet DataSet, int RecsOut, OleVariant Data);
```

Description

This is GetDataPacket, a member of class TDataPacketWriter.

72.1.6.3.9 TDataPacketWriter.GetFieldIdx Method

Pascal

```
function GetFieldIdx(const FieldName: WideString; const Info: TInfoArray): Integer;
```

C++

```
__fastcall int GetFieldIdx(const BSTR FieldName, const TInfoArray Info);
```

Description

This is GetFieldIdx, a member of class TDataPacketWriter.

72.1.6.3.10 TDataPacketWriter.InitPutProcs Method

Pascal

```
function InitPutProcs(ADataSet: TDataSet; var GlobalIdx: Integer): TInfoArray;
```

C++

```
__fastcall TInfoArray InitPutProcs(TDataSet ADataSet, int GlobalIdx);
```

Description

This is InitPutProcs, a member of class TDataPacketWriter.

72.1.6.3.11 TDataPacketWriter.PutADTField Method

Pascal

```
procedure PutADTField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutADTField(PPutFieldInfo Info);
```

Description

This is PutADTField, a member of class TDataPacketWriter.

72.1.6.3.12 TDataPacketWriter.PutArrayField Method

Pascal

```
procedure PutArrayField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutArrayField(PPutFieldInfo Info);
```

Description

This is PutArrayField, a member of class TDataPacketWriter.

72.1.6.3.13 TDataPacketWriter.PutBlobField Method

Pascal

```
procedure PutBlobField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutBlobField(PPutFieldInfo Info);
```

Description

This is PutBlobField, a member of class TDataPacketWriter.

72.1.6.3.14 TDataPacketWriter.PutCalcField Method

Pascal

```
procedure PutCalcField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutCalcField(PPutFieldInfo Info);
```

Description

This is PutCalcField, a member of class TDataPacketWriter.

72.1.6.3.15 TDataPacketWriter.PutDataSetField Method

Pascal

```
procedure PutDataSetField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutDataSetField(PPutFieldInfo Info);
```

Description

This is PutDataSetField, a member of class TDataPacketWriter.

72.1.6.3.16 TDataPacketWriter.PutField Method

Pascal

```
procedure PutField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutField(PPutFieldInfo Info);
```

Description

This is PutField, a member of class TDataPacketWriter.

72.1.6.3.17 TDataPacketWriter.PutStringField Method

Pascal

```
procedure PutStringField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutStringField(PPutFieldInfo Info);
```

Description

This is PutStringField, a member of class TDataPacketWriter.

72.1.6.3.18 TDataPacketWriter.PutVarBytesField Method

Pascal

```
procedure PutVarBytesField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutVarBytesField(PPutFieldInfo Info);
```

Description

This is PutVarBytesField, a member of class TDataPacketWriter.

72.1.6.3.19 TDataPacketWriter.PutWideStringField Method

Pascal

```
procedure PutWideStringField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutWideStringField(PPutFieldInfo Info);
```

Description

This is PutWideStringField, a member of class TDataPacketWriter.

72.1.6.3.20 TDataPacketWriter.RefreshPutProcs Method

Pascal

```
procedure RefreshPutProcs(ADataSet: TDataSet; var Info: TInfoArray);
```

C++

```
__fastcall RefreshPutProcs(TDataSet ADataSet, TInfoArray Info);
```

Description

This is RefreshPutProcs, a member of class TDataPacketWriter.

72.1.6.3.21 TDataPacketWriter.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class TDataPacketWriter.

72.1.6.3.22 TDataPacketWriter.WriteDataSet Method

Pascal

```
function WriteDataSet(DataSet: TDataSet; var Info: TInfoArray; RecsOut: Integer): Integer;
virtual;
```

C++

```
virtual __fastcall int WriteDataSet(TDataSet DataSet, TInfoArray Info, int RecsOut);
```

Description

This is WriteDataSet, a member of class TDataPacketWriter.

72.1.6.3.23 TDataPacketWriter.WriteMetaData Method

Pascal

```
procedure WriteMetaData(DataSet: TDataSet; const Info: TInfoArray; IsReference: Boolean =
False); virtual;
```

C++

```
virtual __fastcall WriteMetaData(TDataSet DataSet, const TInfoArray Info, Boolean IsReference
= False);
```

Description

This is WriteMetaData, a member of class TDataPacketWriter.

72.1.6.4 TDataPacketWriter Properties

The properties of the TDataPacketWriter class are listed here.

Public Properties

	Name	Description
	Constraints (see page 6509)	This is Constraints, a member of class TDataPacketWriter.
	Options (see page 6509)	This is Options, a member of class TDataPacketWriter.
	PacketOptions (see page 6510)	This is PacketOptions, a member of class TDataPacketWriter.

72.1.6.4.1 TDataPacketWriter.Constraints Property

Pascal

```
property Constraints: Boolean;
```

C++

```
__property Boolean Constraints;
```

Description

This is Constraints, a member of class TDataPacketWriter.

72.1.6.4.2 TDataPacketWriter.Options Property

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

This is Options, a member of class TDataPacketWriter.

72.1.6.4.3 TDataPacketWriter.PacketOptions Property

Pascal

```
property PacketOptions: TGetRecordOptions;
```

C++

```
__property TGetRecordOptions PacketOptions;
```

Description

This is PacketOptions, a member of class TDataPacketWriter.

72.1.6.5 TDataPacketWriter Events

The events of the TDataPacketWriter class are listed here.

Public Events

	Name	Description
	OnGetParams (see page 6510)	This is OnGetParams, a member of class TDataPacketWriter.

72.1.6.5.1 TDataPacketWriter.OnGetParams Event

Pascal

```
property OnGetParams: TGetParamsEvent;
```

C++

```
__property TGetParamsEvent OnGetParams;
```

Description

This is OnGetParams, a member of class TDataPacketWriter.

72.1.7 TDataSetProvider Class

TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.

Class Hierarchy

Pascal

```
TDataSetProvider = class(TBaseProvider);
```

C++

```
class TDataSetProvider : public TBaseProvider;
```

File

Provider

Description

Use TDataSetProvider to provide data from a dataset to a client dataset or XML broker and to resolve updates from that client dataset or XML broker back to the dataset or to its underlying database server. TDataSetProvider can be a part of the same

application as the client dataset or XML broker, or it can be placed in the application server of a multi-tiered application. It serves as a data broker between a remote database server and a client dataset.

TDataSetProvider packages data from a dataset and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database or source dataset.

Client datasets and XML brokers communicate with a provider through the IAppServer (see page 5222) interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 6526) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet

IAppServer (see page 5222)

Determining the Source of Data

Using Provider Components

72.1.7.1 TDataSetProvider Members

The following tables list the members exposed by TDataSetProvider.

Protected Methods

	Name	Description
	CheckDataSet (see page 6514)	This is CheckDataSet, a member of class TDataSetProvider.
	CheckResolver (see page 6449)	Instantiates a resolver for the Resolver (see page 6456) property if it does not already exist.
V	CreateDataPacket (see page 6515)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 6515)	Creates a resolver component for handling the application of updates.
V	DoAfterApplyUpdates (see page 6469)	Generates an AfterApplyUpdates (see page 6483) event.
V	DoAfterExecute (see page 6469)	Generates an AfterExecute (see page 6484) event.
V	DoAfterGetParams (see page 6470)	Generates an AfterGetParams (see page 6485) event.
V	DoAfterGetRecords (see page 6470)	Generates an AfterGetRecords (see page 6485) event.
V	DoAfterRowRequest (see page 6471)	Generates an AfterRowRequest (see page 6486) event.
V	DoAfterUpdateRecord (see page 6450)	Generates an AfterUpdateRecord (see page 6458) event.
V	DoApplyUpdates (see page 6471)	This is DoApplyUpdates, a member of class TCustomProvider.
V	DoBeforeApplyUpdates (see page 6471)	Generates a BeforeApplyUpdates (see page 6487) event.
V	DoBeforeCommit (see page 6450)	This is DoBeforeCommit, a member of class TBaseProvider.
V	DoBeforeExecute (see page 6516)	Generates a BeforeExecute (see page 6534) event.
V	DoBeforeGetParams (see page 6472)	Generates a BeforeGetParams (see page 6488) event.
V	DoBeforeGetRecords (see page 6516)	Generates a BeforeGetRecords (see page 6535) event.
V	DoBeforeRowRequest (see page 6474)	Generates a BeforeRowRequest (see page 6490) event.
V	DoBeforeUpdateRecord (see page 6451)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.

	DoGetProviderAttributes (see page 6517)	Fills a list with information about the dataset that can be used when providing or resolving.
	DoGetRecords (see page 6474)	This is DoGetRecords, a member of class TCustomProvider.
	DoGetTableName (see page 6518)	Generates an OnGetTableName (see page 6539) event.
	DoOnGetData (see page 6451)	Generates an OnGetData (see page 6459) event.
	DoOnUpdateData (see page 6451)	Generates an OnUpdateData (see page 6459) event.
	FetchDetails (see page 6518)	Fills a delta dataset with all nested dataset field values.
	FindRecord (see page 6518)	This is FindRecord, a member of class TDataSetProvider.
	FreeResolver (see page 6452)	Frees the resolver specified by the Resolver (see page 6456) property.
	GetDataSetFromDelta (see page 6519)	Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.
	InternalApplyUpdates (see page 6519)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.
	InternalExecute (see page 6520)	Provides the underlying implementation of the Execute method.
	InternalGetParams (see page 6520)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 6521)	Provides the underlying implementation of the GetRecords (see page 6476) method.
	InternalRowRequest (see page 6522)	Provides the underlying implementation of the RowRequest (see page 6480) method.
	LocateRecord (see page 6522)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	Notification (see page 6522)	Responds to notifications that objects are being created or destroyed.
	Reset (see page 6523)	This is Reset, a member of class TDataSetProvider.
	SetActiveUpdateException (see page 6481)	This is SetActiveUpdateException, a member of class TCustomProvider.
	SetCommandText (see page 6523)	This is SetCommandText, a member of class TDataSetProvider.
	SetParams (see page 6523)	This is SetParams, a member of class TDataSetProvider.
	UpdateRecord (see page 6523)	Retrieves specified data from a source dataset into a delta packet.

Protected Properties

	Name	Description
	PacketDataSet (see page 6456)	This is PacketDataSet, a member of class TBaseProvider.
	TransactionStarted (see page 6528)	This is TransactionStarted, a member of class TDataSetProvider.

Public Methods

	Name	Description
	ApplyUpdates (see page 6468)	This is the overview for the ApplyUpdates method overload.
	Create (see page 6513)	Creates and initializes a TDataSetProvider (see page 6510) object.
	DataRequest (see page 6468)	Generates an OnDataRequest (see page 6490) event.
	Destroy (see page 6513)	Frees an instance of a data set provider.
	Execute (see page 6474)	Executes a query, stored procedure, or custom SQL statement.
	GetParams (see page 6475)	Returns the current parameter values.
	GetRecords (see page 6476)	This is the overview for the GetRecords method overload.
	RowRequest (see page 6480)	Returns information from a specified record.

Public Properties

	Name	Description
	ActiveUpdateException (see page 6524)	This is ActiveUpdateException, a member of class TDataSetProvider.
R	Data (see page 6482)	Denotes a data packet created by the provider.
	OwnerData (see page 6527)	This is OwnerData, a member of class TDataSetProvider.
R	Params (see page 6527)	Stores the parameter values currently in use.
R	Resolver (see page 6456)	Provides access to the resolver component that applies updates and resolves update errors.

Published Events

	Name	Description
	AfterApplyUpdates (see page 6529)	Occurs after the provider applies updates.
	AfterExecute (see page 6530)	Occurs after the provider passes an execute command to its dataset.

	AfterGetParams (see page 6531)	This is AfterGetParams, a member of class TDataSetProvider.
	AfterGetRecords (see page 6531)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 6532)	Occurs after the provider fetches new information about the current record.
	AfterUpdateRecord (see page 6533)	Occurs when the associated resolver has just updated a record.
	BeforeApplyUpdates (see page 6533)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 6534)	This is BeforeExecute, a member of class TDataSetProvider.
	BeforeGetParams (see page 6534)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 6535)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 6536)	Occurs before the provider creates a delta packet with information about the current record.
	BeforeUpdateRecord (see page 6537)	This is BeforeUpdateRecord, a member of class TDataSetProvider.
	OnDataRequest (see page 6537)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 6537)	Occurs after a provider retrieves data but before it is sent to the client.
	OnGetDataSetProperties (see page 6538)	Occurs when the provider adds additional information to a data packet it is creating.
	OnGetTableName (see page 6539)	This is OnGetTableName, a member of class TDataSetProvider.
	OnUpdateData (see page 6539)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 6539)	Occurs when a provider can't update a record because of an error condition.
	OnValidate (see page 6540)	This is OnValidate, a member of class TDataSetProvider.

Published Properties

	Name	Description
	Constraints (see page 6525)	Controls whether database constraints are passed to the client so that it can check them locally.
	DataSet (see page 6525)	Specifies the dataset from which the provider provides data and to which it applies updates.
	Exported (see page 6526)	Allows client applications to pass calls to the provider when it resides on an application server.
	Options (see page 6526)	Influences what is included in data packets and how the information in data packets is used.
	ResolveToDataSet (see page 6528)	Specifies whether updates should be applied to a dataset or directly to the database server.
	UpdateMode (see page 6528)	Controls how the provider locates records when applying updates.

72.1.7.2 TDataSetProvider.Create Constructor

Creates and initializes a TDataSetProvider (see page 6510) object.

Pascal

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

C++

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

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

See Also

Create (see page 6447)

72.1.7.3 TDataSetProvider.Destroy Destructor

Frees an instance of a data set provider.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSetProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the reference to the provider is not nil before calling its Destroy method. Destroy frees all helper objects, including the resolver component referenced by the Resolver (see page 6456) property.

See Also

System::TObject::Free

Resolver (see page 6456)

72.1.7.4 TDataSetProvider Methods

The methods of the TDataSetProvider class are listed here.

Protected Methods

	Name	Description
	CheckDataSet (see page 6514)	This is CheckDataSet, a member of class TDataSetProvider.
V	CreateDataPacket (see page 6515)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 6515)	Creates a resolver component for handling the application of updates.
V	DoBeforeExecute (see page 6516)	Generates a BeforeExecute (see page 6534) event.
V	DoBeforeGetRecords (see page 6516)	Generates a BeforeGetRecords (see page 6535) event.
V	DoGetProviderAttributes (see page 6517)	Fills a list with information about the dataset that can be used when providing or resolving.
V	DoGetTableName (see page 6518)	Generates an OnGetTableName (see page 6539) event.
V	FetchDetails (see page 6518)	Fills a delta dataset with all nested dataset field values.
V	FindRecord (see page 6518)	This is FindRecord, a member of class TDataSetProvider.
V	GetDataSetFromDelta (see page 6519)	Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.
V	InternalApplyUpdates (see page 6519)	Provides the underlying implementation of the ApplyUpdates (see page 6468) method.
V	InternalExecute (see page 6520)	Provides the underlying implementation of the Execute method.
V	InternalGetParams (see page 6520)	Returns the requested parameter values as a Variant array.
V	InternalGetRecords (see page 6521)	Provides the underlying implementation of the GetRecords (see page 6476) method.
V	InternalRowRequest (see page 6522)	Provides the underlying implementation of the RowRequest (see page 6480) method.
V	LocateRecord (see page 6522)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
V	Notification (see page 6522)	Responds to notifications that objects are being created or destroyed.
	Reset (see page 6523)	This is Reset, a member of class TDataSetProvider.
V	SetCommandText (see page 6523)	This is SetCommandText, a member of class TDataSetProvider.
V	SetParams (see page 6523)	This is SetParams, a member of class TDataSetProvider.
V	UpdateRecord (see page 6523)	Retrieves specified data from a source dataset into a delta packet.

72.1.7.4.1 TDataSetProvider.CheckDataSet Method

Pascal

```
procedure CheckDataSet;
```

C++

```
__fastcall CheckDataSet();
```

Description

This is CheckDataSet, a member of class TDataSetProvider.

72.1.7.4.2 TDataSetProvider.CreateDataPacket Method

Generates a data packet for the data associated with the provider.

Pascal

```
procedure CreateDataPacket(PacketOpts: TGetRecordOptions; ProvOpts: TProviderOptions; var
RecsOut: Integer; var Data: OleVariant); override;
```

C++

```
virtual __fastcall CreateDataPacket(TGetRecordOptions PacketOpts, TProviderOptions ProvOpts,
int RecsOut, OleVariant Data);
```

Description

CreateDataPacket is used internally to generate the data packets that are returned by GetRecords (see page 6476).

The PacketOpts parameter indicates what information (other than record values) is included in the data packet. It reflects any Options (see page 6526) passed to the GetRecords (see page 6476) method.

The ProvOpts parameter reflects the current value of the Options (see page 6526) property.

The RecsOut parameter specifies the number of records desired in the packet and returns the actual number of records in the generated data packet.

The Data (see page 6482) parameter returns the generated data packet.

See Also

GetRecords (see page 6476)

DataSet (see page 6525)

Options (see page 6455)

72.1.7.4.3 TDataSetProvider.CreateResolver Method

Creates a resolver component for handling the application of updates.

Pascal

```
function CreateResolver: TCustomResolver; override;
```

C++

```
virtual __fastcall TCustomResolver CreateResolver();
```

Description

CreateResolver is called automatically when a resolver is needed to implement one of the provider's methods. CreateResolver returns a newly instantiated resolver object, which is assigned to the Resolver (see page 6456) property. The type of resolver created depends on the ResolveToDataSet (see page 6528) property:

When ResolveToDataSet (see page 6528) is true, CreateResolver returns a TDataSetResolver (see page 6540) component, which applies updates directly to the dataset specified by the DataSet (see page 6525) property.

When ResolveToDataSet (see page 6528) is false, CreateResolver returns a TSQLResolver (see page 6564) component,

which applies updates directly to the database server associated with DataSet (see page 6525).

See Also

Resolver (see page 6456)

ResolveToDataSet (see page 6528)

72.1.7.4.4 TDataSetProvider.DoBeforeExecute Method

Generates a BeforeExecute (see page 6534) event.

Pascal

```
procedure DoBeforeExecute(const CommandText: WideString; var Params: OleVariant; var
OwnerData: OleVariant); override;
```

C++

```
virtual __fastcall DoBeforeExecute(const BSTR CommandText, OleVariant Params, OleVariant
OwnerData);
```

Description

The Execute (see page 6474) method calls DoBeforeExecute to generate the BeforeExecute (see page 6534) event before it executes CommandText or executes an associated query or stored procedure.

CommandText is either an optional SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. If Options (see page 6526) includes poAllowCommandText, DoBeforeExecute sends CommandText to the associated dataset so that it can execute the provided statement instead of its query or stored procedure.

Params (see page 6527) contains any parameter values that should be applied to CommandText or to the query or stored procedure before execution. DoBeforeExecute assigns these parameter values to the dataset (if appropriate).

OwnerData (see page 6527) supplies custom information to the BeforeExecute (see page 6534) event handler. The event handler can change this value. The returned value is then passed to an AfterExecute (see page 6530) event handler and then returned to the caller of the Execute (see page 6474) method.

See Also

Execute (see page 6474)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

DataSet (see page 6525)

72.1.7.4.5 TDataSetProvider.DoBeforeGetRecords Method

Generates a BeforeGetRecords (see page 6535) event.

Pascal

```
procedure DoBeforeGetRecords(Count: Integer; Options: Integer; const CommandText: WideString;
var Params: OleVariant; var OwnerData: OleVariant); override;
```

C++

```
virtual __fastcall DoBeforeGetRecords(int Count, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

The GetRecords (see page 6476) method calls DoBeforeGetRecords to generate the BeforeGetRecords (see page 6535) event before it fetches data and assembles it into a data packet.

Count is the number of records requested. TDataSetProvider (see page 6510) does not use this parameter.

Options (see page 6526) indicates whether the data packet should include metadata, whether it should start with the first record, and whether the data packet should be encoded in XML. In TDataSetProvider (see page 6510), DoBeforeGetRecords ignores this parameter.

CommandText specifies either an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. If Options (see page 6526) includes poAllowCommandText, DoBeforeGetRecords sends CommandText to the associated dataset so that it can generate the set of records.

Params (see page 6527) contains any parameter values that should be applied to CommandText or to a query or stored procedure before fetching data into a data packet. DoBeforeGetRecords assigns these values to the associated dataset, if appropriate.

OwnerData (see page 6527) supplies custom information to the BeforeGetRecords (see page 6535) event handler. The event handler can change this value. The returned value is then passed to an AfterGetRecords (see page 6531) event handler and then (optionally) returned to the caller of the GetRecords (see page 6476) method.

See Also

GetRecords (see page 6476)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

DataSet (see page 6525)

72.1.7.4.6 TDataSetProvider.DoGetProviderAttributes Method

Fills a list with information about the dataset that can be used when providing or resolving.

Pascal

```
procedure DoGetProviderAttributes(DataSet: TDataSet; List: TList); virtual;
```

C++

```
virtual __fastcall DoGetProviderAttributes(TDataSet DataSet, TList List);
```

Description

DoGetProviderAttributes is used internally to fetch information about the dataset. This includes both information provided by the dataset (using the IProviderSupport method PSGetAttributes) and custom information supplied by an OnGetDataSetProperties (see page 6538) event handler.

The DataSet (see page 6525) parameter specifies the dataset that stores the information. The List parameter is the list that is filled by DoGetProviderAttributes. DoGetProviderAttributes adds information to the list in Name/Value pairs.

See Also

PSGetAttributes

OnGetDataSetProperties (see page 6538)

72.1.7.4.7 TDataSetProvider.DoGetTableName Method

Generates an OnGetTableName (see page 6539) event.

Pascal

```
procedure DoGetTableName(DataSet: TDataSet; var TableName: WideString); virtual;
```

C++

```
virtual __fastcall DoGetTableName(TDataSet DataSet, BSTR TableName);
```

Description

The resolver calls DoGetTableName when it initializes the information it needs to apply updates. Override this method to provide an alternate mechanism for determining the name of the dataset that is the target of an update operation.

DataSet (see page 6525) is the dataset that is the target of the update operation.

The name of the target dataset as it should appear in generated SQL statements is returned as the TableName parameter.

See Also

ongettablename (see page 6539)

Resolver (see page 6456)

72.1.7.4.8 TDataSetProvider.FetchDetails Method

Fills a delta dataset with all nested dataset field values.

Pascal

```
procedure FetchDetails(Source: TDataSet; Delta: TDataSet); override;
```

C++

```
virtual __fastcall FetchDetails(TDataSet Source, TDataSet Delta);
```

Description

FetchDetails is used internally to fetch the current values of any nested datasets when the provider is executing its RowRequest (see page 6480) method. Source is a source dataset that has the current values for all fields. Delta is a delta dataset that must be updated so that its nested detail fields match the values in Source.

See Also

RowRequest (see page 6480)

72.1.7.4.9 TDataSetProvider.FindRecord Method

Pascal

```
function FindRecord(Source: TDataSet; Delta: TDataSet; UpdateMode: TUpdateMode): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean FindRecord(TDataSet Source, TDataSet Delta, TUpdateMode UpdateMode);
```

Description

This is FindRecord, a member of class TDataSetProvider.

72.1.7.4.10 TDataSetProvider.GetDataSetFromDelta Method

Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.

Pascal

```
function GetDataSetFromDelta(ATree: TUpdateTree; Source: TDataSet; Delta: TDataSet; Mode:
TUpdateMode): TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDataSetFromDelta(TUpdateTree ATree, TDataSet Source, TDataSet
Delta, TUpdateMode Mode);
```

Description

GetDataSetFromDelta is used internally to fetch a specific record when the source dataset is unidirectional.

ATree is a TUpdateTree (see page 6572) object that represents the update or data fetch operation.

Source is a source dataset. Because it is unidirectional, however, it is not necessarily positioned on the correct record.

Delta is a delta dataset that indicates the desired record.

Mode indicates the criteria to use for locating a record on the server from the dataset that Source uses. The values that follow these criteria come from Delta.

GetDataSetFromDelta returns a unidirectional dataset that corresponds to Source but that contains a single record based on the criteria in Delta and Mode.

See Also

IsUniDirectional

DataSet (see page 6525)

72.1.7.4.11 TDataSetProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 6468) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates (see page 6468) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 6533) event and before generating the AfterApplyUpdates (see page 6529) event.

If the Options (see page 6526) property includes poReadOnly, InternalApplyUpdates raises an exception. Otherwise, it tells the dataset to start a transaction, calls the ApplyUpdates (see page 6468) method of its Resolver (see page 6456) component to handle the actual update and resolving process, and tells the dataset to complete the transaction.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

Resolver (see page 6456)

ApplyUpdates (see page 6494)

CreateResolver (see page 6515)

Options (see page 6455)

applyupdates (see page 6468)

72.1.7.4.12 TDataSetProvider.InternalExecute Method

Provides the underlying implementation of the Execute method.

Pascal

```
procedure InternalExecute(const CommandText: WideString; var Params: OleVariant); override;
```

C++

```
virtual __fastcall InternalExecute(const BSTR CommandText, OleVariant Params);
```

Description

The Execute method calls InternalExecute to execute the appropriate SQL statement after calling DoBeforeExecute to generate a BeforeExecute event and apply the CommandText and Params. As Implemented in TCustomProvider, InternalExecute does nothing. Descendants override this method to pass the execute command on to an associated dataset.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

See Also

Execute (see page 6474)

DoBeforeExecute (see page 6472)

72.1.7.4.13 TDataSetProvider.InternalGetParams Method

Returns the requested parameter values as a Variant array.

Pascal

```
function InternalGetParams(Types: TParamTypes = AllParamTypes): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetParams(TParamTypes Types = AllParamTypes);
```

Description

The provider uses InternalGetParams internally whenever it needs to assemble the current parameter values into an OleVariant. This provides the underlying implementation of the GetParams (see page 6475) method, as well as assembling parameter values for methods that return output parameters (such as Execute (see page 6474) or GetRecords (see page 6476)).

The Types parameter limits the parameter values that are returned to a subset of all possible parameter types. If this parameter is omitted, all parameter values are returned.

See Also

GetParams (see page 6475)

Execute (see page 6474)

GetRecords (see page 6476)

DataSet (see page 6525)

72.1.7.4.14 TDataSetProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 6476) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 6476) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 6516) to apply the CommandText and Params (see page 6527) and generate a BeforeGetRecords (see page 6535) event. InternalGetRecords calls CreateDataPacket (see page 6515) to create a data packet that reflects the current property settings of the provider. It then generates an OnGetData (see page 6537) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options (see page 6526) indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML but use UTF8 to encode extended characters).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options (see page 6526) property does not include poAllowCommandText.

Params (see page 6527) specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable, Params (see page 6527) specifies field values that filter the result set. On exit from GetRecords (see page 6476), Params (see page 6527) returns any output parameters.

Records are returned as a data packet in a Variant.

Note: If the DataSet (see page 6525) property is not set, InternalGetRecords raises an exception.

See Also

Data (see page 6482)

CreateDataPacket (see page 6515)

OnGetData (see page 6459)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

DoBeforeGetRecords (see page 6516)

72.1.7.4.15 TDataSetProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 6480) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; Options: TFetchOptions): OleVariant;
override;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions Options);
```

Description

The RowRequest (see page 6480) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 6536) event and before generating an AfterRowRequest (see page 6532) event. InternalRowRequest passes this request on to its associated Resolver (see page 6456).

The Row parameter is an OleVariant that describes the record about which information is requested.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 6480)

Resolver (see page 6456)

RowRequest (see page 6500)

72.1.7.4.16 TDataSetProvider.LocateRecord Method

Positions a data set on the record that corresponds to the current update in a specified delta packet.

Pascal

```
procedure LocateRecord(Source: TDataSet; Delta: TDataSet); override;
```

C++

```
virtual __fastcall LocateRecord(TDataSet Source, TDataSet Delta);
```

Description

LocateRecord is called indirectly by the RowRequest (see page 6480) method to locate a record before fetching information about it. The Source parameter is the dataset containing current record values. The Delta parameter is a delta packet that identifies the desired type of information and information identifying the desired record.

LocateRecord uses the criteria specified by UpdateMode (see page 6528) to identify the desired record.

See Also

RowRequest (see page 6480)

UpdateMode (see page 6457)

72.1.7.4.17 TDataSetProvider.Notification Method

Responds to notifications that objects are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to add a check that the dataset specified by the DataSet (see page 6525) property is not being deleted. If DataSet (see page 6525) is about to be destroyed, Notification sets the DataSet (see page 6525) property to nil (Delphi) or NULL (C++).

See Also

OnUpdateError (see page 6460)

Create

72.1.7.4.18 TDataSetProvider.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class TDataSetProvider.

72.1.7.4.19 TDataSetProvider.SetCommandText Method

Pascal

```
procedure SetCommandText(const CommandText: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const BSTR CommandText);
```

Description

This is SetCommandText, a member of class TDataSetProvider.

72.1.7.4.20 TDataSetProvider.SetParams Method

Pascal

```
procedure SetParams(Values: OleVariant); virtual;
```

C++

```
virtual __fastcall SetParams(OleVariant Values);
```

Description

This is SetParams, a member of class TDataSetProvider.

72.1.7.4.21 TDataSetProvider.UpdateRecord Method

Retrieves specified data from a source dataset into a delta packet.

Pascal

```
procedure UpdateRecord(Source: TDataSet; Delta: TDataSet; BlobsOnly: Boolean; KeyOnly:
Boolean); override;
```

C++

```
virtual __fastcall UpdateRecord(TDataSet Source, TDataSet Delta, Boolean BlobsOnly, Boolean
KeyOnly);
```

Description

UpdateRecord is called indirectly by the RowRequest (see page 6480) method to fetch information about a specified record. The Source parameter is the dataset containing indicated record. The Delta parameter is a delta packet that returns the requested data. BlobsOnly indicates whether all field values should be provided (BlobsOnly == false) or only Blob field values (BlobsOnly == true).

See Also

RowRequest (see page 6480)

72.1.7.5 TDataSetProvider Properties

The properties of the TDataSetProvider class are listed here.

Protected Properties

	Name	Description
	TransactionStarted (see page 6528)	This is TransactionStarted, a member of class TDataSetProvider.

Public Properties

	Name	Description
	ActiveUpdateException (see page 6524)	This is ActiveUpdateException, a member of class TDataSetProvider.
	OwnerData (see page 6527)	This is OwnerData, a member of class TDataSetProvider.
R	Params (see page 6527)	Stores the parameter values currently in use.

Published Properties

	Name	Description
	Constraints (see page 6525)	Controls whether database constraints are passed to the client so that it can check them locally.
	DataSet (see page 6525)	Specifies the dataset from which the provider provides data and to which it applies updates.
	Exported (see page 6526)	Allows client applications to pass calls to the provider when it resides on an application server.
	Options (see page 6526)	Influences what is included in data packets and how the information in data packets is used.
	ResolveToDataSet (see page 6528)	Specifies whether updates should be applied to a dataset or directly to the database server.
	UpdateMode (see page 6528)	Controls how the provider locates records when applying updates.

72.1.7.5.1 TDataSetProvider.ActiveUpdateException Property

Pascal

```
property ActiveUpdateException: Exception;
```

C++

```
__property Exception ActiveUpdateException;
```

Description

This is ActiveUpdateException, a member of class TDataSetProvider.

72.1.7.5.2 TDataSetProvider.Constraints Property

Controls whether database constraints are passed to the client so that it can check them locally.

Pascal

```
property Constraints: Boolean;
```

C++

```
__property Boolean Constraints;
```

Description

Set Constraints to specify whether data constraints defined for the provider's associated dataset are sent along with the metadata for the dataset. Sending constraints allows client datasets to check for constraint violations locally, so that updates are less likely to fail when sent back to the database server.

When Constraints is true, constraints are sent. To prevent the provider from sending constraint information, set Constraints to false.

Note: Before constraints can be sent to client applications, they must be imported to the data dictionary on the application server. Use SQL explorer to import constraints to the data dictionary.

See Also

Constraints

DataSet (see page 6525)

72.1.7.5.3 TDataSetProvider.DataSet Property

Specifies the dataset from which the provider provides data and to which it applies updates.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to link the dataset provider to a dataset that it represents. The dataset provider packages the records from (and other information about) DataSet into data packets that can be used by client datasets and XML brokers. DataSet must be defined if the provider is used to create data packets.

If the ResolveToDataSet (see page 6528) property is true, the provider applies updates from the application server directly to this dataset. Otherwise, the provider applies updates directly to the underlying database server.

Note: The provider communicates with DataSet using the IProviderSupport interface. This interface is available in all datasets, although in TDataSet most of the IProviderSupport methods are simple stubs that either do nothing or generate an exception. The dataset classes that ship with the IDE (dbExpress datasets, BDE-enabled datasets, ADO-enabled datasets, InterBase Express datasets,

and client datasets) override these protected methods to implement the IProviderSupport interface in a more useful fashion. If

you are using custom dataset components, check with the vendor whether they can work with a dataset provider.

See Also

applyupdates (see page 6468)

Data (see page 6482)

IProviderSupport

Connecting a Client Dataset to Another Dataset in the Same Application

Determining the Source of Data

72.1.7.5.4 TDataSetProvider.Exported Property

Allows client applications to pass calls to the provider when it resides on an application server.

Pascal

```
property Exported: Boolean;
```

C++

```
__property Boolean Exported;
```

Description

Use Exported to indicate whether the provider is made available to clients of a remote data module that owns it. When Exported is true, client applications can specify the provider as a target for calls made using the IAppServer interface. When Exported is false, IAppServer calls that specify the provider cause an exception.

Note: In addition to the Exported property, the provider must be registered with the remote data module for clients to access it using the data module's IAppServer interface. When the provider's Owner is the remote data module, it is automatically registered. If the provider has a different owner, the server application must explicitly register providers using the remote data module's RegisterProvider method.

See Also

IAppServer (see page 5222)

Providers

registerprovider

72.1.7.5.5 TDataSetProvider.Options Property

Influences what is included in data packets and how the information in data packets is used.

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

Set Options to customize how the provider communicates with client datasets. Options is a set that specifies

Whether nested detail sets and BLOB data are included in data packets or fetched separately.

Whether data packets include field component properties such as formats, display names, and maximum & minimum values.

Whether data packets are provided on a read-only basis, and whether specific types of changes (insertions, edits, or deletions)

are disabled.

Whether changes to master table fields should cause corresponding updates to detail tables.

Whether a single update can affect multiple server records.

Whether the client's records are refreshed when it applies updates.

Whether the client can send SQL statements that override the provider's dataset.

Note: Not all options are supported by an InternetExpress client. Specifically, InternetExpress clients don't support poFetchBlobsOnDemand, poFetchDetailsOnDemand, poDisableInserts, poDisableEdits, psDisableDeletes, poPropogateChanges, and poAllowCommandText.

See Also

Data (see page 6482)

applyupdates (see page 6468)

Overriding the Dataset On the Application Server

Setting Options That Influence the Data Packets

72.1.7.5.6 TDataSetProvider.OwnerData Property

Pascal

```
property OwnerData: OleVariant;
```

C++

```
__property OleVariant OwnerData;
```

Description

This is OwnerData, a member of class TDataSetProvider.

72.1.7.5.7 TDataSetProvider.Params Property

Stores the parameter values currently in use.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

This protected property acts as a temporary cache for parameter values from clients. It is set to the parameter values supplied by calls to the Execute (see page 6474) or GetRecords (see page 6476) method.

Note: If the dataset provider is responding to multiple client datasets or XML brokers

, the value of Params can only be used within the context of a single call to SetParams (see page 6523), Execute (see page 6474), or GetRecords (see page 6476). After that, other client calls can change the value of parameters.

See Also

Execute (see page 6474)

GetRecords (see page 6476)

72.1.7.5.8 TDataSetProvider.ResolveToDataSet Property

Specifies whether updates should be applied to a dataset or directly to the database server.

Pascal

```
property ResolveToDataSet: Boolean;
```

C++

```
__property Boolean ResolveToDataSet;
```

Description

Set ResolveToDataSet to specify how updates are applied. When ResolveToDataSet is true, the Resolver (see page 6456) property is set to a TDataSetResolver (see page 6540) component, which applies updates directly to the dataset specified by the DataSet (see page 6525) property. This can be useful if the application uses the events on the dataset component or if the dataset does not represent the data from a database server (for example, a client dataset).

When ResolveToDataSet is false, the Resolver (see page 6456) property is set to a TSQLResolver (see page 6564) component, which applies updates directly to the database server associated with DataSet (see page 6525). This can be more efficient, because it skips the intermediate step of using the dataset. It is the only way to resolve updates if the source dataset is read-only (for example, a unidirectional dataset).

See Also

DataSet (see page 6525)

Resolver (see page 6456)

Choosing How to Apply Updates Using a Datasetm Provider

72.1.7.5.9 TDataSetProvider.TransactionStarted Property

Pascal

```
property TransactionStarted: Boolean;
```

C++

```
__property Boolean TransactionStarted;
```

Description

This is TransactionStarted, a member of class TDataSetProvider.

72.1.7.5.10 TDataSetProvider.UpdateMode Property

Controls how the provider locates records when applying updates.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria for locating a record in the dataset when applying updates. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original

values of fields that have been modified.

Note: To control how the provider locates records by specifying individual fields, use the field components' ProviderFlags property.

See Also

applyupdates (see page 6468)

ProviderFlags

Influencing How Updates Are Applied

72.1.7.6 TDataSetProvider Events

The events of the TDataSetProvider class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 6529)	Occurs after the provider applies updates.
	AfterExecute (see page 6530)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 6531)	This is AfterGetParams, a member of class TDataSetProvider.
	AfterGetRecords (see page 6531)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 6532)	Occurs after the provider fetches new information about the current record.
	AfterUpdateRecord (see page 6533)	Occurs when the associated resolver has just updated a record.
	BeforeApplyUpdates (see page 6533)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 6534)	This is BeforeExecute, a member of class TDataSetProvider.
	BeforeGetParams (see page 6534)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 6535)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 6536)	Occurs before the provider creates a delta packet with information about the current record.
	BeforeUpdateRecord (see page 6537)	This is BeforeUpdateRecord, a member of class TDataSetProvider.
	OnDataRequest (see page 6537)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 6537)	Occurs after a provider retrieves data but before it is sent to the client.
	OnGetDataSetProperties (see page 6538)	Occurs when the provider adds additional information to a data packet it is creating.
	OnGetTableName (see page 6539)	This is OnGetTableName, a member of class TDataSetProvider.
	OnUpdateData (see page 6539)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 6539)	Occurs when a provider can't update a record because of an error condition.
	OnValidate (see page 6540)	This is OnValidate, a member of class TDataSetProvider.

72.1.7.6.1 TDataSetProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to

communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, on entry to the event handler the OwnerData parameter contains the value set in the BeforeApplyUpdates event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates (see page 6468)

AS_ApplyUpdates (see page 5223)

ApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates (see page 6487)

OnUpdateData (see page 6459)

Responding to Client Update Requests

Communicating with the Client Dataset

72.1.7.6.2 TDataSetProvider.AfterExecute Event

Occurs after the provider passes an execute command to its dataset.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to return custom information to a client dataset after executing a query or stored procedure. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing queries or stored procedures that do not return a cursor. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_Execute method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeExecute event, where it can respond to or change custom information that the client dataset sends.

2.The provider dispatches its Execute method to execute a query or stored procedure.

3.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

4.The client dataset receives an AfterExecute event, where it can respond to the custom information returned by the provider in its AfterExecute event handler.

See Also

Execute (see page 6474)

AS_Execute (see page 5224)

Execute

beforeexecute (see page 6487)

DataSet (see page 6525)

AfterExecute

Communicating with the Client Dataset

72.1.7.6.3 TDataSetProvider.AfterGetParams Event

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

This is AfterGetParams, a member of class TDataSetProvider.

72.1.7.6.4 TDataSetProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords (see page 5227)

OnGetData (see page 6459)

GetRecords (see page 6476)

AfterGetRecords

BeforeGetRecords (see page 6488)

Communicating with the Client Dataset

Responding to Client Data Requests

72.1.7.6.5 TDataSetProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 5228)

RowRequest (see page 6480)

AfterRowRequest

BeforeRowRequest (see page 6490)

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

72.1.7.6.6 TDataSetProvider.AfterUpdateRecord Event

Occurs when the associated resolver has just updated a record.

Pascal

```
property AfterUpdateRecord: TAfterUpdateRecordEvent;
```

C++

```
__property TAfterUpdateRecordEvent AfterUpdateRecord;
```

Description

Write an AfterUpdateRecord event handler to provide special processing immediately after a record has been successfully updated. For example, use AfterUpdateRecord to maintain a log of successful updates.

See Also

beforeupdaterecord (see page 6458)

AfterApplyUpdates (see page 6483)

72.1.7.6.7 TDataSetProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, the OwnerData parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates event handler but not returned to the

caller.

See Also

applyupdates (see page 6468)

AS_ApplyUpdates (see page 5223)

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates (see page 6483)

Responding to Client Update Requests

Communicating with the Client Dataset

72.1.7.6.8 TDataSetProvider.BeforeExecute Event

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

This is BeforeExecute, a member of class TDataSetProvider.

72.1.7.6.9 TDataSetProvider.BeforeGetParams Event

Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to respond to custom information from a client dataset. BeforeGetParams is part of the mechanism by which a provider and a client dataset communicate information during parameter fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches parameter values from the provider, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetParams event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it fetches and encodes the parameter values.

3.The provider generates an OleVariant for the current parameter values of its dataset, if appropriate.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset receives an AfterGetParams event, where it can respond to the custom information returned by the provider in its AfterGetParams event handler.

See Also

AS_GetParams (see page 5225)

GetParams (see page 6475)

BeforeGetParams

aftergetparams (see page 6485)

Communicating with the Client Dataset

72.1.7.6.10 TDataSetProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
```

```
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords (see page 5227)

GetRecords (see page 6476)

BeforeGetRecords

AfterGetRecords (see page 6485)

Communicating with the Client Dataset

Responding to Client Data Requests

72.1.7.6.11 TDataSetProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 5228)

RowRequest (see page 6480)

AfterRowRequest (see page 6486)

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

72.1.7.6.12 TDataSetProvider.BeforeUpdateRecord Event

Pascal

```
property BeforeUpdateRecord: TBeforeUpdateRecordEvent;
```

C++

```
__property TBeforeUpdateRecordEvent BeforeUpdateRecord;
```

Description

This is BeforeUpdateRecord, a member of class TDataSetProvider.

72.1.7.6.13 TDataSetProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 6468)

IAppServer (see page 5222)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

72.1.7.6.14 TDataSetProvider.OnGetData Event

Occurs after a provider retrieves data but before it is sent to the client.

Pascal

```
property OnGetData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnGetData;
```

Description

Write an OnGetData event handler to perform any custom data processing the provider should handle before returning data to a client. The OnGetData event occurs after a provider fetches data from its dataset but before the data is sent on to the client. Using the methods of the DataSet parameter, data can be edited in the OnGetData event handler before sending it to the client dataset or XML broker.

Note: OnGetData occurs when the Data property is read, as well as when the GetRecords method is called.

See Also

GetRecords (see page 6476)

OnUpdateData (see page 6459)

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

Responding to Client Data Requests

72.1.7.6.15 TDataSetProvider.OnGetDataSetProperties Event

Occurs when the provider adds additional information to a data packet it is creating.

Pascal

```
property OnGetDataSetProperties: TGetDSProps;
```

C++

```
__property TGetDSProps OnGetDataSetProperties;
```

Description

Write an OnGetDataSetProperties event handler to include application-defined information in the data packets. This information can later be returned using the GetOptionalParam method of the client dataset that represents the data in the packet.

Note: The following names are reserved and can't be used as the name for storing custom information:

UNIQUE_KEY	DEFAULT_ORDER	CHANGE_LOG	SERVER_COL
CONSTRAINTS	DATASET_CONTEXT	DATASET_DELTA	LCID
BDERECORD_X	TABLE_NAME	MD_FIELDLINKS	UPDATEMODE

See Also

GetOptionalParam

SetOptionalParam

Adding Custom Information to Data Packets

72.1.7.6.16 TDataSetProvider.OnGetTableName Event

Pascal

```
property OnGetTableName: TGetTableNameEvent;
```

C++

```
__property TGetTableNameEvent OnGetTableName;
```

Description

This is OnGetTableName, a member of class TDataSetProvider.

72.1.7.6.17 TDataSetProvider.OnUpdateData Event

Occurs when a provider begins to apply updates.

Pascal

```
property OnUpdateData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnUpdateData;
```

Description

Write an OnUpdateData event to handle special processing before the provider begins applying updates received from the client dataset or XML broker. Using the methods belonging to the DataSet parameter, an application can

Examine data (for example, for values or data changes that should not permitted), and raise exceptions that cancel applying of updates before they occur.

Change data (for example encrypting or decrypting values) before it is sent on to the source dataset or database server.

See Also

applyupdates (see page 6468)

beforeupdaterecord (see page 6458)

Adding Custom Information to Data Packets

Editing Delta Packets Before Updating the Database

Influencing How Updates Are Applied

72.1.7.6.18 TDataSetProvider.OnUpdateError Event

Occurs when a provider can't update a record because of an error condition.

Pascal

```
property OnUpdateError: TResolverErrorEvent;
```

C++

```
__property TResolverErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to error conditions that arise when the provider applies a client dataset's updates. The OnUpdateError event handler is called for each record that can't be updated.

Warning: The provider can't detect when an update conflicts with another user's changes to a memo field. This situation will not

generate an OnUpdateError event.

See Also

applyupdates (see page 6468)

OnReconcileError

NewValue

OldValue

CurValue

Resolving Update Errors On the Provider

72.1.7.6.19 TDataSetProvider.OnValidate Event

Pascal

```
property OnValidate: TOnValidate;
```

C++

```
__property TOnValidate OnValidate;
```

Description

This is OnValidate, a member of class TDataSetProvider.

72.1.8 TDataSetResolver Class

TDataSetResolver applies updates to a dataset on behalf of a provider.

Class Hierarchy

Pascal

```
TDataSetResolver = class(TCustomResolver);
```

C++

```
class TDataSetResolver : public TCustomResolver;
```

File

Provider

Description

TDataSetProvider (see page 6510) components use TDataSetResolver to apply updates to a dataset when their ResolveToDataSet property is true. Otherwise, they use TSQLResolver (see page 6564).

Do not create instances of TDataSetResolver. Provider (see page 6545) components create their own TDataSetResolver objects for applying updates.

See Also

TDataSetProvider (see page 6510)

TBaseProvider (see page 6445)

TSQLResolver (see page 6564)

TDataSetProvider (see page 6510)

72.1.8.1 TDataSetResolver Members

The following tables list the members exposed by TDataSetResolver.

Protected Methods

	Name	Description
	BeginUpdate (see page 6542)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 6543)	Defines a method that deletes a record.
	DoInsert (see page 6543)	Defines a method that inserts a record.
	DoUpdate (see page 6543)	Defines a method that modifies a record.
	EndUpdate (see page 6544)	Provides the interface for a method that performs any finalization after applying updates.
	HandleUpdateError (see page 6497)	Manages the resolving process when errors occur while applying updates.
	InitializeConflictBuffer (see page 6544)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	LogUpdateError (see page 6499)	This is LogUpdateError, a member of class TCustomResolver.
	LogUpdateRecord (see page 6500)	Adds records to the error log if their data has been changed by another application.

Protected Properties

	Name	Description
	Provider (see page 6545)	Indicates the provider that uses this resolver to apply updates.

Public Methods

	Name	Description
	ApplyUpdates (see page 6494)	Applies updates received from a provider.
	Create (see page 6541)	Creates and initializes a dataset resolver.
	Destroy (see page 6493)	Frees an instance of a resolver object.
	FreeTreeData (see page 6497)	This is FreeTreeData, a member of class TCustomResolver.
	InitKeyFields (see page 6498)	Sets the provider flag pfInKey on all key fields in a delta data packet.
	InitTreeData (see page 6498)	This is InitTreeData, a member of class TCustomResolver.
	InternalBeforeResolve (see page 6545)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
	InternalUpdateRecord (see page 6499)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 6572) object.
	RowRequest (see page 6500)	Returns information from a specified record.

Public Properties

	Name	Description
	UpdateTree (see page 6501)	This is UpdateTree, a member of class TCustomResolver.

72.1.8.2 TDataSetResolver.Create Constructor

Creates and initializes a dataset resolver.

Pascal

```
constructor Create(AProvider: TDataSetProvider); reintroduce;
```

C++

```
__fastcall TDataSetResolver(TDataSetProvider AProvider);
```

Description

Provider (see page 6545) components call Create when they need a resolver component for applying updates to a dataset. The AProvider parameter specifies the provider that creates and uses the resolver.

Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

See Also

Resolver (see page 6456)

72.1.8.3 TDataSetResolver Methods

The methods of the TDataSetResolver class are listed here.

Protected Methods

	Name	Description
	BeginUpdate (see page 6542)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 6543)	Defines a method that deletes a record.
	DoInsert (see page 6543)	Defines a method that inserts a record.
	DoUpdate (see page 6543)	Defines a method that modifies a record.
	EndUpdate (see page 6544)	Provides the interface for a method that performs any finalization after applying updates.
	InitializeConflictBuffer (see page 6544)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Public Methods

	Name	Description
	InternalBeforeResolve (see page 6545)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

72.1.8.3.1 TDataSetResolver.BeginUpdate Method

Provides the interface for a method that performs any initialization before applying updates.

Pascal

```
procedure BeginUpdate; override;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The ApplyUpdates method calls BeginUpdate before applying updates in a delta packet.

As implemented in TCustomResolver, BeginUpdate does nothing. Descendants that need take specific action before applying updates override this method.

See Also

EndUpdate (see page 6496)

ApplyUpdates (see page 6494)

72.1.8.3.2 TDataSetResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree);
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoInsert (see page 6495)

DoUpdate (see page 6496)

72.1.8.3.3 TDataSetResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree);
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver does not implement the DoInsert method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoUpdate (see page 6496)

72.1.8.3.4 TDataSetResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree);
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoInsert (see page 6495)

72.1.8.3.5 TDataSetResolver.EndUpdate Method

Provides the interface for a method that performs any finalization after applying updates.

Pascal

```
procedure EndUpdate; override;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

The ApplyUpdates method calls EndUpdate after applying updates.

As implemented in TCustomResolver, EndUpdate does nothing. Descendants that need take specific action after applying updates override this method.

See Also

BeginUpdate (see page 6494)

ApplyUpdates (see page 6494)

72.1.8.3.6 TDataSetResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree);
```

Description

The HandleUpdateError method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with appropriate

values for field's CurValue property.

Descendants of TCustomResolver implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 6497)

OnUpdateError (see page 6460)

72.1.8.3.7 TDataSetResolver.InternalBeforeResolve Method

Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

Pascal

```
procedure InternalBeforeResolve(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InternalBeforeResolve(TUpdateTree Tree);
```

Description

The ApplyUpdates method wraps delta packets in a TUpdateTree object. Once this is complete, InternalBeforeResolve performs any other initialization before the updates are applied.

As implemented in TCustomResolver, InternalBeforeResolve does nothing. Descendants override this method to perform class-specific initialization.

See Also

ApplyUpdates (see page 6494)

inittreedata (see page 6498)

72.1.8.4 TDataSetResolver Properties

The properties of the TDataSetResolver class are listed here.

Protected Properties

	Name	Description
	Provider (see page 6545)	Indicates the provider that uses this resolver to apply updates.

72.1.8.4.1 TDataSetResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TDataSetProvider;
```

C++

```
__property TDataSetProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to

generate events on the provider at appropriate points in the update process.

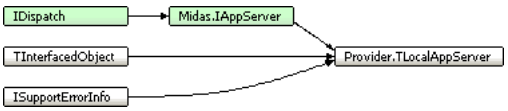
See Also

Create (see page 6493)

Resolver (see page 6456)

72.1.9 TLocalAppServer Class

Class Hierarchy



Pascal

```
TLocalAppServer = class(TInterfacedObject, IAppServer, ISupportErrorInfo);
```

C++

```
class TLocalAppServer : public TInterfacedObject, public IAppServer, public ISupportErrorInfo;
```

File

Provider

Description

This is class Provider.TLocalAppServer.

72.1.9.1 TLocalAppServer Members

The following tables list the members exposed by TLocalAppServer.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 6548)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 6549)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 6549)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 6550)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 6551)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 6552)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 6553)	Returns information from a specified record of the provider's dataset.
	GetIDsOfNames (see page 6554)	This is GetIDsOfNames, a member of class TLocalAppServer.
	GetTypeInfo (see page 6554)	This is GetTypeInfo, a member of class TLocalAppServer.
	GetTypeInfoCount (see page 6554)	This is GetTypeInfoCount, a member of class TLocalAppServer.
	InterfaceSupportsErrorInfo (see page 6554)	This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.
	Invoke (see page 6555)	This is Invoke, a member of class TLocalAppServer.

Public Methods

	Name	Description
	Create (see page 6547)	This is the overview for the Create constructor overload.
V	Destroy (see page 6547)	Disposes of an object instance.
V	SafeCallException (see page 6555)	Handles exceptions in methods declared using the safecall calling convention.

72.1.9.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TLocalAppServer.Create (TCustomProvider) (see page 6547)	This is Create, a member of class TLocalAppServer.
	TLocalAppServer.Create (TDataset) (see page 6547)	This is Create, a member of class TLocalAppServer.

72.1.9.2.1 TLocalAppServer.Create Constructor (TCustomProvider)

Pascal

```
constructor Create(AProvider: TCustomProvider); overload;
```

C++

```
__fastcall TLocalAppServer(TCustomProvider AProvider);
```

Description

This is Create, a member of class TLocalAppServer.

72.1.9.2.2 TLocalAppServer.Create Constructor (TDataset)

Pascal

```
constructor Create(ADataset: TDataset); overload;
```

C++

```
__fastcall TLocalAppServer(TDataset ADataset);
```

Description

This is Create, a member of class TLocalAppServer.

72.1.9.3 TLocalAppServer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocalAppServer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

72.1.9.4 TLocalAppServer Methods

The methods of the TLocalAppServer class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 6548)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 6549)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 6549)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 6550)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 6551)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 6552)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 6553)	Returns information from a specified record of the provider's dataset.
	GetIDsOfNames (see page 6554)	This is GetIDsOfNames, a member of class TLocalAppServer.
	GetTypeInfo (see page 6554)	This is GetTypeInfo, a member of class TLocalAppServer.
	GetTypeInfoCount (see page 6554)	This is GetTypeInfoCount, a member of class TLocalAppServer.
	InterfaceSupportsErrorInfo (see page 6554)	This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.
	Invoke (see page 6555)	This is Invoke, a member of class TLocalAppServer.

Public Methods

	Name	Description
	SafeCallException (see page 6555)	Handles exceptions in methods declared using the safecall calling convention.

72.1.9.4.1 TLocalAppServer.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates

applyupdates (see page 6468)

BeforeApplyUpdates (see page 6487)

AfterApplyUpdates (see page 6483)

BeforeApplyUpdates

AfterApplyUpdates

AS_ApplyUpdates

as_applyupdates

Communicating with the Client Dataset

72.1.9.4.2 TLocalAppServer.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest (see page 6490)

DataRequest

Communicating with the Client Dataset

72.1.9.4.3 TLocalAppServer.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 6525)

beforeexecute (see page 6487)

AfterExecute (see page 6484)

Params

Execute

Execute (see page 6474)

BeforeExecute

AfterExecute

Options (see page 6455)

PackageParams

UnpackParams

Communicating with the Client Dataset

72.1.9.4.4 TLocalAppServer.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 6525)

Params

FetchParams

GetParams (see page 6475)

BeforeGetParams (see page 6488)

aftergetparams (see page 6485)

AS_Execute (see page 5224)

UnpackParams

Communicating with the Client Dataset

72.1.9.4.5 TLocalAppServer.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that uses the IAppServer interface of an application server. When calling any other IAppServer method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider

by setting their ProviderName property to one of these names.

See Also

providers

ProviderName

Providers

Communicating with the Client Dataset

72.1.9.4.6 TLocalAppServer.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data

Params

GetNextPacket

GetRecords (see page 6476)

BeforeGetRecords

BeforeGetRecords (see page 6488)

AfterGetRecords (see page 6485)

Options (see page 6455)

PackageParams

UnpackParams

Communicating with the Client Dataset

72.1.9.4.7 TLocalAppServer.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest (see page 6480)

BeforeRowRequest (see page 6490)

AfterRowRequest (see page 6486)

FetchBlobs

FetchDetails

RefreshRecord

Communicating with the Client Dataset

72.1.9.4.8 TLocalAppServer.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TLocalAppServer.

72.1.9.4.9 TLocalAppServer.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TLocalAppServer.

72.1.9.4.10 TLocalAppServer.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TLocalAppServer.

72.1.9.4.11 TLocalAppServer.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TGUID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TGUID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.

72.1.9.4.12 TLocalAppServer.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TLocalAppServer.

72.1.9.4.13 TLocalAppServer.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

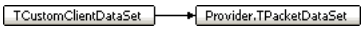
System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject, System::TObject::SafeCallException simply returns E_UNEXPECTED. This is the appropriate response for classes that do no support interfaces

72.1.10 TPacketDataSet Class

TPacketDataSet represents a data packet or delta packet that is used by a provider.

Class Hierarchy



Pascal

```
TPacketDataSet = class(TCustomClientDataSet);
```

C++

```
class TPacketDataSet : public TCustomClientDataSet;
```

File

Provider

Description

TPacketDataSet represents a data packet that a provider creates to represent its dataset's records and metadata or a delta data packet that a provider receives when it should apply updates. Provider (see page 6441) components create instances of TPacketDataSet to represent data packets before they are sent to clients and to represent delta packets after they are received from clients.

TPacketDataSet provides a convenient way for providers and resolvers to work with data packets. As a descendant of TCustomClientDataSet, TPacketDataSet allows providers and resolvers to use any of the standard dataset properties and methods to manipulate the data packet's records. In addition, TPacketDataSet introduces new properties and methods that describe what information goes into the data packet or how updates should be applied.

Applications do not need to work directly with packet datasets. They are mainly of use to component writers who want to create custom provider components.

See Also

TCUstomClientDataSet

TCustomProvider (see page 6464)

72.1.10.1 TPacketDataSet Members

The following tables list the members exposed by TPacketDataSet.

Protected Methods

	Name	Description
	DataEvent (see page 6558)	This is DataEvent, a member of class TPacketDataSet.
	GetStateFieldValue (see page 6559)	This is GetStateFieldValue, a member of class TPacketDataSet.
	InternalClose (see page 6560)	This is InternalClose, a member of class TPacketDataSet.
	InternalInitRecord (see page 6560)	This is InternalInitRecord, a member of class TPacketDataSet.
	InternalOpen (see page 6561)	This is InternalOpen, a member of class TPacketDataSet.
	SetFieldData (see page 6561)	This is SetFieldData, a member of class TPacketDataSet.
	SetStateFieldValue (see page 6561)	This is SetStateFieldValue, a member of class TPacketDataSet.

Protected Properties

	Name	Description
	WritingCurValues (see page 6563)	This is WritingCurValues, a member of class TPacketDataSet.

Public Methods

	Name	Description
	AssignCurValues (see page 6557)	This is the overview for the AssignCurValues method overload.
	Create (see page 6556)	Creates and initializes instance of TPacketDataSet (see page 6555).
	CreateFromDelta (see page 6558)	This is CreateFromDelta, a member of class TPacketDataSet.
	HasCurValues (see page 6559)	Indicates whether the CurValue property is available for fields in the packet dataset.
	HasMergeConflicts (see page 6559)	Indicates whether a current record can be merged with updates in a delta packet.
	InitAltRecBuffers (see page 6560)	Fetches values for the CurValue, OldValue, and NewValue properties of fields.
	UpdateKind (see page 6561)	Indicates the type of update the current record represents.

Public Properties

	Name	Description
	NewValuesModified (see page 6562)	Indicates whether the NewValue property of any field in the current record has changed.
	StreamMetaData (see page 6562)	This is StreamMetaData, a member of class TPacketDataSet.
	UseCurValues (see page 6562)	Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.

72.1.10.2 TPacketDataSet.Create Constructor

Creates and initializes instance of TPacketDataSet (see page 6555).

Pascal

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

C++

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

Description

Most applications do not need to create packet datasets because they are created by the provider components that use them.Create initializes the properties of the packet dataset.

AOwner is the component that is responsible for freeing the TPacketDataSet (see page 6555) instance. If AOwner is nil (Delphi) or NULL (C++), the application must explicitly free the packet data set object.

See Also

Owner

72.1.10.3 TPacketDataSet Methods

The methods of the TPacketDataSet class are listed here.

Protected Methods

	Name	Description
	DataEvent (see page 6558)	This is DataEvent, a member of class TPacketDataSet.
	GetStateFieldValue (see page 6559)	This is GetStateFieldValue, a member of class TPacketDataSet.
	InternalClose (see page 6560)	This is InternalClose, a member of class TPacketDataSet.
	InternalInitRecord (see page 6560)	This is InternalInitRecord, a member of class TPacketDataSet.
	InternalOpen (see page 6561)	This is InternalOpen, a member of class TPacketDataSet.
	SetFieldData (see page 6561)	This is SetFieldData, a member of class TPacketDataSet.
	SetStateFieldValue (see page 6561)	This is SetStateFieldValue, a member of class TPacketDataSet.

Public Methods

	Name	Description
	AssignCurValues (see page 6557)	This is the overview for the AssignCurValues method overload.
	CreateFromDelta (see page 6558)	This is CreateFromDelta, a member of class TPacketDataSet.
	HasCurValues (see page 6559)	Indicates whether the CurValue property is available for fields in the packet dataset.
	HasMergeConflicts (see page 6559)	Indicates whether a current record can be merged with updates in a delta packet.
	InitAltRecBuffers (see page 6560)	Fetches values for the CurValue, OldValue, and NewValue properties of fields.
	UpdateKind (see page 6561)	Indicates the type of update the current record represents.

72.1.10.3.1 AssignCurValues Method

This is the overview for the AssignCurValues method overload.

Overload List

	Name	Description
	TPacketDataSet.AssignCurValues (TDataSet) (see page 6558)	Updates the current field value for each field in the packet dataset.
	TPacketDataSet.AssignCurValues (Variant) (see page 6558)	Updates the current field value for each field in the packet dataset.

72.1.10.3.1.1 TPacketDataSet.AssignCurValues Method (TDataSet)

Updates the current field value for each field in the packet dataset.

Pascal

```
procedure AssignCurValues(Source: TDataSet); overload;
procedure AssignCurValues(const CurValues: Variant); overload;
```

C++

```
__fastcall AssignCurValues(TDataSet Source);
__fastcall AssignCurValues(const Variant CurValues);
```

Description

When a resolver component encounters an update error, it calls AssignCurValues to set the CurValue property of fields before triggering the OnUpdateError event handler.

If AssignCurValues is passed a dataset as the Source parameter, then for each field in the packet dataset, the CurValue property is set to the value of the field with the same name in the source dataset.

If AssignCurValues is passed a Variant array as the CurValues parameter, then each entry in the array is used to assign the CurValue property of a field in the packet dataset. CurValues can have one of two forms: an array of field values, or an array of Variant arrays, where each of those arrays has two elements: a field name and a value. In the first form, field values are assigned by position. In the second form, fields are located based on name and the CurValue property assigned accordingly.

See Also

CurValue

OnUpdateError (see page 6460)

72.1.10.3.2 TPacketDataSet.CreateFromDelta Method

Pascal

```
procedure CreateFromDelta(Source: TPacketDataSet);
```

C++

```
__fastcall CreateFromDelta(TPacketDataSet Source);
```

Description

This is CreateFromDelta, a member of class TPacketDataSet.

72.1.10.3.3 TPacketDataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TPacketDataSet.

72.1.10.3.4 TPacketDataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TPacketDataSet.

72.1.10.3.5 TPacketDataSet.HasCurValues Method

Indicates whether the CurValue property is available for fields in the packet dataset.

Pascal

```
function HasCurValues: Boolean;
```

C++

```
__fastcall Boolean HasCurValues();
```

Description

When an error occurs while applying an updated record, resolver components attempt to locate the current record in the underlying database (or the dataset to which updates are applied.) This current record can then be used when trying to determine the cause of the update error. If the current record can be located, HasCurValues returns true and the fields in the packet dataset have their CurValue properties set from this record. If the record can't be located (for example, if another user changed a key field value or deleted the record), then HasCurValues returns false and the CurValue property of packet dataset fields can't be used.

See Also

UseCurValues (see page 6562)

CurValue

72.1.10.3.6 TPacketDataSet.HasMergeConflicts Method

Indicates whether a current record can be merged with updates in a delta packet.

Pascal

```
function HasMergeConflicts: Boolean;
```

C++

```
__fastcall Boolean HasMergeConflicts();
```

Description

When resolving update errors in an OnUpdateError event handler, applications can specify that the updated record should be merged automatically with the current field values of the record. (To do this, set the Response parameter to rrMerge.) However, it is not always possible to merge client edits with the current record. If the updated record includes changes to field values that were also changed by other users, the merge will fail and HasMergeConflicts returns true.

See Also

CurValue

NewValue

OnUpdateError (see page 6460)

72.1.10.3.7 TPacketDataSet.InitAltRecBuffers Method

Fetches values for the CurValue, OldValue, and NewValue properties of fields.

Pascal

```
procedure InitAltRecBuffers(CheckModified: Boolean = True);
```

C++

```
__fastcall InitAltRecBuffers(Boolean CheckModified = True);
```

Description

While applying updates, InitAltRecBuffers is called automatically to ensure that fields in the packet dataset have values for the CurValue, OldValue, and NewValue properties. The CheckModified parameter specifies whether InitAltRecBuffers should check if the current record has been modified when it sets values. This check allows the packet dataset to reflect appropriate values when applying updates that are changed by an OnUpdateError event handler.

See Also

OldValue

NewValue

CurValue

OnUpdateError (see page 6460)

72.1.10.3.8 TPacketDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TPacketDataSet.

72.1.10.3.9 TPacketDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: TRecordBuffer); override;
```

C++

```
virtual __fastcall InternalInitRecord(TRecordBuffer Buffer);
```

Description

This is InternalInitRecord, a member of class TPacketDataSet.

72.1.10.3.10 TPacketDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TPacketDataSet.

72.1.10.3.11 TPacketDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TPacketDataSet.

72.1.10.3.12 TPacketDataSet.SetStateFieldValue Method

Pascal

```
procedure SetStateFieldValue(State: TDataSetState; Field: TField; const Value: Variant);
override;
```

C++

```
virtual __fastcall SetStateFieldValue(TDataSetState State, TField Field, const Variant Value);
```

Description

This is SetStateFieldValue, a member of class TPacketDataSet.

72.1.10.3.13 TPacketDataSet.UpdateKind Method

Indicates the type of update the current record represents.

Pascal

```
function UpdateKind: TUpdateKind;
```

C++

```
__fastcall TUpdateKind UpdateKind();
```

Description

Call UpdateKind while applying updates to determine the type of update the current record represents. UpdateKind returns

ukInsert if the current record is a newly inserted record.

ukDelete if the current record represents a record that should be deleted.

ukModify if the current record represents updates to an existing record.

72.1.10.4 TPacketDataSet Properties

The properties of the TPacketDataSet class are listed here.

Protected Properties

	Name	Description
	WritingCurValues (see page 6563)	This is WritingCurValues, a member of class TPacketDataSet.

Public Properties

	Name	Description
R	NewValuesModified (see page 6562)	Indicates whether the NewValue property of any field in the current record has changed.
	StreamMetaData (see page 6562)	This is StreamMetaData, a member of class TPacketDataSet.
	UseCurValues (see page 6562)	Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.

72.1.10.4.1 TPacketDataSet.NewValuesModified Property

Indicates whether the NewValue property of any field in the current record has changed.

Pascal

```
property NewValuesModified: Boolean;
```

C++

```
__property Boolean NewValuesModified;
```

Description

Read NewValuesModified to determine whether any fields in the current record were edited during the resolving process.

See Also

NewValue

72.1.10.4.2 TPacketDataSet.StreamMetaData Property

Pascal

```
property StreamMetaData: Boolean;
```

C++

```
__property Boolean StreamMetaData;
```

Description

This is StreamMetaData, a member of class TPacketDataSet.

72.1.10.4.3 TPacketDataSet.UseCurValues Property

Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.

Pascal

```
property UseCurValues: Boolean;
```

C++

```
__property Boolean UseCurValues;
```

Description

UseCurValues is set during the resolving process when a record is corrected in an OnUpdateError event handler. This allows the resolver to reapply a corrected record rather than using the originally provided values as the basis of an update.

See Also

HasCurValues (see page 6559)

CurValue

OldValue

OnUpdateError (see page 6460)

72.1.10.4.4 TPacketDataSet.WritingCurValues Property

Pascal

```
property WritingCurValues: Boolean;
```

C++

```
__property Boolean WritingCurValues;
```

Description

This is WritingCurValues, a member of class TPacketDataSet.

72.1.11 TProvider Class

TProvider provides data to and applies updates from a client dataset.

Class Hierarchy

Pascal

```
TProvider = class(TDataSetProvider);
```

C++

```
class TProvider : public TDataSetProvider;
```

File

Provider

Description

TProvider is provided for backward compatibility. It is identical to TDataSetProvider (see page 6510). New applications should use TDataSetProvider (see page 6510) instead.

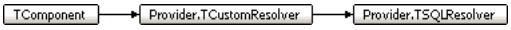
See Also

TDataSetProvider (see page 6510)

72.1.12 TSQLResolver Class

TSQLResolver applies updates to a database server on behalf of a provider.

Class Hierarchy



Pascal

```
TSQLResolver = class(TCustomResolver);
```

C++

```
class TSQLResolver : public TCustomResolver;
```

File

Provider

Description

Provider (see page 6571) components use TSQLResolver to apply updates from a client application directly to a database server when their ResolveToDataSet property is false. Otherwise, they use TDataSetResolver (see page 6540).

Do not create instances of TSQLResolver. Provider (see page 6571) components create their own TSQLResolver objects for applying updates.

See Also

TBaseProvider (see page 6445)

TDataSetResolver (see page 6540)

TDataSetProvider (see page 6510)

72.1.12.1 TSQLResolver Members

The following tables list the members exposed by TSQLResolver.

Protected Methods

	Name	Description
	BeginUpdate (see page 6494)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 6566)	Defines a method that deletes a record.
	DoExecSQL (see page 6567)	This is DoExecSQL, a member of class TSQLResolver.
	DoGetValues (see page 6567)	This is DoGetValues, a member of class TSQLResolver.
	DoInsert (see page 6567)	Defines a method that inserts a record.
	DoUpdate (see page 6568)	Defines a method that modifies a record.
	EndUpdate (see page 6496)	Provides the interface for a method that performs any finalization after applying updates.
	GenDeleteSQL (see page 6568)	This is GenDeleteSQL, a member of class TSQLResolver.
	GenInsertSQL (see page 6569)	This is GenInsertSQL, a member of class TSQLResolver.
	GenSelectSQL (see page 6569)	This is GenSelectSQL, a member of class TSQLResolver.
	GenUpdateSQL (see page 6569)	This is GenUpdateSQL, a member of class TSQLResolver.
	GenWhereSQL (see page 6569)	This is GenWhereSQL, a member of class TSQLResolver.
	GetProvider (see page 6570)	This is GetProvider, a member of class TSQLResolver.
	HandleUpdateError (see page 6497)	Manages the resolving process when errors occur while applying updates.

	InitializeConflictBuffer (see page 6570)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	InternalDoUpdate (see page 6570)	This is InternalDoUpdate, a member of class TSQLResolver.
	LogUpdateError (see page 6499)	This is LogUpdateError, a member of class TCustomResolver.
	LogUpdateRecord (see page 6500)	Adds records to the error log if their data has been changed by another application.
	UseFieldInUpdate (see page 6571)	This is UseFieldInUpdate, a member of class TSQLResolver.
	UseFieldInWhere (see page 6571)	This is UseFieldInWhere, a member of class TSQLResolver.

Protected Properties

	Name	Description
	Provider (see page 6571)	Indicates the provider that uses this resolver to apply updates.

Public Methods

	Name	Description
	ApplyUpdates (see page 6494)	Applies updates received from a provider.
	Create (see page 6565)	Creates and initializes an SQL resolver.
	Destroy (see page 6565)	Destroys an instance of TSQLResolver (see page 6564).
	FreeTreeData (see page 6568)	This is FreeTreeData, a member of class TSQLResolver.
	InitKeyFields (see page 6498)	Sets the provider flag pfInKey on all key fields in a delta data packet.
	InitTreeData (see page 6570)	This is InitTreeData, a member of class TSQLResolver.
	InternalBeforeResolve (see page 6498)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
	InternalUpdateRecord (see page 6499)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 6572) object.
	RowRequest (see page 6500)	Returns information from a specified record.

Public Properties

	Name	Description
	UpdateTree (see page 6501)	This is UpdateTree, a member of class TCustomResolver.

72.1.12.2 TSQLResolver.Create Constructor

Creates and initializes an SQL resolver.

Pascal

```
constructor Create(AProvider: TDataSetProvider); reintroduce;
```

C++

```
__fastcall TSQLResolver(TDataSetProvider AProvider);
```

Description

Provider (see page 6571) components call Create when they need a resolver component for applying updates to a database server. Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

The AProvider parameter specifies the provider that creates and uses the resolver.

See Also

Resolver (see page 6456)

72.1.12.3 TSQLResolver.Destroy Destructor

Destroys an instance of TSQLResolver (see page 6564).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLResolver();
```

Description

Most applications should not delete instances of TSQLResolver (see page 6564). SQL resolvers are freed automatically by the provider components that use them.

See Also

System::TObject::Free

Resolver (see page 6456)

72.1.12.4 TSQLResolver Methods

The methods of the TSQLResolver class are listed here.

Protected Methods

	Name	Description
	DoDelete (see page 6566)	Defines a method that deletes a record.
	DoExecSQL (see page 6567)	This is DoExecSQL, a member of class TSQLResolver.
	DoGetValues (see page 6567)	This is DoGetValues, a member of class TSQLResolver.
	DoInsert (see page 6567)	Defines a method that inserts a record.
	DoUpdate (see page 6568)	Defines a method that modifies a record.
	GenDeleteSQL (see page 6568)	This is GenDeleteSQL, a member of class TSQLResolver.
	GenInsertSQL (see page 6569)	This is GenInsertSQL, a member of class TSQLResolver.
	GenSelectSQL (see page 6569)	This is GenSelectSQL, a member of class TSQLResolver.
	GenUpdateSQL (see page 6569)	This is GenUpdateSQL, a member of class TSQLResolver.
	GenWhereSQL (see page 6569)	This is GenWhereSQL, a member of class TSQLResolver.
	GetProvider (see page 6570)	This is GetProvider, a member of class TSQLResolver.
	InitializeConflictBuffer (see page 6570)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	InternalDoUpdate (see page 6570)	This is InternalDoUpdate, a member of class TSQLResolver.
	UseFieldInUpdate (see page 6571)	This is UseFieldInUpdate, a member of class TSQLResolver.
	UseFieldInWhere (see page 6571)	This is UseFieldInWhere, a member of class TSQLResolver.

Public Methods

	Name	Description
	FreeTreeData (see page 6568)	This is FreeTreeData, a member of class TSQLResolver.
	InitTreeData (see page 6570)	This is InitTreeData, a member of class TSQLResolver.

72.1.12.4.1 TSQLResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree);
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoInsert (see page 6495)

DoUpdate (see page 6496)

72.1.12.4.2 TSQLResolver.DoExecSQL Method

Pascal

```
procedure DoExecSQL(SQL: TWideStringList; Params: TParams); virtual;
```

C++

```
virtual __fastcall DoExecSQL(TWideStringList SQL, TParams Params);
```

Description

This is DoExecSQL, a member of class TSQLResolver.

72.1.12.4.3 TSQLResolver.DoGetValues Method

Pascal

```
procedure DoGetValues(SQL: TWideStringList; Params: TParams; DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall DoGetValues(TWideStringList SQL, TParams Params, TDataSet DataSet);
```

Description

This is DoGetValues, a member of class TSQLResolver.

72.1.12.4.4 TSQLResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree);
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver does not implement the DoInsert method. Descendants override this method to insert records in their own

specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoUpdate (see page 6496)

72.1.12.4.5 TSQLResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree);
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 6494)

DoDelete (see page 6495)

DoInsert (see page 6495)

72.1.12.4.6 TSQLResolver.FreeTreeData Method

Pascal

```
procedure FreeTreeData(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall FreeTreeData(TUpdateTree Tree);
```

Description

This is FreeTreeData, a member of class TSQLResolver.

72.1.12.4.7 TSQLResolver.GenDeleteSQL Method

Pascal

```
procedure GenDeleteSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string);
virtual;
```

C++

```
virtual __fastcall GenDeleteSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params,
UnicodeString Alias);
```

Description

This is GenDeleteSQL, a member of class TSQLResolver.

72.1.12.4.8 TSQLResolver.GenInsertSQL Method

Pascal

```
procedure GenInsertSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams); virtual;
```

C++

```
virtual __fastcall GenInsertSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params);
```

Description

This is GenInsertSQL, a member of class TSQLResolver.

72.1.12.4.9 TSQLResolver.GenSelectSQL Method

Pascal

```
procedure GenSelectSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string;
Mode: TUpdateMode = upWhereKeyOnly); virtual;
```

C++

```
virtual __fastcall GenSelectSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params,
UnicodeString Alias, TUpdateMode Mode = upWhereKeyOnly);
```

Description

This is GenSelectSQL, a member of class TSQLResolver.

72.1.12.4.10 TSQLResolver.GenUpdateSQL Method

Pascal

```
procedure GenUpdateSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string);
virtual;
```

C++

```
virtual __fastcall GenUpdateSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params,
UnicodeString Alias);
```

Description

This is GenUpdateSQL, a member of class TSQLResolver.

72.1.12.4.11 TSQLResolver.GenWhereSQL Method

Pascal

```
procedure GenWhereSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; GenUpdateMode:
TUpdateMode; Alias: string); virtual;
```

C++

```
virtual __fastcall GenWhereSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params, TUpdateMode
GenUpdateMode, UnicodeString Alias);
```

Description

This is GenWhereSQL, a member of class TSQLResolver.

72.1.12.4.12 TSQLResolver.GetProvider Method

Pascal

```
function GetProvider: TDataSetProvider; virtual;
```

C++

```
virtual __fastcall TDataSetProvider GetProvider();
```

Description

This is GetProvider, a member of class TSQLResolver.

72.1.12.4.13 TSQLResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree);
```

Description

The HandleUpdateError method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with appropriate values for field's CurValue property.

Descendants of TCustomResolver implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 6497)

OnUpdateError (see page 6460)

72.1.12.4.14 TSQLResolver.InitTreeData Method

Pascal

```
procedure InitTreeData(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitTreeData(TUpdateTree Tree);
```

Description

This is InitTreeData, a member of class TSQLResolver.

72.1.12.4.15 TSQLResolver.InternalDoUpdate Method

Pascal

```
procedure InternalDoUpdate(Tree: TUpdateTree; UpdateKind: TUpdateKind); virtual;
```

C++

```
virtual __fastcall InternalDoUpdate(TUpdateTree Tree, TUpdateKind UpdateKind);
```

Description

This is InternalDoUpdate, a member of class TSQLResolver.

72.1.12.4.16 TSQLResolver.UseFieldInUpdate Method

Pascal

```
function UseFieldInUpdate(Field: TField): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseFieldInUpdate(TField Field);
```

Description

This is UseFieldInUpdate, a member of class TSQLResolver.

72.1.12.4.17 TSQLResolver.UseFieldInWhere Method

Pascal

```
function UseFieldInWhere(Field: TField; Mode: TUpdateMode): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseFieldInWhere(TField Field, TUpdateMode Mode);
```

Description

This is UseFieldInWhere, a member of class TSQLResolver.

72.1.12.5 TSQLResolver Properties

The properties of the TSQLResolver class are listed here.

Protected Properties

	Name	Description
	Provider (see page 6571)	Indicates the provider that uses this resolver to apply updates.

72.1.12.5.1 TSQLResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TDataSetProvider;
```

C++

```
__property TDataSetProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to generate events on the provider at appropriate points in the update process.

See Also

Create (see page 6493)

Resolver (see page 6456)

72.1.13 TUpdateTree Class

TUpdateTree stores information about a delta packet on behalf of a resolver.

Class Hierarchy

Pascal

```
TUpdateTree = class(TObject);
```

C++

```
class TUpdateTree : public TObject;
```

File

Provider

Description

When a client dataset applies its updates to the source dataset or database server, the provider that represents that dataset or server passes the updates to a resolver component. The resolver takes this delta data packet and builds a TUpdateTree to store the updates and information about them in a convenient form. In particular, the TUpdateTree structure allows a resolver to handle master/detail relationships (represented as nested dataset fields in the delta packet) so that updates are applied in the proper order.

Applications can't use TUpdateTree directly. Most of the structured information about the delta packet is represented in protected properties that are only available to the resolver classes in the Provider (see page 6441) unit.

See Also

TCustomResolver (see page 6491)

TCustomClientDataSet

TCustomProvider (see page 6464)

72.1.13.1 TUpdateTree Members

The following tables list the members exposed by TUpdateTree.

Protected Properties

	Name	Description
	DetailList (see page 6577)	This is DetailList, a member of class TUpdateTree.
	Opened (see page 6578)	This is Opened, a member of class TUpdateTree.

Public Methods

	Name	Description
	Clear (see page 6574)	This is Clear, a member of class TUpdateTree.
	Create (see page 6573)	Creates and initializes an instance of TUpdateTree (see page 6572).

	Destroy (see page 6573)	Destroys an instance of TUpdateTree (see page 6572).
	DoUpdates (see page 6574)	This is DoUpdates, a member of class TUpdateTree.
	InitData (see page 6574)	This is InitData, a member of class TUpdateTree.
	InitDelta (see page 6575)	This is the overview for the InitDelta method overload.
	InitErrorPacket (see page 6575)	This is InitErrorPacket, a member of class TUpdateTree.
	RefreshData (see page 6576)	This is RefreshData, a member of class TUpdateTree.

Public Properties

	Name	Description
	Data (see page 6576)	This is Data, a member of class TUpdateTree.
	Delta (see page 6576)	This is Delta, a member of class TUpdateTree.
R	DetailCount (see page 6577)	This is DetailCount, a member of class TUpdateTree.
R	Details (see page 6577)	This is Details, a member of class TUpdateTree.
	ErrorDS (see page 6577)	This is ErrorDS, a member of class TUpdateTree.
R	HasErrors (see page 6578)	This is HasErrors, a member of class TUpdateTree.
R	IsNested (see page 6578)	This is IsNested, a member of class TUpdateTree.
	Name (see page 6578)	This is Name, a member of class TUpdateTree.
R	Parent (see page 6578)	This is Parent, a member of class TUpdateTree.
R	Resolver (see page 6579)	This is Resolver, a member of class TUpdateTree.
	Source (see page 6579)	This is Source, a member of class TUpdateTree.

72.1.13.2 TUpdateTree.Create Constructor

Creates and initializes an instance of TUpdateTree (see page 6572).

Pascal

```
constructor Create(AParent: TUpdateTree; AResolver: TCustomResolver);
```

C++

```
__fastcall TUpdateTree(TUpdateTree AParent, TCustomResolver AResolver);
```

Description

Applications do not need to call the TUpdateTree (see page 6572) constructor. Update trees are created automatically by the resolver components that use them.

The AParent parameter specifies the TUpdateTree (see page 6572) for the updates to a master table if this update tree represents a detail data set. If the TUpdateTree (see page 6572) does not represent the updates to a detail dataset, AParent is nil (Delphi) or NULL (C++).

The AResolver parameter specifies the resolver component that uses this update tree when applying updates. This parameter must be specified because the update tree calls some of this resolver's methods.

72.1.13.3 TUpdateTree.Destroy Destructor

Destroys an instance of TUpdateTree (see page 6572).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TUpdateTree();
```

Description

Applications do not need to free TUpdateTree (see page 6572) instances. Update trees are freed by the resolver components that use them.

See Also

System::TObject::Free

72.1.13.4 TUpdateTree Methods

The methods of the TUpdateTree class are listed here.

Public Methods

	Name	Description
	Clear (see page 6574)	This is Clear, a member of class TUpdateTree.
	DoUpdates (see page 6574)	This is DoUpdates, a member of class TUpdateTree.
	InitData (see page 6574)	This is InitData, a member of class TUpdateTree.
	InitDelta (see page 6575)	This is the overview for the InitDelta method overload.
	InitErrorPacket (see page 6575)	This is InitErrorPacket, a member of class TUpdateTree.
	RefreshData (see page 6576)	This is RefreshData, a member of class TUpdateTree.

72.1.13.4.1 TUpdateTree.Clear Method

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

This is Clear, a member of class TUpdateTree.

72.1.13.4.2 TUpdateTree.DoUpdates Method

Pascal

```
function DoUpdates: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoUpdates();
```

Description

This is DoUpdates, a member of class TUpdateTree.

72.1.13.4.3 TUpdateTree.InitData Method

Pascal

```
procedure InitData(ASource: TDataSet); virtual;
```

C++

```
virtual __fastcall InitData(TDataSet ASource);
```

Description

This is InitData, a member of class TUpdateTree.

72.1.13.4.4 InitDelta Method

This is the overview for the InitDelta method overload.

Overload List

	Name	Description
	TUpdateTree.InitDelta (OleVariant) (see page 6575)	This is InitDelta, a member of class TUpdateTree.
	TUpdateTree.InitDelta (TPacketDataSet) (see page 6575)	This is InitDelta, a member of class TUpdateTree.

72.1.13.4.4.1 TUpdateTree.InitDelta Method (OleVariant)

Pascal

```
procedure InitDelta(const ADelta: OleVariant); virtual; overload;
```

C++

```
virtual __fastcall InitDelta(const OleVariant ADelta);
```

Description

This is InitDelta, a member of class TUpdateTree.

72.1.13.4.4.2 TUpdateTree.InitDelta Method (TPacketDataSet)

Pascal

```
procedure InitDelta(ADelta: TPacketDataSet); virtual; overload;
```

C++

```
virtual __fastcall InitDelta(TPacketDataSet ADelta);
```

Description

This is InitDelta, a member of class TUpdateTree.

72.1.13.4.5 TUpdateTree.InitErrorPacket Method

Pascal

```
procedure InitErrorPacket(E: EUpdateError; Response: TResolverResponse); virtual;
```

C++

```
virtual __fastcall InitErrorPacket(EUpdateError E, TResolverResponse Response);
```

Description

This is InitErrorPacket, a member of class TUpdateTree.

72.1.13.4.6 TUpdateTree.RefreshData Method

Pascal

```
procedure RefreshData(Options: TFetchOptions); virtual;
```

C++

```
virtual __fastcall RefreshData(TFetchOptions Options);
```

Description

This is RefreshData, a member of class TUpdateTree.

72.1.13.5 TUpdateTree Properties

The properties of the TUpdateTree class are listed here.

Protected Properties

	Name	Description
R	DetailList (see page 6577)	This is DetailList, a member of class TUpdateTree.
	Opened (see page 6578)	This is Opened, a member of class TUpdateTree.

Public Properties

	Name	Description
	Data (see page 6576)	This is Data, a member of class TUpdateTree.
	Delta (see page 6576)	This is Delta, a member of class TUpdateTree.
R	DetailCount (see page 6577)	This is DetailCount, a member of class TUpdateTree.
R	Details (see page 6577)	This is Details, a member of class TUpdateTree.
	ErrorDS (see page 6577)	This is ErrorDS, a member of class TUpdateTree.
R	HasErrors (see page 6578)	This is HasErrors, a member of class TUpdateTree.
R	IsNested (see page 6578)	This is IsNested, a member of class TUpdateTree.
	Name (see page 6578)	This is Name, a member of class TUpdateTree.
R	Parent (see page 6578)	This is Parent, a member of class TUpdateTree.
R	Resolver (see page 6579)	This is Resolver, a member of class TUpdateTree.
	Source (see page 6579)	This is Source, a member of class TUpdateTree.

72.1.13.5.1 TUpdateTree.Data Property

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

This is Data, a member of class TUpdateTree.

72.1.13.5.2 TUpdateTree.Delta Property

Pascal

```
property Delta: TPacketDataSet;
```

C++

```
__property TPacketDataSet Delta;
```

Description

This is Delta, a member of class TUpdateTree.

72.1.13.5.3 TUpdateTree.DetailCount Property

Pascal

```
property DetailCount: Integer;
```

C++

```
__property int DetailCount;
```

Description

This is DetailCount, a member of class TUpdateTree.

72.1.13.5.4 TUpdateTree.DetailList Property

Pascal

```
property DetailList: TList;
```

C++

```
__property TList DetailList;
```

Description

This is DetailList, a member of class TUpdateTree.

72.1.13.5.5 TUpdateTree.Details Property

Pascal

```
property Details [Index: Integer]: TUpdateTree;
```

C++

```
__property TUpdateTree Details[int Index];
```

Description

This is Details, a member of class TUpdateTree.

72.1.13.5.6 TUpdateTree.ErrorDS Property

Pascal

```
property ErrorDS: TPacketDataSet;
```

C++

```
__property TPacketDataSet ErrorDS;
```

Description

This is ErrorDS, a member of class TUpdateTree.

72.1.13.5.7 TUpdateTree.HasErrors Property

Pascal

```
property HasErrors: Boolean;
```

C++

```
__property Boolean HasErrors;
```

Description

This is HasErrors, a member of class TUpdateTree.

72.1.13.5.8 TUpdateTree.IsNested Property

Pascal

```
property IsNested: Boolean;
```

C++

```
__property Boolean IsNested;
```

Description

This is IsNested, a member of class TUpdateTree.

72.1.13.5.9 TUpdateTree.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class TUpdateTree.

72.1.13.5.10 TUpdateTree.Opened Property

Pascal

```
property Opened: Boolean;
```

C++

```
__property Boolean Opened;
```

Description

This is Opened, a member of class TUpdateTree.

72.1.13.5.11 TUpdateTree.Parent Property

Pascal

```
property Parent: TUpdateTree;
```

C++

```
__property TUpdateTree Parent;
```

Description

This is Parent, a member of class TUpdateTree.

72.1.13.5.12 TUpdateTree.Resolver Property

Pascal

```
property Resolver: TCustomResolver;
```

C++

```
__property TCustomResolver Resolver;
```

Description

This is Resolver, a member of class TUpdateTree.

72.1.13.5.13 TUpdateTree.Source Property

Pascal

```
property Source: TDataSet;
```

C++

```
__property TDataSet Source;
```

Description

This is Source, a member of class TUpdateTree.

72.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IProviderContainer (see page 6580)	This is class Provider.IProviderContainer.

72.2.1 IProviderContainer Interface

Class Hierarchy

Provider.IProviderContainer

Pascal

```
IProviderContainer = interface;
```

C++

```
__interface IProviderContainer;
```

File

Provider

Description

This is class Provider.IProviderContainer.

72.2.1.1 IProviderContainer Members

The following tables list the members exposed by IProviderContainer.

Public Methods

	Name	Description
	RegisterProvider (see page 6581)	This is RegisterProvider, a member of class IProviderContainer.
	UnRegisterProvider (see page 6581)	This is UnRegisterProvider, a member of class IProviderContainer.

72.2.1.2 IProviderContainer Methods

The methods of the IProviderContainer class are listed here.

Public Methods

	Name	Description
	RegisterProvider (see page 6581)	This is RegisterProvider, a member of class IProviderContainer.
	UnRegisterProvider (see page 6581)	This is UnRegisterProvider, a member of class IProviderContainer.

72.2.1.2.1 IProviderContainer.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Prov);
```

Description

This is RegisterProvider, a member of class IProviderContainer.

72.2.1.2.2 IProviderContainer.UnRegisterProvider Method

Pascal

```
procedure UnRegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Prov);
```

Description

This is UnRegisterProvider, a member of class IProviderContainer.

72.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetObjectProperty (see page 6582)	This is function Provider.GetObjectProperty.
	GetStringProperty (see page 6582)	This is function Provider.GetStringProperty.
	VarArrayFromStrings (see page 6582)	Creates a Variant array that contains the strings in a TStrings object.
	VarArrayFromWideStrings (see page 6583)	This is function Provider.VarArrayFromWideStrings.

72.3.1 Provider.GetObjectProperty Function

Pascal

```
function GetObjectProperty(Instance: TPersistent; const PropName: string): TObject;
```

C++

```
TObject * GetObjectProperty(TPersistent * Instance, const UnicodeString PropName);
```

File

Provider

Description

This is function Provider.GetObjectProperty.

72.3.2 Provider.GetStringProperty Function

Pascal

```
function GetStringProperty(Instance: TPersistent; const PropName: string): WideString;
```

C++

```
BSTR GetStringProperty(TPersistent * Instance, const UnicodeString PropName);
```

File

Provider

Description

This is function Provider.GetStringProperty.

72.3.3 Provider.VarArrayFromStrings Function

Creates a Variant array that contains the strings in a TStrings object.

Pascal

```
function VarArrayFromStrings(Strings: TStrings): Variant;
```

C++

```
Variant VarArrayFromStrings(TStrings Strings);
```

File

Provider

Description

Call VarArrayFromStrings to create a Variant array to represent the strings in a TStrings object. VarArrayFromStrings creates a new Variant array of WideString values, where each WideString corresponds to a string in the specified TStrings object.

Strings is the TStrings object that supplies the strings for the Variant array.

VarArrayFromStrings returns the new Variant array.

See Also

PackageParams

VarArrayCreate

72.3.4 Provider.VarArrayFromWideStrings Function

Pascal

```
function VarArrayFromWideStrings(Strings: TWideStrings): Variant;
```

C++

```
Variant VarArrayFromWideStrings(TWideStrings Strings);
```

File

Provider

Description

This is function Provider.VarArrayFromWideStrings.

72.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TGetRecordOption (see page 6584)	TGetRecordOption and TGetRecordOptions (see page 6591) influence what type of information is added to data packets.
	TProviderOption (see page 6585)	TProviderOption and TProviderOptions (see page 6593) influence what type of information is communicated between client datasets and provider components.
	TResolverResponse (see page 6587)	TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Records

	Name	Description
	TPutFieldInfo (see page 6587)	This is record Provider.TPutFieldInfo.

72.4.1 Provider.TGetRecordOption Enumeration

TGetRecordOption and TGetRecordOptions (see page 6591) influence what type of information is added to data packets.

Pascal

```
TGetRecordOption = (
  grMetaData,
  grReset,
  grXML,
  grXMLUTF8
);
```

C++

```
enum TGetRecordOption {
  grMetaData,
  grReset,
  grXML,
  grXMLUTF8
};
```

File

Provider

Description

TGetRecordOptions (see page 6591) is a set of TGetRecordOption values. TGetRecordOption values are listed in the following table:

Value	Meaning
grMetaData	The data packet includes metadata as well as record values.
grReset	The records in the data packet start with the first record, regardless of the contents of any previously sent data packets.
grXML	The data packet is transmitted as a string of XML instead of as an OleVariant. Extended characters are expressed using escape sequences.

grXMLUTF8	The data packet is transmitted as a string of XML instead of as an OleVariant. Extended characters are encoded using UTF-8.

See Also

TCustomProvider (see page 6464)

72.4.2 Provider.TProviderOption Enumeration

TProviderOption and TProviderOptions (see page 6593) influence what type of information is communicated between client datasets and provider components.

Pascal

```
TProviderOption = (
  poFetchBlobsOnDemand,
  poFetchDetailsOnDemand,
  poIncFieldProps,
  poCascadeDeletes,
  poCascadeUpdates,
  poReadOnly,
  poAllowMultiRecordUpdates,
  poDisableInserts,
  poDisableEdits,
  poDisableDeletes,
  poNoReset,
  poAutoRefresh,
  poPropogateChanges,
  poAllowCommandText,
  poRetainServerOrder,
  poUseQuoteChar
);
```

C++

```
enum TProviderOption {
  poFetchBlobsOnDemand,
  poFetchDetailsOnDemand,
  poIncFieldProps,
  poCascadeDeletes,
  poCascadeUpdates,
  poReadOnly,
  poAllowMultiRecordUpdates,
  poDisableInserts,
  poDisableEdits,
  poDisableDeletes,
  poNoReset,
  poAutoRefresh,
  poPropogateChanges,
  poAllowCommandText,
  poRetainServerOrder,
  poUseQuoteChar
};
```

File

Provider

Description

TProviderOptions (see page 6593) is a set of TProviderOption values. TProviderOption values are listed in the following table:

Value	Meaning
poFetchBlobsOnDemand	BLOB fields are not included in data packets. Instead, client applications must request these values on an as-needed basis. If the client dataset's FetchOnDemand property is true, the client requests these values automatically. Otherwise, the client application uses the FetchBlobs method to retrieve BLOB data.
poFetchDetailsOnDemand	When the provider represents the master of a master/detail relationship, nested detail values are not included in the data packet. Instead, client applications request these on an as-needed basis. If the client dataset's FetchOnDemand property is true, the client requests these values automatically. Otherwise the client application uses the FetchDetails method to retrieve nested detail sets.
poIncFieldProps	The data packet includes the following field properties (where applicable): Alignment, DisplayLabel, DisplayWidth, Visible, DisplayFormat, EditFormat, MaxValue, MinValue, Currency, EditMask, DisplayValues
poCascadeDeletes	Tells the server to delete detail records automatically when master table records are deleted. To use this option, the provider must represent the master of a master/detail relationship and the database must support cascaded deletes as part of its referential integrity settings.
poCascadeUpdates	Tells the server to update detail records automatically when master table key values are changed. To use this option, the provider must represent the master of a master/detail relationship and the database must support cascaded updates as part of its referential integrity settings.
poReadOnly	The client dataset can't apply updates to the provider.
poAllowMultiRecordUpdates	Enables individual updates that affect multiple records. When poAllowMultiRecordUpdates is not included, updates that change multiple records are automatically aborted.
poDisableInserts	Clients can't add new records. If the client tries to insert a record an exception is raised. (This does not affect the client's ability to edit or delete records)
poDisableEdits	Clients can't modify existing data values. If the client tries to edit a field an exception is raised. (This does not affect the client's ability to insert or delete records)
poDisableDeletes	Clients can't delete records. If the client tries to delete a record an exception is raised. (This does not affect the client's ability to edit or insert records)
poNoReset	Ignores the reset flag in calls to AS_GetRecords (TGetRecordsOption value of grReset).
poAutoRefresh	Refreshes the client dataset with current record values whenever it applies updates.
poPropogateChanges	Changes made in a BeforeUpdateRecord or AfterUpdateRecord event handler are sent back to the client and merged into the client dataset.
poAllowCommandText	Enables the client to override the provider's dataset by replacing its associated SQL text or the name of its stored procedure or table. These overrides are supplied to the GetRecords or Execute method.
poRetainServerOrder	Signals that the client should not attempt to re-establish the default order by sorting the records in the data packet. This prevents the client from changing the record order because, for example, it is unable to duplicate the locale on the server.
poUseQuoteChar	Specifies that the quote character (“) used by the SQL provider will also be used to quote SQL identifiers in the SQL queries created by the provider. This option is True by default; set it False to eliminate the quotes.

See Also

TBaseProvider (see page 6445)

TGetRecordOption (see page 6584)

72.4.3 Provider.TPutFieldInfo Record

Pascal

```
TPutFieldInfo = record
  FieldNo: Integer;
  Field: TField;
  DataSet: TDataSet;
  Size: Integer;
  IsDetail: Boolean;
  Opened: Boolean;
  PutProc: TPutFieldProc;
  LocalFieldIndex: Integer;
  FieldInfos: Pointer;
end;
```

C++

```
struct TPutFieldInfo {
  int FieldNo;
  TField Field;
  TDataSet DataSet;
  int Size;
  Boolean IsDetail;
  Boolean Opened;
  TPutFieldProc PutProc;
  int LocalFieldIndex;
  void * FieldInfos;
};
```

File

Provider

Description

This is record Provider.TPutFieldInfo.

72.4.4 Provider.TResolverResponse Enumeration

TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Pascal

```
TResolverResponse = (
  rrSkip,
  rrAbort,
  rrMerge,
  rrApply,
  rrIgnore
);
```

C++

```
enum TResolverResponse {
  rrSkip,
  rrAbort,
  rrMerge,
  rrApply,
```

```
  rrIgnore
};
```

File

Provider

Description

The following table lists the possible values for TResolverResponse and what they indicate:

Value	Meaning
rrSkip	Skip updating the record that raised the error condition, and leave the unapplied changes in the cache.
rrAbort	Abort the entire update operation without displaying an error message.
rrMerge	Merge the updates in the delta packet with the new underlying value from the database. This will only succeed if none of the updated fields have been changed by another user.
rrApply	Apply the current record value (as it has been fixed in the OnUpdateError handler) instead of the original update.
rrIgnore	Ignore the error condition: Do not apply the record but do not return the record to the client dataset for resolving either.

See Also

OnUpdateError (see page 6460)

TCustomResolver (see page 6491)

72.5 Types

The following table lists types in this documentation.

Types

Name	Description
PPutFieldInfo (see page 6589)	This is type Provider.PPutFieldInfo.
TAfterUpdateRecordEvent (see page 6589)	This is type Provider.TAfterUpdateRecordEvent.
TBeforeCommitEvent (see page 6590)	This is type Provider.TBeforeCommitEvent.
TBeforeUpdateRecordEvent (see page 6590)	This is type Provider.TBeforeUpdateRecordEvent.
TDataRequestEvent (see page 6590)	This is type Provider.TDataRequestEvent.
TGetDSProps (see page 6591)	This is type Provider.TGetDSProps.
TGetParamsEvent (see page 6591)	This is type Provider.TGetParamsEvent.
TGetRecordOptions (see page 6591)	This is type Provider.TGetRecordOptions.
TGetTableNameEvent (see page 6592)	This is type Provider.TGetTableNameEvent.
TInfoArray (see page 6592)	This is type Provider.TInfoArray.
TOnValidate (see page 6592)	This is type Provider.TOnValidate.
TProviderDataEvent (see page 6593)	This is type Provider.TProviderDataEvent.
TProviderOptions (see page 6593)	This is type Provider.TProviderOptions.
TPutFieldProc (see page 6593)	This is type Provider.TPutFieldProc.
TResolverErrorEvent (see page 6594)	This is type Provider.TResolverErrorEvent.

72.5.1 Provider.PPutFieldInfo Type

Pascal

```
PPutFieldInfo = ^TPutFieldInfo;
```

C++

```
^TPutFieldInfo PPutFieldInfo;
```

File

Provider

Description

This is type Provider.PPutFieldInfo.

72.5.2 Provider.TAfterUpdateRecordEvent Type

Pascal

```
TAfterUpdateRecordEvent = procedure (Sender: TObject; SourceDS: TDataSet; DeltaDS:
TCustomClientDataSet; UpdateKind: TUpdateKind) of object;
```

C++

```
(Sender: TObject; SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind)
( TAfterUpdateRecordEvent)();
```

File

Provider

Description

This is type Provider.TAfterUpdateRecordEvent.

72.5.3 Provider.TBeforeCommitEvent Type

Pascal

```
TBeforeCommitEvent = procedure (Sender:TObject; DeltaDS: TCustomClientDataSet; ErrorCount,
MaxErrors: integer; const ResultVar: OleVariant) of object;
```

C++

```
(Sender:TObject; DeltaDS: TCustomClientDataSet; ErrorCount, MaxErrors: integer; const
ResultVar: OleVariant) ( TBeforeCommitEvent)();
```

File

Provider

Description

This is type Provider.TBeforeCommitEvent.

72.5.4 Provider.TBeforeUpdateRecordEvent Type

Pascal

```
TBeforeUpdateRecordEvent = procedure (Sender: TObject; SourceDS: TDataSet; DeltaDS:
TCustomClientDataSet; UpdateKind: TUpdateKind; var Applied: Boolean) of object;
```

C++

```
(Sender: TObject; SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind;
var Applied: Boolean) ( TBeforeUpdateRecordEvent)();
```

File

Provider

Description

This is type Provider.TBeforeUpdateRecordEvent.

72.5.5 Provider.TDataRequestEvent Type

Pascal

```
TDataRequestEvent = function (Sender: TObject; Input: OleVariant): OleVariant of object;
```

C++

```
(Sender: TObject; Input: OleVariant): OleVariant ( TDataRequestEvent)();
```

File

Provider

Description

This is type Provider.TDataRequestEvent.

72.5.6 Provider.TGetDSProps Type

Pascal

```
TGetDSProps = procedure (Sender: TObject; DataSet: TDataSet; out Properties: OleVariant) of object;
```

C++

```
(Sender: TObject; DataSet: TDataSet; out Properties: OleVariant) ( TGetDSProps)();
```

File

Provider

Description

This is type Provider.TGetDSProps.

72.5.7 Provider.TGetParamsEvent Type

Pascal

```
TGetParamsEvent = procedure (DataSet: TDataSet; Params: TList) of object;
```

C++

```
(DataSet: TDataSet; Params: TList) ( TGetParamsEvent)();
```

File

Provider

Description

This is type Provider.TGetParamsEvent.

72.5.8 Provider.TGetRecordOptions Type

Pascal

```
TGetRecordOptions = set of TGetRecordOption;
```

C++

```
TGetRecordOption TGetRecordOptions;
```

File

Provider

Description

This is type Provider.TGetRecordOptions.

72.5.9 Provider.TGetTableNameEvent Type

Pascal

```
TGetTableNameEvent = procedure (Sender: TObject; DataSet: TDataSet; var TableName: WideString)
of object;
```

C++

```
(Sender: TObject; DataSet: TDataSet; var TableName: WideString) ( TGetTableNameEvent)();
```

File

Provider

Description

This is type Provider.TGetTableNameEvent.

72.5.10 Provider.TInfoArray Type

Pascal

```
TInfoArray = array of TPutFieldInfo;
```

C++

```
array of TPutFieldInfo TInfoArray;
```

File

Provider

Description

This is type Provider.TInfoArray.

72.5.11 Provider.TOnValidate Type

Pascal

```
TOnValidate = procedure (const Delta: OleVariant) of object;
```

C++

```
(const Delta: OleVariant) ( TOnValidate)();
```

File

Provider

Description

This is type Provider.TOnValidate.

72.5.12 Provider.TProviderDataEvent Type

Pascal

```
TProviderDataEvent = procedure (Sender: TObject; DataSet: TCustomClientDataSet) of object;
```

C++

```
(Sender: TObject; DataSet: TCustomClientDataSet) ( TProviderDataEvent)();
```

File

Provider

Description

This is type Provider.TProviderDataEvent.

72.5.13 Provider.TProviderOptions Type

Pascal

```
TProviderOptions = set of TProviderOption;
```

C++

```
TProviderOption TProviderOptions;
```

File

Provider

Description

This is type Provider.TProviderOptions.

72.5.14 Provider.TPutFieldProc Type

Pascal

```
TPutFieldProc = procedure (Info: PPutFieldInfo) of object;
```

C++

```
(Info: PPutFieldInfo) ( TPutFieldProc)();
```

File

Provider

Description

This is type Provider.TPutFieldProc.

72.5.15 Provider.TResolverErrorEvent Type

Pascal

```
TResolverErrorEvent = procedure (Sender: TObject; DataSet: TCustomClientDataSet; E:
EUpdateError; UpdateKind: TUpdateKind; var Response: TResolverResponse) of object;
```

C++

```
(Sender: TObject; DataSet: TCustomClientDataSet; E: EUpdateError; UpdateKind: TUpdateKind; var
Response: TResolverResponse) ( TResolverErrorEvent)();
```

File

Provider

Description

This is type Provider.TResolverErrorEvent.

72.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
InformixLob (see page 6595)	This is variable Provider.InformixLob.

72.6.1 Provider.InformixLob Variable

Pascal

```
InformixLob: Boolean;
```

C++

```
Boolean InformixLob;
```

File

Provider

Description

This is variable Provider.InformixLob.

72.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MetaDataOption (see page 6596)	Specifies the options available when retrieving metadata from a target database.
ResetOption (see page 6597)	This is constant Provider.ResetOption.
XMLOption (see page 6597)	This is constant Provider.XMLOption.
XMLUTF8Option (see page 6598)	This is constant Provider.XMLUTF8Option.

72.7.1 Provider.MetaDataOption Constant

Specifies the options available when retrieving metadata from a target database.

Pascal

```
MetaDataOption: Integer = 1 shl ord(grMetaData);
```

C++

```
int MetaDataOption = 1 shl ord(grMetaData);
```

File

Provider

Description

Each database provides a number of options for data that can be retrieved from and about the database. When you build your own data provider, it can be helpful to use these options when implementing the ISQLMetaData interface. The following table describes the allowable metadata options.

Element	Description
CatalogName	Specifies a system table name.
DatabaseName	Specifies the name of the database.
DatabaseVersion	Specifies the version number of the database.
MaxColumnInSelect	Limits the number of columns that can be specified in a SELECT statement. The column count starts with the first column. Any columns beyond the limit are not returned.
MaxColumnsInTable	Limits the number of columns that can be returned from a table. The column count starts with the first column. Any columns beyond the limit are not returned.
MaxObjectNameLength	Limits the number of characters returned of a database object name. Names exceeding this limit are truncated. If database object names are similar, truncating them might result in duplicate names or meaningless names.
MaxRowSize	Limits the size of any returned rows. This can be useful if you suspect there might be BLOBs in the rows you are selecting, but you want to protect the performance of your query.
MaxSQLLength	Limits the length, in characters, of a SQL statement.
ObjectQuoteChar	Specifies the quote character to use when delimiting object names containing special characters, such as spaces.

PackageName	Specifies the name of a package.
ProcSupportsCursor	Specifies whether or not a stored procedure supports the use of a cursor. Set to **true** if it supports a cursor, otherwise set to **false**.
ProcSupportsCursors	Specifies whether or not a stored procedure supports the use of multiple cursors. Set to **true** if it supports cursors, otherwise set to **false**.
SQLEscapeChar	Specifies the character to use to delimit escape sequences in SQL statements.
SchemaName	Specifies the name of a database schema if different from the DatabaseName.
SupportsTransaction	Specifies whether or not the database can be named in a transaction-based SQL statement. Set to **true** if it supports a transaction, otherwise set to **false**.
SupportsTransactions	Specifies whether or not the database can be named in multiple transaction-based SQL statements. Set to **true** if it supports transactions, otherwise set to **false**.
TransactionIsoLevel	Specifies the transaction isolation level. Isolation levels supported by BDP.NET include ReadCommitted, RepeatableRead, DirtyRead, and Custom.

72.7.2 Provider.ResetOption Constant

Pascal

```
ResetOption: Integer = 1 shl ord(grReset);
```

C++

```
int ResetOption = 1 shl ord(grReset);
```

File

Provider

Description

This is constant Provider.ResetOption.

72.7.3 Provider.XMLOption Constant

Pascal

```
XMLOption: Integer = 1 shl ord(grXML);
```

C++

```
int XMLOption = 1 shl ord(grXML);
```

File

Provider

Description

This is constant Provider.XMLOption.

72.7.4 Provider.XMLUTF8Option Constant

Pascal

```
XMLUTF8Option: Integer = 1 shl ord(grXMLUTF8);
```

C++

```
int XMLUTF8Option = 1 shl ord(grXMLUTF8);
```

File

Provider

Description

This is constant Provider.XMLUTF8Option.

73 Registry Namespace

This is namespace Registry.

Topics

Name	Description
Classes (see page 6600)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6691)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	ERegistryException (see page 6600)	ERegistryException is the exception class for registry errors.
	TRegIniFile (see page 6601)	TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.
	TRegistry (see page 6610)	TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.
	TRegistryIniFile (see page 6663)	TRegistryIniFile is a wrapper for the Windows system registry.

Structs, Records, Enums

	Name	Description
	TRegDataInfo (see page 6691)	TRegDataInfo is a type used by the GetDataInfo method of TRegistery.
	TRegDataType (see page 6691)	TRegDataType is a type used by the GetDataInfo method of TRegistery.
	TRegKeyInfo (see page 6692)	TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

73.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERegistryException (see page 6600)	ERegistryException is the exception class for registry errors.
	TRegIniFile (see page 6601)	TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.
	TRegistry (see page 6610)	TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.
	TRegistryIniFile (see page 6663)	TRegistryIniFile is a wrapper for the Windows system registry.

73.1.1 ERegistryException Class

ERegistryException is the exception class for registry errors.

Class Hierarchy

Pascal

```
ERegistryException = class(Exception);
```

C++

```
class ERegistryException : public Exception;
```

File

Registry

Description

ERegistryException is raised when an application cannot

Create a new key in the registry.

Read a data value associated with a key.

Read a binary data value associated with a key into a local buffer.

Write a data value to a key.

Key creation fails when an application attempts to create a subkey under a key for which it does not have KEY_CREATE_SUB_KEY security access.

Reading a data value associated with a key fails when an application does not have KEY_READ security access, or when the application attempts to read the wrong kind of data from a key (for example, attempting to read string data from an integer data value).

Reading a binary data value into a local buffer fails when an application does not allocate a buffer large enough for the data.

Writing a data value to a key fails because an application does not have KEY_WRITE security access for the current key.

See Also

TRegistry (see page 6610)

TRegIniFile (see page 6601)

TRegistry (see page 6610)

73.1.2 TRegIniFile Class

TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.

Class Hierarchy

Pascal

```
TRegIniFile = class(TRegistry);
```

C++

```
class TRegIniFile : public TRegistry;
```

File

Registry

Description

TRegIniFile presents a simple interface to the system registry, hiding the need to know about the underlying structure of the registry. TRegIniFile acts as a helper object to TRegistryIniFile (see page 6663), which descends from TCustomIniFile (see page 4258) (and is therefore more widely compatible with other INI file classes).

TRegIniFile enables handling the Windows 95/NT system registry as if it were a Windows 3.x INI file. Instead of processing an INI file, however, TRegIniFile reads from and writes to the system registry.

Although TRegIniFile descends from TRegistry (see page 6610), it adds the properties and methods used by a TIniFile (see page 4294) object. The FileName (see page 6610) passed to a TRegIniFile object becomes a subkey under the system registry's root key (HKEY_CURRENT_USER by default). What corresponds to a section in an INI file is treated as a key in the system registry, and what corresponds to data entries under a section in an INI file are treated as data values under a key in the system registry.

Note: TRegIniFile is intended to be used primarily as a helper object to TRegistryIniFile (see page 6663), which unifies the interface to INI files and the system registry. However, applications can use TRegIniFile directly.

See Also

TIniFile (see page 4294)

TMemIniFile (see page 4307)

TCustomIniFile (see page 4258)

TRegistryIniFile (see page 6663)

73.1.2.1 TRegIniFile Members

The following tables list the members exposed by TRegIniFile.

Protected Methods

	Name	Description
	ChangeKey (see page 6619)	This is ChangeKey, a member of class TRegistry.

	GetBaseKey (see page 6626)	This is GetBaseKey, a member of class TRegistry.
	GetData (see page 6626)	This is GetData, a member of class TRegistry.
	GetKey (see page 6628)	This is GetKey, a member of class TRegistry.
	PutData (see page 6637)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 6644)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
	CloseKey (see page 6619)	Writes the current key to the registry and closes the key.
	Create (see page 6603)	This is the overview for the Create constructor overload.
	Create (see page 6615)	This is the overview for the Create constructor overload.
	CreateKey (see page 6622)	Creates a new key in the registry.
	DeleteKey (see page 6604)	Erases a data value that is associated with a key.
	DeleteValue (see page 6625)	Erases a data value that is associated with the current key.
V	Destroy (see page 6617)	Closes the current key and destroys the TRegistry (see page 6610) object.
	EraseSection (see page 6605)	Deletes a key and its data values in the system registry.
	GetDataAsString (see page 6626)	This is GetDataAsString, a member of class TRegistry.
	GetDataInfo (see page 6627)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 6627)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 6628)	Returns the enumerated data type of a specified data value associated with the current key.
	GetKeyInfo (see page 6628)	Returns information about the current key.
	GetKeyNames (see page 6629)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 6629)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 6630)	Determines if the current key has subkeys.
	KeyExists (see page 6630)	Determines if a specified key exists.
	LoadKey (see page 6632)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.
	MoveKey (see page 6633)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
	OpenKey (see page 6634)	Opens a specified key.
	OpenKeyReadOnly (see page 6636)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 6637)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 6605)	Retrieves a Boolean value from a specified data value associated with a key.
	ReadCurrency (see page 6638)	Retrieves a currency value from a specified data value associated with the current key.
	ReadDate (see page 6638)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 6639)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 6639)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 6606)	Retrieves an integer value from a specified data value (Ident) associated with a key (Section).
	ReadSection (see page 6606)	Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.
	ReadSections (see page 6607)	Retrieves the names of all subkeys associated with the current key.
	ReadSectionValues (see page 6607)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 6608)	Retrieves a string value from a specified data value (Ident) associated with a key (Section).
	ReadTime (see page 6641)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 6641)	Establishes a connection to a registry on another computer.
	RenameValue (see page 6642)	Changes the name of a data value associated with the current key.

	ReplaceKey (see page 6642)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 6643)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
	SaveKey (see page 6643)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 6644)	Removes a hive from the registry.
	ValueExists (see page 6645)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 6645)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 6608)	Writes a Boolean value to a specified data value associated with a specified key.
	WriteCurrency (see page 6646)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 6646)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 6647)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 6650)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 6652)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 6609)	Writes a long integer value to a specified data value associated with a specified key.
	WriteString (see page 6609)	Writes a string value to a specified data value associated with a specified key.
	WriteTime (see page 6656)	Stores a time value in a specifically named data value associated with the current key.

Public Properties

	Name	Description
	Access (see page 6657)	Specifies the level of access to use when opening keys.
R	CurrentKey (see page 6660)	Specifies the registry key that is currently open.
R	CurrentPath (see page 6660)	Indicates the registry path associated with the current key.
R	FileName (see page 6610)	Specifies the name of the key opened or created in the system registry.
	LazyWrite (see page 6660)	Specifies how keys are written to the registry when calling the CloseKey (see page 6619) procedure.
	RootKey (see page 6661)	Specifies the root key for the TRegistry (see page 6610) object in an application.

73.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegIniFile.Create (string) (see page 6603)	Creates a TRegIniFile (see page 6601) object for an application.
	TRegIniFile.Create (string, LongWord) (see page 6603)	Creates a TRegIniFile (see page 6601) object for an application.

73.1.2.2.1 TRegIniFile.Create Constructor (string)

Creates a TRegIniFile (see page 6601) object for an application.

Pascal

```
constructor Create(const FileName: string); overload;
```

```
constructor Create(const FileName: string; AAccess: LongWord); overload;
```

C++

```
__fastcall TRegIniFile(const UnicodeString FileName);
__fastcall TRegIniFile(const UnicodeString FileName, LongWord AAccess);
```

Description

Use Create to construct a TRegIniFile (see page 6601) object for an application.

FileName (see page 6610) initializes the FileName (see page 6610) property.

AAccess initializes the Access (see page 6657) property, which determines the access rights used when opening keys. If no AAccess parameter is supplied, Access (see page 6657) is initialized to KEY_ALL_ACCESS.

See Also

FileName (see page 6610)

Create (see page 6615)

Access (see page 6657)

73.1.2.3 TRegIniFile Methods

The methods of the TRegIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 6604)	Erases a data value that is associated with a key.
	EraseSection (see page 6605)	Deletes a key and its data values in the system registry.
	ReadBool (see page 6605)	Retrieves a Boolean value from a specified data value associated with a key.
	ReadInteger (see page 6606)	Retrieves an integer value from a specified data value (Ident) associated with a key (Section).
	ReadSection (see page 6606)	Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.
	ReadSections (see page 6607)	Retrieves the names of all subkeys associated with the current key.
	ReadSectionValues (see page 6607)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 6608)	Retrieves a string value from a specified data value (Ident) associated with a key (Section).
	WriteBool (see page 6608)	Writes a Boolean value to a specified data value associated with a specified key.
	WriteInteger (see page 6609)	Writes a long integer value to a specified data value associated with a specified key.
	WriteString (see page 6609)	Writes a string value to a specified data value associated with a specified key.

73.1.2.3.1 TRegIniFile.DeleteKey Method

Erases a data value that is associated with a key.

Pascal

```
procedure DeleteKey(const Section: string; const Ident: string);
```

C++

```
__fastcall DeleteKey(const UnicodeString Section, const UnicodeString Ident);
```

Description

Call DeleteKey to erase a data value associated with a key. Section is string containing the name of the system registry key containing the value to delete. Ident is a string containing the name of the data value to delete.

Note: DeleteKey for TRegIniFile (see page 6601) obscures the DeleteKey method of TRegistry (see page 6610), from which TRegIniFile (see page 6601) descends. You can access TRegistry (see page 6610)'s DeleteKey method by casting the call to DeleteKey as type TRegistry (see page 6610), or by using the scope resolution operator.

Note: On Windows 95, DeleteKey will delete the key and all its subkeys. On Windows NT, the subkeys must be explicitly deleted by separate calls to DeleteKey.

See Also

DeleteKey (see page 6623)

EraseSection (see page 6605)

73.1.2.3.2 TRegIniFile.EraseSection Method

Deletes a key and its data values in the system registry.

Pascal

```
procedure EraseSection(const Section: string);
```

C++

```
__fastcall EraseSection(const UnicodeString Section);
```

Description

Call EraseSection to remove a key and its data values from the system registry. Section identifies the key to delete.

See Also

DeleteKey (see page 6604)

73.1.2.3.3 TRegIniFile.ReadBool Method

Retrieves a Boolean value from a specified data value associated with a key.

Pascal

```
function ReadBool(const Section: string; const Ident: string; Default: Boolean): Boolean;
```

C++

```
__fastcall Boolean ReadBool(const UnicodeString Section, const UnicodeString Ident, Boolean
Default);
```

Description

Call ReadBool to retrieve a Boolean value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is a Boolean value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadBool function of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the ReadBool function of TRegistry (see page 6610) instead of this function, cast the call to ReadBool as TRegistry (see page 6610).

See Also

ReadInteger (see page 6606)

ReadString (see page 6608)

ReadBool (see page 6638)

73.1.2.3.4 TRegIniFile.ReadInteger Method

Retrieves an integer value from a specified data value (Ident) associated with a key (Section).

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
```

C++

```
__fastcall Longint ReadInteger(const UnicodeString Section, const UnicodeString Ident, Longint
Default);
```

Description

Call ReadInteger to retrieve an integer value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is an integer value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadInteger function of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the ReadInteger function of TRegistry (see page 6610) instead of this function, cast the call to ReadInteger as TRegistry (see page 6610).

See Also

ReadBool (see page 6605)

ReadString (see page 6608)

ReadInteger (see page 6640)

73.1.2.3.5 TRegIniFile.ReadSection Method

Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings);
```

C++

```
__fastcall ReadSection(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all data values associated with a key into a TStrings object.

Section is a string identifying the key for which to retrieve data-value names.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object such as a string list, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSections (see page 6607)

ReadSectionValues (see page 6607)

TListBox

Items

73.1.2.3.6 TRegIniFile.ReadSections Method

Retrieves the names of all subkeys associated with the current key.

Pascal

```
procedure ReadSections(Strings: TStrings);
```

C++

```
__fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all subkeys associated with the current key.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

CurrentKey (see page 6660)

ReadSection (see page 6606)

ReadSectionValues (see page 6607)

TListBox

Items

73.1.2.3.7 TRegIniFile.ReadSectionValues Method

Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings);
```

C++

```
__fastcall ReadSectionValues(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read all data values associated with a specified key into a string object.

Section is the name of the key from which to retrieve data-values.

Strings specifies the string list object to hold the retrieved values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself. Use the Values property of the string object to access a specific string in the object.

See Also

ReadSection (see page 6606)

ReadSections (see page 6607)

TListBox

Items

Values

73.1.2.3.8 TRegIniFile.ReadString Method

Retrieves a string value from a specified data value (Ident) associated with a key (Section).

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string): string;
```

C++

```
__fastcall UnicodeString ReadString(const UnicodeString Section, const UnicodeString Ident,
const UnicodeString Default);
```

Description

Call ReadString to retrieve a string value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is a string value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadString function of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the ReadString function of TRegistry (see page 6610) instead of this function, cast the call to ReadString as TRegistry (see page 6610).

See Also

ReadBool (see page 6605)

ReadInteger (see page 6606)

ReadString (see page 6640)

73.1.2.3.9 TRegIniFile.WriteBool Method

Writes a Boolean value to a specified data value associated with a specified key.

Pascal

```
procedure WriteBool(const Section: string; const Ident: string; Value: Boolean);
```

C++

```
__fastcall WriteBool(const UnicodeString Section, const UnicodeString Ident, Boolean Value);
```

Description

Call WriteBool to store a Boolean value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the Boolean value to write into the data value.

Note: This function overrides the WriteBool procedure of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the WriteBool procedure of TRegistry (see page 6610) instead of this procedure, cast the call to WriteBool as TRegistry (see page 6610).

See Also

WriteInteger (see page 6609)

WriteString (see page 6609)

WriteBool (see page 6646)

73.1.2.3.10 TRegIniFile.WriteInteger Method

Writes a long integer value to a specified data value associated with a specified key.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint);
```

C++

```
__fastcall WriteInteger(const UnicodeString Section, const UnicodeString Ident, Longint Value);
```

Description

Call WriteInteger to store an integer value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the integer value to write into the data value.

Note: This function overrides the WriteInteger procedure of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the WriteInteger procedure of TRegistry (see page 6610) instead of this procedure, cast the call to WriteInteger as TRegistry (see page 6610).

See Also

WriteBool (see page 6608)

WriteString (see page 6609)

WriteInteger (see page 6653)

73.1.2.3.11 TRegIniFile.WriteString Method

Writes a string value to a specified data value associated with a specified key.

Pascal

```
procedure WriteString(const Section: string; const Ident: string; const Value: string);
```

C++

```
__fastcall WriteString(const UnicodeString Section, const UnicodeString Ident, const
UnicodeString Value);
```

Description

Call WriteString to store a string value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the string value to write into the data value.

Note: This function overrides the WriteString procedure of TRegistry (see page 6610), from which TRegIniFile (see page 6601) is derived. To call the WriteString procedure of TRegistry (see page 6610) instead of this procedure, cast the call to WriteString as TRegistry (see page 6610).

See Also

WriteBool (see page 6608)

WriteInteger (see page 6609)

WriteString (see page 6653)

73.1.2.4 TRegIniFile Properties

The properties of the TRegIniFile class are listed here.

Public Properties

	Name	Description
	FileName (see page 6610)	Specifies the name of the key opened or created in the system registry.

73.1.2.4.1 TRegIniFile.FileName Property

Specifies the name of the key opened or created in the system registry.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Query FileName to determine the name of the key in the system registry that is the root key for all TRegIniFile (see page 6601) operations.

See Also

CurrentKey (see page 6660)

RootKey (see page 6661)

73.1.3 TRegistry Class

TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.

Class Hierarchy

Pascal

```
TRegistry = class(TObject);
```

C++

```
class TRegistry : public TObject;
```

File

Registry

Description

Use TRegistry to encapsulate access to the Windows system registry in an application. The registry is a database that an application can use to store and retrieve configuration information. Configuration information is stored in a hierarchical tree. Each node in the tree is called a key. Every key can contain subkeys and data values that represent part of the configuration information for an application.

All keys that an application creates, opens, reads, or writes are subkeys of predefined root keys. By default, a TRegistry object is created with a root key of HKEY_CURRENT_USER.

Only one key is accessible at a time in a TRegistry object. To determine the key that is currently accessible, read the value of the CurrentKey (see page 6660) property. TRegistry methods enable an application to open, close, save, move, copy, and delete keys.

One or more data values containing actual configuration information can be stored in a key. TRegistry methods enable an application to query a key to see if it contains data, to read data in a key, and to write data to a key.

Note: The TRegistry component is not fully compatable with the Windows NT environment. Functions such as RestoreKey (see page 6643) and SaveKey (see page 6643) may not perform properly without a significant number of workarounds.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                           mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
      MessageDlg('Unable to create key! Exiting.',
                 mtError, mbOKCancel, 0);
      Exit();
    end;

  {checking if the values exist and inserting when neccesary}

  if not reg.KeyExists('Creation\ Date') then
    begin
      today := Now;
        reg.WriteDateTime('Creation\ Date', today);
    end;

  if not reg.KeyExists('Licenced\ To') then
    begin
        reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
    end;

  if not reg.KeyExists('App\ Location') then
    begin
        reg.WriteExpandString('App\ Location',
                              '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
    end;

  if not reg.KeyExists('Projects\ Location') then
```

```
    begin
        reg.WriteExpandString('Projects\ Location',
                              '%USERPROFILE%\\MyApplication\\Projects\\');
    end;

  reg.CloseKey();

end;

procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
var
  reg : TRegistry;
begin
  reg := TRegistry.Create(KEY_WRITE);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
  reg.DeleteKey('Software\\MyCompanyName');

  reg.CloseKey();
end;
```

73.1.3.1 TRegistry Members

The following tables list the members exposed by TRegistry.

Protected Methods

	Name	Description
	ChangeKey (see page 6619)	This is ChangeKey, a member of class TRegistry.
	GetBaseKey (see page 6626)	This is GetBaseKey, a member of class TRegistry.
	GetData (see page 6626)	This is GetData, a member of class TRegistry.
	GetKey (see page 6628)	This is GetKey, a member of class TRegistry.
	PutData (see page 6637)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 6644)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
	CloseKey (see page 6619)	Writes the current key to the registry and closes the key.
	Create (see page 6615)	This is the overview for the Create constructor overload.
	CreateKey (see page 6622)	Creates a new key in the registry.
	DeleteKey (see page 6623)	Removes a specified key and its associated data from the registry.
	DeleteValue (see page 6625)	Erases a data value that is associated with the current key.
	Destroy (see page 6617)	Closes the current key and destroys the TRegistry (see page 6610) object.
	GetDataAsString (see page 6626)	This is GetDataAsString, a member of class TRegistry.
	GetDataInfo (see page 6627)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 6627)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 6628)	Returns the enumerated data type of a specified data value associated with the current key.
	GetKeyInfo (see page 6628)	Returns information about the current key.
	GetKeyNames (see page 6629)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 6629)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 6630)	Determines if the current key has subkeys.
	KeyExists (see page 6630)	Determines if a specified key exists.
	LoadKey (see page 6632)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.

	MoveKey (see page 6633)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
	OpenKey (see page 6634)	Opens a specified key.
	OpenKeyReadOnly (see page 6636)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 6637)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 6638)	Retrieves a Boolean value from a specified data value associated with the current key.
	ReadCurrency (see page 6638)	Retrieves a currency value from a specified data value associated with the current key.
	ReadDate (see page 6638)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 6639)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 6639)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 6640)	Retrieves an integer value from a specified data value associated with the current key.
	ReadString (see page 6640)	Retrieves a string value from a specified data value associated with the current key.
	ReadTime (see page 6641)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 6641)	Establishes a connection to a registry on another computer.
	RenameValue (see page 6642)	Changes the name of a data value associated with the current key.
	ReplaceKey (see page 6642)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 6643)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
	SaveKey (see page 6643)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 6644)	Removes a hive from the registry.
	ValueExists (see page 6645)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 6645)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 6646)	Stores a Boolean value in a specifically named data value associated with the current key.
	WriteCurrency (see page 6646)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 6646)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 6647)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 6650)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 6652)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 6653)	Stores an integer value in a specifically named data value associated with the current key.
	WriteString (see page 6653)	Stores a string value in a specifically named data value associated with the current key.
	WriteTime (see page 6656)	Stores a time value in a specifically named data value associated with the current key.

Public Properties

	Name	Description
	Access (see page 6657)	Specifies the level of access to use when opening keys.
R	CurrentKey (see page 6660)	Specifies the registry key that is currently open.
R	CurrentPath (see page 6660)	Indicates the registry path associated with the current key.
	LazyWrite (see page 6660)	Specifies how keys are written to the registry when calling the CloseKey (see page 6619) procedure.
	RootKey (see page 6661)	Specifies the root key for the TRegistry (see page 6610) object in an application.

73.1.3.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegistry.Create () (see page 6615)	Creates a TRegistry (see page 6610) object for an application.
	TRegistry.Create (LongWord) (see page 6615)	Creates a TRegistry (see page 6610) object for an application.

73.1.3.2.1 TRegistry.Create Constructor ()

Creates a TRegistry (see page 6610) object for an application.

Pascal

```
constructor Create; overload;
constructor Create(AAccess: LongWord); overload;
```

C++

```
__fastcall TRegistry();
__fastcall TRegistry(LongWord AAccess);
```

Description

Create instantiates a TRegistry (see page 6610) object and sets the RootKey (see page 6661) property to HKEY_CURRENT_USER and the LazyWrite (see page 6660) property to true. If an AAccess parameter is supplied, this value is used to initialize the Access (see page 6657) property. Otherwise, Access (see page 6657) is initialized to KEY_ALL_ACCESS.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary
```

```
    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
```

```
        MessageDlg('Unable to create key! Exiting.',
                   mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

LazyWrite (see page 6660)

RootKey (see page 6661)

Destroy (see page 6617)

Access (see page 6657)

73.1.3.3 TRegistry.Destroy Destructor

Closes the current key and destroys the TRegistry (see page 6610) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistry();
```

Description

Destroy closes the current key and frees the resources allocated for a registry object when it is no longer needed.

See Also

CurrentKey (see page 6660)

Create (see page 6615)

System::TObject::Free

73.1.3.4 TRegistry Methods

The methods of the TRegistry class are listed here.

Protected Methods

	Name	Description
	ChangeKey (see page 6619)	This is ChangeKey, a member of class TRegistry.
	GetBaseKey (see page 6626)	This is GetBaseKey, a member of class TRegistry.
	GetData (see page 6626)	This is GetData, a member of class TRegistry.
	GetKey (see page 6628)	This is GetKey, a member of class TRegistry.
	PutData (see page 6637)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 6644)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
	CloseKey (see page 6619)	Writes the current key to the registry and closes the key.
	CreateKey (see page 6622)	Creates a new key in the registry.
	DeleteKey (see page 6623)	Removes a specified key and its associated data from the registry.
	DeleteValue (see page 6625)	Erases a data value that is associated with the current key.
	GetDataAsString (see page 6626)	This is GetDataAsString, a member of class TRegistry.
	GetDataInfo (see page 6627)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 6627)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 6628)	Returns the enumerated data type of a specified data value associated with the current key.
	GetKeyInfo (see page 6628)	Returns information about the current key.
	GetKeyNames (see page 6629)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 6629)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 6630)	Determines if the current key has subkeys.
	KeyExists (see page 6630)	Determines if a specified key exists.
	LoadKey (see page 6632)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.
	MoveKey (see page 6633)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
	OpenKey (see page 6634)	Opens a specified key.
	OpenKeyReadOnly (see page 6636)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 6637)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 6638)	Retrieves a Boolean value from a specified data value associated with the current key.

	ReadCurrency (see page 6638)	Retrieves a currency value from a specified data value associated with the current key.
	ReadDate (see page 6638)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 6639)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 6639)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 6640)	Retrieves an integer value from a specified data value associated with the current key.
	ReadString (see page 6640)	Retrieves a string value from a specified data value associated with the current key.
	ReadTime (see page 6641)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 6641)	Establishes a connection to a registry on another computer.
	RenameValue (see page 6642)	Changes the name of a data value associated with the current key.
	ReplaceKey (see page 6642)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 6643)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
	SaveKey (see page 6643)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 6644)	Removes a hive from the registry.
	ValueExists (see page 6645)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 6645)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 6646)	Stores a Boolean value in a specifically named data value associated with the current key.
	WriteCurrency (see page 6646)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 6646)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 6647)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 6650)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 6652)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 6653)	Stores an integer value in a specifically named data value associated with the current key.
	WriteString (see page 6653)	Stores a string value in a specifically named data value associated with the current key.
	WriteTime (see page 6656)	Stores a time value in a specifically named data value associated with the current key.

73.1.3.4.1 TRegistry.ChangeKey Method

Pascal

```
procedure ChangeKey(Value: HKey; const Path: string);
```

C++

```
__fastcall ChangeKey(HKey Value, const UnicodeString Path);
```

Description

This is ChangeKey, a member of class TRegistry.

73.1.3.4.2 TRegistry.CloseKey Method

Writes the current key to the registry and closes the key.

Pascal

```
procedure CloseKey;
```

C++

```
__fastcall CloseKey();
```

Description

Call CloseKey to write the current key to the registry and close the key. An application should not keep keys open any longer than necessary. Many TRegistry (see page 6610) methods routinely call CloseKey after reading from or writing to the registry. Calling CloseKey when there is no current key has no effect.

Note: When an application closes a key, the key is written to the registry. The LazyWrite (see page 6660) property determines how the write takes place.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
```

```
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                           mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
      MessageDlg('Unable to create key! Exiting.',
                  mtError, mbOKCancel, 0);
      Exit();
    end;

  {checking if the values exist and inserting when neccesary}

  if not reg.KeyExists('Creation\ Date') then
    begin
      today := Now;
        reg.WriteDateTime('Creation\ Date', today);
    end;

  if not reg.KeyExists('Licenced\ To') then
    begin
        reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
    end;

  if not reg.KeyExists('App\ Location') then
    begin
```

```
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CurrentKey (see page 6660)

LazyWrite (see page 6660)

73.1.3.4.3 TRegistry.CreateKey Method

Creates a new key in the registry.

Pascal

```
function CreateKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean CreateKey(const UnicodeString Key);
```

Description

Use CreateKey to add a new key to the registry. Key is the name of the key to create. Key can be an absolute or relative name. An absolute key begins with a backslash (\) and is a subkey of the root key. A relative key is a subkey of the current key.

CreateKey returns true if key creation is successful. On error, an exception is raised. Attempting to create a key that already exists has no effect.

CreateKey creates only non-volatile keys. If you need to create volatile keys on Windows NT, you must call the Windows API directly.

CreateKey creates keys with a security access of KEY_ALL_ACCESS. If you need to create keys with more restricted access, you must call the Windows API directly.

A key created by CreateKey has no values. To set key values, use the WriteCurrency (see page 6646), WriteBinaryData (see page 6645), WriteBool (see page 6646), WriteDate (see page 6646), WriteDateTime (see page 6647), WriteFloat (see page 6652), WriteInteger (see page 6653), WriteString (see page 6653), or WriteTime (see page 6656) procedures.

See Also

WriteCurrency (see page 6646)

WriteBinaryData (see page 6645)

WriteBool (see page 6646)

WriteDate (see page 6646)

WriteDateTime (see page 6647)

WriteFloat (see page 6652)

WriteInteger (see page 6653)

WriteString (see page 6653)

WriteTime (see page 6656)

Access (see page 6657)

73.1.3.4.4 TRegistry.DeleteKey Method

Removes a specified key and its associated data from the registry.

Pascal

```
function DeleteKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean DeleteKey(const UnicodeString Key);
```

Description

Call DeleteKey to remove a specified key and its associated data, if any, from the registry. Under Windows 95, if the key has subkeys, the subkeys and any associated data are also removed. Under Windows NT, subkeys must be explicitly deleted by separate calls to DeleteKey.

DeleteKey returns true if key deletion is successful. On error, DeleteKey returns false.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                    mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                    mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary
```

```
    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
      MessageDlg('Unable to create key! Exiting.',
```

```
                    mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CreateKey (see page 6622)

DeleteValue (see page 6625)

73.1.3.4.5 TRegistry.DeleteValue Method

Erases a data value that is associated with the current key.

Pascal

```
function DeleteValue(const Name: string): Boolean;
```

C++

```
__fastcall Boolean DeleteValue(const UnicodeString Name);
```

Description

Call DeleteValue to remove a specific data value associated with the current key. Name is string containing the name of the

value to delete. Keys can contain multiple data values, and every value associated with a key has a unique name.

DeleteValue returns true if the deletion is successful, false if the deletion fails.

See Also

DeleteKey (see page 6623)

WriteCurrency (see page 6646)

WriteBinaryData (see page 6645)

WriteBool (see page 6646)

WriteDate (see page 6646)

WriteDateTime (see page 6647)

WriteFloat (see page 6652)

WriteInteger (see page 6653)

WriteString (see page 6653)

WriteTime (see page 6656)

73.1.3.4.6 TRegistry.GetBaseKey Method

Pascal

```
function GetBaseKey(Relative: Boolean): HKey;
```

C++

```
__fastcall HKey GetBaseKey(Boolean Relative);
```

Description

This is GetBaseKey, a member of class TRegistry.

73.1.3.4.7 TRegistry.GetData Method

Pascal

```
function GetData(const Name: string; Buffer: Pointer; BufSize: Integer; var RegData:
TRegDataType): Integer;
```

C++

```
__fastcall int GetData(const UnicodeString Name, void * Buffer, int BufSize, TRegDataType
RegData);
```

Description

This is GetData, a member of class TRegistry.

73.1.3.4.8 TRegistry.GetDataAsString Method

Pascal

```
function GetDataAsString(const ValueName: string; PrefixType: Boolean = false): string;
```

C++

```
__fastcall UnicodeString GetDataAsString(const UnicodeString ValueName, Boolean PrefixType =
false);
```

Description

This is GetDataAsString, a member of class TRegistry.

73.1.3.4.9 TRegistry.GetDataInfo Method

Returns information about a specified data value associated with the current key.

Pascal

```
function GetDataInfo(const ValueName: string; var Value: TRegDataInfo): Boolean;
```

C++

```
__fastcall Boolean GetDataInfo(const UnicodeString ValueName, TRegDataInfo Value);
```

Description

Call GetDataInfo to get information about a data value associated with the current key. ValueName is a string containing the name of the data value to query. Value is a TRegDataInfo (see page 6691) value in which information about the data value is returned. If the data value is of type rdString or rdExpandString, the size of the data value includes an extra byte for the terminating null character.

GetDataInfo returns true if data value information is returned. If GetDataInfo fails, it returns false, and the Value record is filled with zeros.

See Also

CurrentKey (see page 6660)

GetDataSize (see page 6627)

GetDataType (see page 6628)

GetKeyInfo (see page 6628)

73.1.3.4.10 TRegistry.GetDataSize Method

Returns the size, in bytes, of a specified data value associated with the current key.

Pascal

```
function GetDataSize(const ValueName: string): Integer;
```

C++

```
__fastcall int GetDataSize(const UnicodeString ValueName);
```

Description

Call GetDataSize to determine the size, in bytes, of a data value associated with the current key. ValueName is a string containing the name of the data value to query.

On success, GetDataSize returns the size of the data value. On failure, GetDataSize returns -1.

Note: If the data value is a string, GetDataSize returns the size of the data value and one extra byte for the terminating null character.

See Also

CurrentKey (see page 6660)

GetDataInfo (see page 6627)

GetDataType (see page 6628)

GetKeyInfo (see page 6628)

73.1.3.4.11 TRegistry.GetDataType Method

Returns the enumerated data type of a specified data value associated with the current key.

Pascal

```
function GetDataType(const ValueName: string): TRegDataType;
```

C++

```
__fastcall TRegDataType GetDataType(const UnicodeString ValueName);
```

Description

Call GetDataType to determine the data type of a data value associated with the current key. ValueName is a string containing the name of the data value to query.

GetDataType returns a TRegDataType (see page 6691) value that indicates the type of value associated with the current key.

See Also

CurrentKey (see page 6660)

GetDataInfo (see page 6627)

GetDataSize (see page 6627)

GetKeyInfo (see page 6628)

73.1.3.4.12 TRegistry.GetKey Method

Pascal

```
function GetKey(const Key: string): HKEY;
```

C++

```
__fastcall HKEY GetKey(const UnicodeString Key);
```

Description

This is GetKey, a member of class TRegistry.

73.1.3.4.13 TRegistry.GetKeyInfo Method

Returns information about the current key.

Pascal

```
function GetKeyInfo(var Value: TRegKeyInfo): Boolean;
```

C++

```
__fastcall Boolean GetKeyInfo(TRegKeyInfo Value);
```

Description

Call GetKeyInfo to return all information about the current key to an application. Value is a variable declared in an application to store the key information returned by this function.

On success GetKeyInfo returns true. On failure it returns false, and the Value record is set to zeros.

See Also

CurrentKey (see page 6660)

GetKeyNames (see page 6629)

GetValueNames (see page 6629)

HasSubKeys (see page 6630)

73.1.3.4.14 TRegistry.GetKeyNames Method

Returns a string list containing the names of all subkeys belonging to the current key.

Pascal

```
procedure GetKeyNames(Strings: TStrings);
```

C++

```
__fastcall GetKeyNames(TStrings Strings);
```

Description

Call GetKeyNames to determine the names of all subkeys belonging to the current key. Determining the names of subkeys is useful in an application that iterates through a set of keys.

Strings is a variable of type TStrings into which to return the list of subkey names.

See Also

CurrentKey (see page 6660)

GetKeyInfo (see page 6628)

GetValueNames (see page 6629)

73.1.3.4.15 TRegistry.GetValueNames Method

Returns a string list containing the names of all data values associated with the current key.

Pascal

```
procedure GetValueNames(Strings: TStrings);
```

C++

```
__fastcall GetValueNames(TStrings Strings);
```

Description

Call GetValueNames to determine the names of all data values associated with the current key. Determining the names of data values is useful in an application that iterates through a set of data values for a key.

Strings is a variable of type TStrings into which to return the list of data value names.

See Also

CurrentKey (see page 6660)

GetDataInfo (see page 6627)

GetKeyNames (see page 6629)

ValueExists (see page 6645)

73.1.3.4.16 TRegistry.HasSubKeys Method

Determines if the current key has subkeys.

Pascal

```
function HasSubKeys: Boolean;
```

C++

```
__fastcall Boolean HasSubKeys();
```

Description

Call HasSubKeys to determine if the current key has any subkeys. HasSubKeys returns true if the current key has at least one subkey. Otherwise it returns false.

See Also

CurrentKey (see page 6660)

KeyExists (see page 6630)

73.1.3.4.17 TRegistry.KeyExists Method

Determines if a specified key exists.

Pascal

```
function KeyExists(const Key: string): Boolean;
```

C++

```
__fastcall Boolean KeyExists(const UnicodeString Key);
```

Description

Call KeyExists to determine if a key of a specified name exists. Key is the name of the key for which to search.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
```

```
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
```

```
      begin
        MessageDlg('Unable to create key! Exiting.',
                   mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

LoadKey (see page 6632)

ValueExists (see page 6645)

73.1.3.4.18 TRegistry.LoadKey Method

Creates a subkey under the root key and loads registry information from a file into the newly created subkey.

Pascal

```
function LoadKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean LoadKey(const UnicodeString Key, const UnicodeString FileName);
```

Description

Call LoadKey to:

1Create (see page 6615) a new subkey under the root key, and

2Load registry information from a file into the subkey. Registry (see page 6599) information can include data values, subkeys, and data values for those subkeys.

LoadKey is intended to simplify creation of a key, its values and subkeys, and the values for those subkeys in a single operation. A key, its subkeys, and all data values of the key and its subkeys is called a hive. Rather than creating each key and value separately, an application can read a hive from a file. This is especially useful for applications that users can reconfigure at run time.

Note: Before an application calls LoadKey, the RootKey (see page 6661) property must be set to HKEY_USERS, HKEY_LOCAL_MACHINE, or to a key returned by a previous call to RegistryConnect (see page 6641).

The Key parameter is the name of the subkey to create. The FileName parameter is the location of the file containing registry information to store in the subkey. The file specified by FileName must be one previously created using the SaveKey (see page 6643) function or the RegSaveKey Windows API function. On systems that use a file allocation table (FAT), FileName cannot include an extension.

See Also

RootKey (see page 6661)

CreateKey (see page 6622)

RegistryConnect (see page 6641)

SaveKey (see page 6643)

UnLoadKey (see page 6644)

73.1.3.4.19 TRegistry.MoveKey Method

Moves an existing key, its subkeys, and data values to a new location using a new key name.

Pascal

```
procedure MoveKey(const OldName: string; const NewName: string; Delete: Boolean);
```

C++

```
__fastcall MoveKey(const UnicodeString OldName, const UnicodeString NewName, Boolean Delete);
```

Description

Call MoveKey to copy or move an existing key, its subkeys, and data values to a different location. Under Windows 95, copying is recursive. If a key contains subkeys and data values, they too, are copied. When the operation is complete, the new key is closed. Under Windows NT, subkeys must be moved explicitly with separate calls to MoveKey.

OldName specifies the key to copy or move. NewName specifies a name for the duplicate key to create. If the key specified by NewName does not exist, MoveKey creates it. Delete specifies whether to delete the old key after the copy operation. If Delete is true, the key specified by OldName is deleted after it is copied. Otherwise the old key is closed.

See Also

CreateKey (see page 6622)

DeleteKey (see page 6623)

OpenKey (see page 6634)

OpenKey (see page 6634)

73.1.3.4.20 TRegistry.OpenKey Method

Opens a specified key.

Pascal

```
function OpenKey(const Key: string; CanCreate: Boolean): Boolean;
```

C++

```
__fastcall Boolean OpenKey(const UnicodeString Key, Boolean CanCreate);
```

Description

Call OpenKey to make a specified key the current key. Key is the name of the key to open. If Key is nil (Delphi) or NULL (C++), the CurrentKey (see page 6660) property is set to the key specified by the RootKey (see page 6661) property.

CanCreate specifies whether to create the specified key if it does not exist. If CanCreate is true, the key is created if necessary.

Key is opened or created with the security access value specified by the Access (see page 6657) property. OpenKey only creates non-volatile keys, A non-volatile key is stored in the registry and is preserved when the system is restarted.

OpenKey returns true if the key is successfully opened or created

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }
```

```
    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
      MessageDlg('Unable to create key! Exiting.',
                 mtError, mbOKCancel, 0);
      Exit();
    end;

  {checking if the values exist and inserting when neccesary}

  if not reg.KeyExists('Creation\ Date') then
    begin
      today := Now;
        reg.WriteDateTime('Creation\ Date', today);
    end;
```

```
    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CurrentKey (see page 6660)

RootKey (see page 6661)

CreateKey (see page 6622)

OpenKeyReadOnly (see page 6636)

Access (see page 6657)

73.1.3.4.21 TRegistry.OpenKeyReadOnly Method

Opens a specified key in a read-only manner.

Pascal

```
function OpenKeyReadOnly(const Key: String): Boolean;
```

C++

```
__fastcall Boolean OpenKeyReadOnly(const UnicodeString Key);
```

Description

Call OpenKeyReadOnly to make a specified key the current key. Key is the name of the key to open. If Key is nil (Delphi) or NULL (C++), the CurrentKey (see page 6660) property is set to the key specified by the RootKey (see page 6661) property.

Key is opened with the security access value KEY_READ.

OpenKey (see page 6634) returns true if the key is successfully opened and CurrentKey (see page 6660) is set to the key.

Note: After a successful call to OpenKeyReadOnly the Access (see page 6657) property of the registry component is automatically changed to KEY_READ, regardless of what that property was set to prior to the method call.

See Also

CurrentKey (see page 6660)

RootKey (see page 6661)

CreateKey (see page 6622)

OpenKey (see page 6634)

Access (see page 6657)

73.1.3.4.22 TRegistry.PutData Method

Pascal

```
procedure PutData(const Name: string; Buffer: Pointer; BufSize: Integer; RegData:
TRegDataType);
```

C++

```
__fastcall PutData(const UnicodeString Name, void * Buffer, int BufSize, TRegDataType RegData);
```

Description

This is PutData, a member of class TRegistry.

73.1.3.4.23 TRegistry.ReadBinaryData Method

Retrieves a binary value from a specified data value associated with the current key.

Pascal

```
function ReadBinaryData(const Name: string; var Buffer; BufSize: Integer): Integer;
```

C++

```
__fastcall int ReadBinaryData(const UnicodeString Name,  Buffer, int BufSize);
```

Description

Call ReadBinaryData to read a binary value from a specified data value associated with the current key. Name is the name of the data value to read. Buffer is the application variable into which to read the registry data. Buffer must be large enough to hold all of the data returned. BufSize specifies the size of Buffer.

If successful, ReadBinaryData writes the requested data into Buffer and returns the number of bytes written. If the Registry (see page 6599) entry contains a known type (such as a string), ReadBinaryData raises an exception.

Note: Binary data is typically a complex data structure (a record in Delphi or struct in C++). It might also be an icon or a bitmap although Microsoft recommends against storing graphics objects in the registry for performance reasons.

See Also

CurrentKey (see page 6660)

ReadBool (see page 6638)

ReadCurrency (see page 6638)

ReadDate (see page 6638)

ReadDateTime (see page 6639)

ReadFloat (see page 6639)

ReadInteger (see page 6640)

ReadString (see page 6640)

ReadTime (see page 6641)

73.1.3.4.24 TRegistry.ReadBool Method

Retrieves a Boolean value from a specified data value associated with the current key.

Pascal

```
function ReadBool(const Name: string): Boolean;
```

C++

```
__fastcall Boolean ReadBool(const UnicodeString Name);
```

Description

Call ReadBool to read a Boolean value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadBool returns the requested Boolean value. If the Registry (see page 6599) entry does not contain a boolean or integer value, an exception is raised.

See Also

CurrentKey (see page 6660)

WriteBool (see page 6646)

73.1.3.4.25 TRegistry.ReadCurrency Method

Retrieves a currency value from a specified data value associated with the current key.

Pascal

```
function ReadCurrency(const Name: string): Currency;
```

C++

```
__fastcall Currency ReadCurrency(const UnicodeString Name);
```

Description

Call ReadCurrency to read a Currency value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadCurrency returns a Currency value. If the Registry (see page 6599) entry does not contain a Currency value, an exception is raised.

See Also

CurrentKey (see page 6660)

WriteCurrency (see page 6646)

73.1.3.4.26 TRegistry.ReadDate Method

Retrieves a date value from a specified data value associated with the current key.

Pascal

```
function ReadDate(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadDate(const UnicodeString Name);
```

Description

Call ReadDate to read a date value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadDate returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the Registry (see page 6599) entry does not contain a date or time value, an exception is raised.

Note: Only use ReadDate to read data values previously stored with the WriteDate (see page 6646) procedure.

See Also

CurrentKey (see page 6660)

WriteDate (see page 6646)

ReadDateTime (see page 6639)

73.1.3.4.27 TRegistry.ReadDateTime Method

Retrieves a date and time value from a specified data value associated with the current key.

Pascal

```
function ReadDateTime(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadDateTime(const UnicodeString Name);
```

Description

Call ReadDateTime to read a datetime value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadDateTime returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the Registry (see page 6599) entry does not contain a date or time value, an exception is raised.

See Also

CurrentKey (see page 6660)

WriteDateTime (see page 6647)

ReadDate (see page 6638)

ReadTime (see page 6641)

73.1.3.4.28 TRegistry.ReadFloat Method

Retrieves a float value from a specified data value associated with the current key.

Pascal

```
function ReadFloat(const Name: string): Double;
```

C++

```
__fastcall Double ReadFloat(const UnicodeString Name);
```

Description

Call ReadFloat to read a float value from a specified data value associated with the current key. Name is the name of the data value to read.

If the Registry (see page 6599) entry does not contain a floating-point value, an exception is raised.

See Also

CurrentKey (see page 6660)

WriteFloat (see page 6652)

73.1.3.4.29 TRegistry.ReadInteger Method

Retrieves an integer value from a specified data value associated with the current key.

Pascal

```
function ReadInteger(const Name: string): Integer;
```

C++

```
__fastcall int ReadInteger(const UnicodeString Name);
```

Description

Call ReadInteger to read an integer value from a specified data value associated with the current key. Name is the name of the data value to read.

If the Registry (see page 6599) entry does not contain an integer, an exception is raised.

See Also

CurrentKey (see page 6660)

WriteInteger (see page 6653)

73.1.3.4.30 TRegistry.ReadString Method

Retrieves a string value from a specified data value associated with the current key.

Pascal

```
function ReadString(const Name: string): string;
```

C++

```
__fastcall UnicodeString ReadString(const UnicodeString Name);
```

Description

Call ReadString to read a string value from a specified data value associated with the current key. Name is the name of the data value to read.

If there is no Registry (see page 6599) entry with the specified name, ReadString returns an empty string. If the Registry (see page 6599) entry contains something other than a string, an exception is raised.

Note: ReadString retrieves data stored by either the WriteString (see page 6653) or WriteExpandString (see page 6650)

method.

See Also

CurrentKey (see page 6660)

WriteString (see page 6653)

WriteExpandString (see page 6650)

73.1.3.4.31 TRegistry.ReadTime Method

Retrieves a time value from a specified data value associated with the current key.

Pascal

```
function ReadTime(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadTime(const UnicodeString Name);
```

Description

Call ReadTime to read a time value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadTime returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the specified Registry (see page 6599) entry does not contain a date or time value, an exception is raised.

Note: Only use ReadTime to read data values previously stored with the WriteTime (see page 6656) method.

See Also

CurrentKey (see page 6660)

WriteTime (see page 6656)

ReadDateTime (see page 6639)

73.1.3.4.32 TRegistry.RegistryConnect Method

Establishes a connection to a registry on another computer.

Pascal

```
function RegistryConnect(const UNCName: string): Boolean;
```

C++

```
__fastcall Boolean RegistryConnect(const UnicodeString UNCName);
```

Description

Call RegistryConnect to establish a connection to a registry on another computer running Windows 95 or Windows NT. UNCName is the name of the remote computer, and must take the following form:

```
\\computername
```

where computername is the name of the remote computer. If UNCName is nil (Delphi) or NULL (C++), the local machine name is used.

Note: Before calling RegistryConnect, an application must set the RootKey (see page 6661) property for its registry object to HKEY_USERS or HKEY_LOCAL_MACHINE.

If connection is successful, RegistryConnect sets the RootKey (see page 6661) property to the root key of the remote computer and returns true. If unsuccessful, RegistryConnect returns false, and RootKey (see page 6661) remains unchanged.

See Also

RootKey (see page 6661)

LoadKey (see page 6632)

ReplaceKey (see page 6642)

RestoreKey (see page 6643)

SaveKey (see page 6643)

73.1.3.4.33 TRegistry.RenameValue Method

Changes the name of a data value associated with the current key.

Pascal

```
procedure RenameValue(const OldName: string; const NewName: string);
```

C++

```
__fastcall RenameValue(const UnicodeString OldName, const UnicodeString NewName);
```

Description

Call RenameValue to change the name of a data value associated with the current key. OldName is a string containing the current name of the data value. NewName is a string containing the replacement name for the data value.

If OldName is the name of an existing data value for the current key, and NewName is not the name of an existing data value for the current key, RenameValue changes the data value name as specified. Otherwise the current name remains unchanged.

See Also

CurrentKey (see page 6660)

GetValueNames (see page 6629)

ValueExists (see page 6645)

MoveKey (see page 6633)

73.1.3.4.34 TRegistry.ReplaceKey Method

Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.

Pascal

```
function ReplaceKey(const Key: string; const FileName: string; const BackUpFileName: string):
Boolean;
```

C++

```
__fastcall Boolean ReplaceKey(const UnicodeString Key, const UnicodeString FileName, const
UnicodeString BackUpFileName);
```

Description

Call ReplaceKey to replace the hive file backing a key and all its subkeys with a different hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

FileName indicates the name of the file containing the new key and subkey information. BackUpFileName is the name of a

backup file used to store the current key and subkey information before it is overwritten.

ReplaceKey returns true if it is successful.

Note: The file specified by FileName parameter must be created with the SaveKey (see page 6643) function, or by directly calling the Windows 95/NT RegSaveKey API function. Under the FAT file system FileName cannot have an extension.

See Also

LoadKey (see page 6632)

RestoreKey (see page 6643)

SaveKey (see page 6643)

73.1.3.4.35 TRegistry.RestoreKey Method

Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.

Pascal

```
function RestoreKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean RestoreKey(const UnicodeString Key, const UnicodeString FileName);
```

Description

Call RestoreKey to open the specified key with a security access value of KEY_ALL_ACCESS and overwrite any information in the key and its subkeys with registry data stored in a hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the name of the key to open. FileName is the name of the file containing the hive information.

If RestoreKey successfully reads and copies the registry information in FileName to the specified key, the return value of RestoreKey is true and the key is closed.

Note: The file specified by FileName parameter must be created with the SaveKey (see page 6643) function, or by directly calling the Windows 95/NT RegSaveKey API function. Under the FAT file system FileName cannot have an extension.

See Also

LoadKey (see page 6632)

ReplaceKey (see page 6642)

RegistryConnect (see page 6641)

SaveKey (see page 6643)

73.1.3.4.36 TRegistry.SaveKey Method

Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.

Pascal

```
function SaveKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean SaveKey(const UnicodeString Key, const UnicodeString FileName);
```

Description

Call SaveKey to open a key with a security access value of KEY_ALL_ACCESS, and save the key and its subkeys and data values to a hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the key to save. FileName is the file into which to save the key information. It must be the name of a new file that does not already exist. On FAT file systems FileName cannot include an extension.

Files created by SaveKey are passed as parameters to the RestoreKey (see page 6643), ReplaceKey (see page 6642) and LoadKey (see page 6632) functions.

If SaveKey is successful it returns true and closes the key.

See Also

LoadKey (see page 6632)

ReplaceKey (see page 6642)

RestoreKey (see page 6643)

73.1.3.4.37 TRegistry.SetCurrentKey Method

Pascal

```
procedure SetCurrentKey(Value: HKEY);
```

C++

```
__fastcall SetCurrentKey(HKEY Value);
```

Description

This is SetCurrentKey, a member of class TRegistry.

73.1.3.4.38 TRegistry.UnLoadKey Method

Removes a hive from the registry.

Pascal

```
function UnLoadKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean UnLoadKey(const UnicodeString Key);
```

Description

Call UnLoadKey to remove a hive from the registry without modifying its associated hive file, if any. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the name of the key at the root of the hive to remove.

If successful, UnLoadKey returns true.

Note: Before calling this function, set RootKey (see page 6661) to HKEY_USERS, HKEY_LOCAL_MACHINE, or call RegistryConnect (see page 6641).

See Also

RootKey (see page 6661)

LoadKey (see page 6632)

ReplaceKey (see page 6642)

RestoreKey (see page 6643)

SaveKey (see page 6643)

73.1.3.4.39 TRegistry.ValueExists Method

Determines if a specified data value exists for the current key.

Pascal

```
function ValueExists(const Name: string): Boolean;
```

C++

```
__fastcall Boolean ValueExists(const UnicodeString Name);
```

Description

Call ValueExists to determine if a particular data item exists in the registry. Calling ValueExists is especially useful before calling other TRegistry (see page 6610) methods that operate only on existing data items.

Name is the name of the data value for which to check.

ValueExists returns true if a match if found, false otherwise.

See Also

CurrentKey (see page 6660)

KeyExists (see page 6630)

73.1.3.4.40 TRegistry.WriteBinaryData Method

Stores a specifically named data value associated with the current key.

Pascal

```
procedure WriteBinaryData(const Name: string; var Buffer; BufSize: Integer);
```

C++

```
__fastcall WriteBinaryData(const UnicodeString Name,  Buffer, int BufSize);
```

Description

Call WriteBinaryData to store a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteBinary. If Name does not exist, it is created.

Buffer is a data buffer containing the data to store in the registry. BufSize indicates the size of Buffer.

Note: Data-value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as separate files, and those file names should be stored in the registry. Application elements such as icons, bitmaps, and executable files should be stored as separate files rather than as values in the registry.

If WriteBinaryData fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 6660)

ReadBinaryData (see page 6637)

73.1.3.4.41 TRegistry.WriteBool Method

Stores a Boolean value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteBool(const Name: string; Value: Boolean);
```

C++

```
__fastcall WriteBool(const UnicodeString Name, Boolean Value);
```

Description

Call WriteBool to store a boolean value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteBool. If Name does not exist, it is created.

If WriteBool fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 6660)

ReadBool (see page 6638)

73.1.3.4.42 TRegistry.WriteCurrency Method

Stores a currency value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteCurrency(const Name: string; Value: Currency);
```

C++

```
__fastcall WriteCurrency(const UnicodeString Name, Currency Value);
```

Description

Call WriteCurrency to store a Currency value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteCurrency. If Name does not exist, it is created.

Value is a Currency value to store in the registry.

If WriteCurrency fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 6660)

ReadCurrency (see page 6638)

73.1.3.4.43 TRegistry.WriteDate Method

Stores a date value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteDate(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteDate(const UnicodeString Name, TDateTime Value);
```

Description

Call WriteDate to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteDate. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteDate fails, an exception is raised, and the value is not written to the registry.

If successful, WriteDate stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

See Also

CurrentKey (see page 6660)

ReadDate (see page 6638)

WriteDateTime (see page 6647)

73.1.3.4.44 TRegistry.WriteDateTime Method

Stores a datetime value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteDateTime(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteDateTime(const UnicodeString Name, TDateTime Value);
```

Description

Call WriteDateTime to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteDateTime. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteDateTime fails, an exception is raised, and the value is not written to the registry.

If successful, WriteDateTime stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
```

```
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
```

```
    reg := TRegistry.Create(KEY_READ);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
      begin
        MessageDlg('Key not found! Created now.',
                               mtInformation, mbOKCancel, 0);
      end;
    reg.Access := KEY_WRITE;
    openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

    if not openResult = True then
      begin
        MessageDlg('Unable to create key! Exiting.',
                    mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CurrentKey (see page 6660)

ReadDateTime (see page 6639)

WriteDate (see page 6646)

WriteTime (see page 6656)

73.1.3.4.45 TRegistry.WriteExpandString Method

Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.

Pascal

```
procedure WriteExpandString(const Name: string; const Value: string);
```

C++

```
__fastcall WriteExpandString(const UnicodeString Name, const UnicodeString Value);
```

Description

Call WriteExpandString to store a string that contains unexpanded references to environment variables such as "%PATH%". The string is stored in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteExpandString. If Name does not exist, it is created.

Value is a string value to store in the registry.

If WriteExpandString fails, an exception is raised.

Note: To write strings that do not contain unexpanded environment variables, use WriteString (see page 6653) instead.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }
```

```
    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
  openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

  if not openResult = True then
    begin
      MessageDlg('Unable to create key! Exiting.',
                  mtError, mbOKCancel, 0);
      Exit();
    end;

  {checking if the values exist and inserting when neccesary}
```

```
    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CurrentKey (see page 6660)

ReadString (see page 6640)

WriteString (see page 6653)

73.1.3.4.46 TRegistry.WriteFloat Method

Stores a Double value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteFloat(const Name: string; Value: Double);
```

C++

```
__fastcall WriteFloat(const UnicodeString Name, Double Value);
```

Description

Call WriteFloat to store a float value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteFloat. If Name does not exist, it is created.

Value is a float value to store in the registry.

If WriteFloat fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 6660)

ReadFloat (see page 6639)

73.1.3.4.47 TRegistry.WriteInteger Method

Stores an integer value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteInteger(const Name: string; Value: Integer);
```

C++

```
__fastcall WriteInteger(const UnicodeString Name, int Value);
```

Description

Call WriteInteger to store an integer value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteInteger. If Name does not exist, it is created.

Value is a integer value to store in the registry.

If WriteInteger fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 6660)

ReadInteger (see page 6640)

73.1.3.4.48 TRegistry.WriteString Method

Stores a string value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteString(const Name: string; const Value: string);
```

C++

```
__fastcall WriteString(const UnicodeString Name, const UnicodeString Value);
```

Description

Call WriteString to store a string value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteString. If Name does not exist, it is created.

Value is a string value to store in the registry.

If WriteString fails, an exception is raised, and the value is not written to the registry.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
```

```
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
```

```
  find, insert and delete Keys and Items into the Windows Registry.
  This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
  on a form, for inserting and deleting the values.
  }
  procedure TForm3.InsertToRegBtnClick(Sender: TObject);
  var
    reg        : TRegistry;
    openResult : Boolean;
    today      : TDateTime;
  begin
    reg := TRegistry.Create(KEY_READ);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
      begin
        MessageDlg('Key not found! Created now.',
                              mtInformation, mbOKCancel, 0);
      end;
    reg.Access := KEY_WRITE;
    openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

    if not openResult = True then
      begin
        MessageDlg('Unable to create key! Exiting.',
                    mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');
```

```
  reg.CloseKey();
end;
```

See Also

CurrentKey (see page 6660)

ReadString (see page 6640)

WriteExpandString (see page 6650)

73.1.3.4.49 TRegistry.WriteTime Method

Stores a time value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteTime(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteTime(const UnicodeString Name, TDateTime Value);
```

Description

Call WriteTime to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteTime. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteTime fails, an exception is raised, and the value is not written to the registry.

If successful, WriteTime stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

See Also

CurrentKey (see page 6660)

ReadTime (see page 6641)

WriteDateTime (see page 6647)

73.1.3.5 TRegistry Properties

The properties of the TRegistry class are listed here.

Public Properties

	Name	Description
	Access (see page 6657)	Specifies the level of access to use when opening keys.
R	CurrentKey (see page 6660)	Specifies the registry key that is currently open.
R	CurrentPath (see page 6660)	Indicates the registry path associated with the current key.
	LazyWrite (see page 6660)	Specifies how keys are written to the registry when calling the CloseKey (see page 6619) procedure.
	RootKey (see page 6661)	Specifies the root key for the TRegistry (see page 6610) object in an application.

73.1.3.5.1 TRegistry.Access Property

Specifies the level of access to use when opening keys.

Pascal

```
property Access: LongWord;
```

C++

```
__property LongWord Access;
```

Description

Use Access to specify the level of security access to use when opening keys. The OpenKey (see page 6634) method uses the value of Access when opening a registry key. Access is initialized by the TRegistry (see page 6610) constructor, but can be altered before calling OpenKey (see page 6634).

The value of Access is a set of flags or'd together, where each flag indicates a type of permission. Choose a value that grants sufficient permission to allow the desired actions while not exceeding the access level of the system running the application. The following table lists the flags that can be included:

Flag	Meaning
KEY_ALL_ACCESS	combination of KEY_READ, KEY_WRITE, and KEY_CREATE_LINK.
KEY_READ	combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY.
KEY_WRITE	combination of KEY_SET_VALUE and KEY_CREATE_SUB_KEY.
KEY_QUERY_VALUE	grants permission to query subkey data.
KEY_ENUMERATE_SUB_KEYS	grants permission to enumerate subkeys.
KEY_NOTIFY	grants ability to receive change notifications.
KEY_SET_VALUE	grants permission to set subkey data.
KEY_CREATE_SUB_KEY	grants permission to create subkeys.
KEY_CREATE_LINK	grants permission to create symbolic links.
KEY_EXECUTE	restricts permission to read-only access.

Note: Access is ignored by the CreateKey (see page 6622) method, which always uses KEY_ALL_ACCESS.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                   mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
```

```
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
                   mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;
```

```
    if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
      begin
        MessageDlg('Key not found! Created now.',
                               mtInformation, mbOKCancel, 0);
      end;
    reg.Access := KEY_WRITE;
    openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

    if not openResult = True then
      begin
        MessageDlg('Unable to create key! Exiting.',
                     mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                               '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                               '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

OpenKey (see page 6634)

Create (see page 6615)

OpenKeyReadOnly (see page 6636)

73.1.3.5.2 TRegistry.CurrentKey Property

Specifies the registry key that is currently open.

Pascal

```
property CurrentKey: HKEY;
```

C++

```
__property HKEY CurrentKey;
```

Description

Use CurrentKey to determine the registry key that is currently open. All operations performed by TRegistry (see page 6610) affect only the current key. To access another key, call the OpenKey (see page 6634) or OpenKeyReadOnly (see page 6636) method.

See Also

CurrentPath (see page 6660)

RootKey (see page 6661)

OpenKey (see page 6634)

OpenKeyReadOnly (see page 6636)

73.1.3.5.3 TRegistry.CurrentPath Property

Indicates the registry path associated with the current key.

Pascal

```
property CurrentPath: string;
```

C++

```
__property UnicodeString CurrentPath;
```

Description

Use CurrentPath to determine the registry path of the current key. The system registry is hierarchical, and CurrentPath returns a string that spells out the hierarchy of keys from the root key to the current key, inclusive.

CurrentPath is automatically reset each time a different key becomes the current key.

See Also

CurrentKey (see page 6660)

RootKey (see page 6661)

73.1.3.5.4 TRegistry.LazyWrite Property

Specifies how keys are written to the registry when calling the CloseKey (see page 6619) procedure.

Pascal

```
property LazyWrite: Boolean;
```

C++

```
__property Boolean LazyWrite;
```

Description

Use LazyWrite to specify how keys are written to the registry. LazyWrite is initialized to true when a registry object is first created. When LazyWrite is true, keys are written to the registry when they are closed, but the CloseKey (see page 6619) procedure may return before the write operation takes place.

When LazyWrite is false, keys are written to the registry before CloseKey (see page 6619) returns. Set LazyWrite to false only when absolutely necessary. Setting LazyWrite to false guarantees that the changes an application makes to keys are written to the registry before further application processing takes place, but it uses more system resources when keys are closed. If LazyWrite is false, and an application is displaying or changing many registry entries, there will be a degradation in application performance.

See Also

CloseKey (see page 6619)

73.1.3.5.5 TRegistry.RootKey Property

Specifies the root key for the TRegistry (see page 6610) object in an application.

Pascal

```
property RootKey: HKEY;
```

C++

```
__property HKEY RootKey;
```

Description

Use RootKey to determine the hierarchy of subkeys an application can access, or to specify the root key for the application.

By default, RootKey is set to HKEY_CURRENT_USER when the registry object is created. To change the root key, specify a valid integer value for the RootKey property.

Note: For some operations, such as LoadKey (see page 6632) or RegistryConnect (see page 6641), an application may need to change the default value of RootKey.

C++ Examples:

```
/*
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
for inserting and deleting the values.
*/
void __fastcall TForm2::InsertToRegBtnClick(TObject *Sender)
{
    TRegistry* reg = new TRegistry(KEY_READ);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    if(!reg->KeyExists("Software\\MyCompanyName\\MyApplication\\"))
    {
        MessageDlg("Key not found! Created now.",
                    mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    reg->Access = KEY_WRITE;
    bool openResult = reg->OpenKey("Software\\MyCompanyName\\MyApplication\\", true);

    if(!openResult)
    {
        MessageDlg("Unable to create key! Exiting.",
```

```
                    mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }
    //checking if the values exist and inserting when neccesary

    if(!reg->KeyExists("Creation\ Date"))
    {
        TDateTime today = TDateTime::CurrentDateTime();
        reg->WriteDateTime("Creation\ Date", today);
    }

    if(!reg->KeyExists("Licenced\ To"))
    {
        reg->WriteString("Licenced\ To", "MySurname\ MyFirstName");
    }

    if(!reg->KeyExists("App\ Location"))
    {
        reg->WriteExpandString("App\ Location",
                               "%PROGRAMFILES%\\MyCompanyName\\MyApplication\\");
    }

    if(!reg->KeyExists("Projects\ Location"))
    {
        reg->WriteExpandString("Projects\ Location",
                               "%USERPROFILE%\\MyApplication\\Projects\\");
    }

    reg->CloseKey();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DelFromRegBtnClick(TObject *Sender)
{
    //deleting the example registry entries
    TRegistry* reg = new TRegistry(KEY_WRITE);
    reg->RootKey = HKEY_LOCAL_MACHINE;

    reg->DeleteKey("Software\\MyCompanyName\\MyApplication");
    reg->DeleteKey("Software\\MyCompanyName");

    reg->CloseKey();
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example shows how to use the TRegistry class in order to
find, insert and delete Keys and Items into the Windows Registry.
This example uses two buttons: InsertToRegBtn and DeleteFromRegBtn,
on a form, for inserting and deleting the values.
}
procedure TForm3.InsertToRegBtnClick(Sender: TObject);
var
  reg        : TRegistry;
  openResult : Boolean;
  today      : TDateTime;
begin
  reg := TRegistry.Create(KEY_READ);
  reg.RootKey := HKEY_LOCAL_MACHINE;

  if (not reg.KeyExists('Software\\MyCompanyName\\MyApplication\\')) then
    begin
      MessageDlg('Key not found! Created now.',
                          mtInformation, mbOKCancel, 0);
    end;
  reg.Access := KEY_WRITE;
```

```
    openResult := reg.OpenKey('Software\\MyCompanyName\\MyApplication\\',True);

    if not openResult = True then
      begin
        MessageDlg('Unable to create key! Exiting.',
                   mtError, mbOKCancel, 0);
        Exit();
      end;

    {checking if the values exist and inserting when neccesary}

    if not reg.KeyExists('Creation\ Date') then
      begin
        today := Now;
          reg.WriteDateTime('Creation\ Date', today);
      end;

    if not reg.KeyExists('Licenced\ To') then
      begin
          reg.WriteString('Licenced\ To', 'MySurname\ MyFirstName');
      end;

    if not reg.KeyExists('App\ Location') then
      begin
          reg.WriteExpandString('App\ Location',
                                '%PROGRAMFILES%\\MyCompanyName\\MyApplication\\');
      end;

    if not reg.KeyExists('Projects\ Location') then
      begin
          reg.WriteExpandString('Projects\ Location',
                                '%USERPROFILE%\\MyApplication\\Projects\\');
      end;

    reg.CloseKey();

  end;

  procedure TForm3.DeleteFromRegBtnClick(Sender: TObject);
  var
    reg : TRegistry;
  begin
    reg := TRegistry.Create(KEY_WRITE);
    reg.RootKey := HKEY_LOCAL_MACHINE;

    reg.DeleteKey('Software\\MyCompanyName\\MyApplication');
    reg.DeleteKey('Software\\MyCompanyName');

    reg.CloseKey();
  end;
```

See Also

CurrentKey (see page 6660)

LoadKey (see page 6632)

RegistryConnect (see page 6641)

73.1.4 TRegistryIniFile Class

TRegistryIniFile is a wrapper for the Windows system registry.

Class Hierarchy

Pascal

```
TRegistryIniFile = class(TCustomIniFile);
```

C++

```
class TRegistryIniFile : public TCustomIniFile;
```

File

Registry

Description

TRegistryIniFile presents a simple interface to the system registry and hides the need to know about the underlying structure of the registry. TRegistryIniFile enables handling the Windows system registry as if it were a Windows 3.x INI file. Instead of processing an INI file, however, TRegistryIniFile reads from and writes to the system registry. Because TRegistryIniFile, like TIniFile (see page 4294), descends from TCustomIniFile (see page 4258), you can use it in common code that writes to both INI files and the registry.

In addition, application developers that are switching from the use of INI files to the system registry can use this object to migrate their application with a minimum of coding changes. By finding all references to TIniFile (see page 4294) in an application, replacing them with TRegistryIniFile, and recompiling the application, a developer can update an application to use the system registry instead of INI files without having to code any new logic into the application.

Note: Information contained in existing users' INI files is not migrated into the system registry using the method described above. A one-time operation to explicitly copy the INI file data, through TRegistryIniFile, must be used to do this.

TRegistryIniFile adapts the methods inherited from TCustomIniFile (see page 4258) to operate on Registry (see page 6599) entries rather than INI file keys. It reinterprets the FileName property as a subkey under the system registry's root key (HKEY_CURRENT_USER by default). What corresponds to a section in an INI file is treated as a key in the system registry, and what corresponds to data entries under a section in an INI file are treated as data values under a key in the system registry.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}
```

```
void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
```

```
      {
      Read all saved values from the last session. The section name
      is the name of the form. Also use form's properties as defaults
      }
      Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
      Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
      Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
      Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
      Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

      { Load last window state }
      case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
        true : WindowState := wsMaximized;
        false: WindowState := wsNormal;
      end;

    finally
      SettingsFile.Free;
    end;
  end;

  procedure TForm2.btStoreClick(Sender: TObject);
  var
    SettingsFile: TCustomIniFile;
  begin
    { Open an instance }
    SettingsFile := OpenIniFileInstance();

    {
    Store current form properties to be used in later sessions.
    }
    try
      SettingsFile.WriteInteger (Name, 'Top', Top);
      SettingsFile.WriteInteger (Name, 'Left', Left);
      SettingsFile.WriteInteger (Name, 'Width', Width);
      SettingsFile.WriteInteger (Name, 'Height', Height);
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
```

```
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

TIniFile (see page 4294)

TMemIniFile (see page 4307)

TCustomIniFile (see page 4258)

TRegIniFile (see page 6601)

73.1.4.1 TRegistryIniFile Members

The following tables list the members exposed by TRegistryIniFile.

Public Methods

	Name	Description
	Create (see page 6668)	This is the overview for the Create constructor overload.
	DeleteKey (see page 6669)	Erases the data value associated with an INI file entry.
	Destroy (see page 6668)	Disposes of an object instance.
	EraseSection (see page 6670)	Erases an entire section of an INI file.
	ReadBinaryStream (see page 6670)	Retrieves a binary value from the registry.
	ReadBool (see page 4265)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 6671)	Retrieves a date value from an INI file.
	ReadDateTime (see page 6671)	Retrieves a date-time value from an INI file.
	ReadFloat (see page 6672)	Retrieves a float value from an INI file.
	ReadInteger (see page 6672)	Retrieves an integer value from an INI file.
	ReadSection (see page 6676)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 4273)	This is the overview for the ReadSections method overload.
	ReadSections (see page 6676)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 6677)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 6677)	Retrieves a string value from an INI file.
	ReadTime (see page 6680)	Retrieves a time value from an INI file.
	SectionExists (see page 4278)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 6681)	Flushes buffered INI file data to disk.
	ValueExists (see page 4278)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 6681)	Writes a binary value to the registry.
	WriteBool (see page 4279)	Writes a Boolean value to an ini file.
	WriteDate (see page 6682)	Writes a date value to an INI file.
	WriteDateTime (see page 6682)	Writes a date-time value to an INI file.
	WriteFloat (see page 6683)	Writes a float value to an INI file.
	WriteInteger (see page 6683)	Writes an integer value to an INI file.
	WriteString (see page 6686)	Writes a string value to an INI file.
	WriteTime (see page 6689)	Writes a time value to an INI file.

Public Properties

	Name	Description
	FileName (see page 4291)	Contains the name of the ini file from which to read and to which to write information.
	RegIniFile (see page 6690)	Contains and internal instance of a TRegIniFile (see page 6601) object.

73.1.4.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegistryIniFile.Create (string) (see page 6668)	Creates a TRegistryIniFile (see page 6663) object for an application.
	TRegistryIniFile.Create (string, LongWord) (see page 6668)	Creates a TRegistryIniFile (see page 6663) object for an application.

73.1.4.2.1 TRegistryIniFile.Create Constructor (string)

Creates a TRegistryIniFile (see page 6663) object for an application.

Pascal

```
constructor Create(const FileName: string); overload;
constructor Create(const FileName: string; AAccess: LongWord); overload;
```

C++

```
__fastcall TRegistryIniFile(const UnicodeString FileName);
__fastcall TRegistryIniFile(const UnicodeString FileName, LongWord AAccess);
```

Description

Create constructs a TRegistryIniFile (see page 6663) and creates an internal TRegIniFile (see page 6601) object used to access the system registry. TRegistryIniFile (see page 6663) uses this TRegIniFile (see page 6601) object as a helper object to provide much of its functionality.

FileName is used as the FileName property for the internal TRegIniFile (see page 6601) object in the RegIniFile (see page 6690) property.

AAccess initializes the Access property of the internal TRegIniFile (see page 6601) object.

See Also

TRegIniFile (see page 6601)

RegIniFile (see page 6690)

System::TObject::Create

Access (see page 6657)

73.1.4.3 TRegistryIniFile.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistryIniFile();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free

73.1.4.4 TRegistryIniFile Methods

The methods of the TRegistryIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 6669)	Erases the data value associated with an INI file entry.
	EraseSection (see page 6670)	Erases an entire section of an INI file.
	ReadBinaryStream (see page 6670)	Retrieves a binary value from the registry.
	ReadDate (see page 6671)	Retrieves a date value from an INI file.
	ReadDateTime (see page 6671)	Retrieves a date-time value from an INI file.
	ReadFloat (see page 6672)	Retrieves a float value from an INI file.
	ReadInteger (see page 6672)	Retrieves an integer value from an INI file.
	ReadSection (see page 6676)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 6676)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 6677)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 6677)	Retrieves a string value from an INI file.
	ReadTime (see page 6680)	Retrieves a time value from an INI file.
	UpdateFile (see page 6681)	Flushes buffered INI file data to disk.
	WriteBinaryStream (see page 6681)	Writes a binary value to the registry.
	WriteDate (see page 6682)	Writes a date value to an INI file.
	WriteDateTime (see page 6682)	Writes a date-time value to an INI file.
	WriteFloat (see page 6683)	Writes a float value to an INI file.
	WriteInteger (see page 6683)	Writes an integer value to an INI file.
	WriteString (see page 6686)	Writes a string value to an INI file.
	WriteTime (see page 6689)	Writes a time value to an INI file.

73.1.4.4.1 TRegistryIniFile.DeleteKey Method

Erases the data value associated with an INI file entry.

Pascal

```
procedure DeleteKey(const Section: string; const Ident: string); override;
```

C++

```
virtual __fastcall DeleteKey(const UnicodeString Section, const UnicodeString Ident);
```

Description

Call DeleteKey to erase a data value associated with an INI file entry. Section is string containing the name of an INI file section, and Ident is a string containing the name of the key from for which to set a nil (Delphi) or NULL (C++) value.

Note: Attempting to erase a data value in a nonexistent section or attempting to erase data from a nonexistent key are not errors. In these cases, DeleteKey creates the section and key and sets its initial value to an empty value.

See Also

EraseSection (see page 6670)

ValueExists (see page 4278)

73.1.4.4.2 TRegistryIniFile.EraseSection Method

Erases an entire section of an INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const UnicodeString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

See Also

SectionExists (see page 4278)

DeleteKey (see page 6669)

73.1.4.4.3 TRegistryIniFile.ReadBinaryStream Method

Retrieves a binary value from the registry.

Pascal

```
function ReadBinaryStream(const Section: string; const Name: string; Value: TStream): Integer;
override;
```

C++

```
virtual __fastcall int ReadBinaryStream(const UnicodeString Section, const UnicodeString Name,
TStream Value);
```

Description

Call ReadBinaryStream to read a binary value into a specified stream. After reading the binary value from the INI file into the current position of the stream, ReadBinaryStream returns the number of bytes written to the stream. The stream is left positioned immediately after the new binary value.

Section is the key for the section that contains the binary data.

Name is the subkey under which the binary data is stored.

Value is a stream to which the binary value is written.

See Also

ReadString (see page 6677)

WriteBinaryStream (see page 6681)

73.1.4.4.4 TRegistryIniFile.ReadDate Method

Retrieves a date value from an INI file.

Pascal

```
function ReadDate(const Section: string; const Name: string; Default: TDateTime): TDateTime;
override;
```

C++

```
virtual __fastcall TDateTime ReadDate(const UnicodeString Section, const UnicodeString Name,
TDateTime Default);
```

Description

Call ReadDate to read a date value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date value. Default is the date value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDateTime (see page 6671)

ReadFloat (see page 6672)

ReadInteger (see page 6672)

ReadString (see page 6677)

ReadTime (see page 6680)

73.1.4.4.5 TRegistryIniFile.ReadDateTime Method

Retrieves a date-time value from an INI file.

Pascal

```
function ReadDateTime(const Section: string; const Name: string; Default: TDateTime):
TDateTime; override;
```

C++

```
virtual __fastcall TDateTime ReadDateTime(const UnicodeString Section, const UnicodeString
Name, TDateTime Default);
```

Description

Call ReadDateTime to read a date-time value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date-time value. Default is the date-time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 6671)

ReadDateTime

ReadFloat (see page 6672)

ReadInteger (see page 6672)

ReadString (see page 6677)

ReadTime (see page 6680)

73.1.4.4.6 TRegistryIniFile.ReadFloat Method

Retrieves a float value from an INI file.

Pascal

```
function ReadFloat(const Section: string; const Name: string; Default: Double): Double;
override;
```

C++

```
virtual __fastcall Double ReadFloat(const UnicodeString Section, const UnicodeString Name,
Double Default);
```

Description

Call ReadFloat to read a float value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the float value. Default is the float value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 6671)

ReadDateTime (see page 6671)

ReadInteger (see page 6672)

ReadString (see page 6677)

ReadTime (see page 6680)

73.1.4.4.7 TRegistryIniFile.ReadInteger Method

Retrieves an integer value from an INI file.

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
override;
```

C++

```
virtual __fastcall Longint ReadInteger(const UnicodeString Section, const UnicodeString Ident,
Longint Default);
```

Description

Call ReadInteger to read an integer value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the integer value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);
```

```
        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
```

```
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;

end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

ReadBool (see page 4265)

ReadDate (see page 6671)

ReadDateTime (see page 6671)

ReadFloat (see page 6672)

ReadString (see page 6677)

ReadTime (see page 6680)

73.1.4.4.8 TRegistryIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object such as a string list, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSections (see page 6676)

ReadSectionValues (see page 6677)

SectionExists (see page 4278)

73.1.4.4.9 ReadSections Method

This is the overview for the ReadSections method overload.

Overload List

	Name	Description
	TRegistryIniFile.ReadSections (string, TStrings) (see page 6676)	Reads the names of all sections in an INI file into a string list.
	TRegistryIniFile.ReadSections (TStrings) (see page 6676)	Reads the names of all sections in an INI file into a string list.

73.1.4.4.9.1 TRegistryIniFile.ReadSections Method (TStrings)

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override; overload;
procedure ReadSections(const Section: string; Strings: TStrings); override; overload;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
virtual __fastcall ReadSections(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list object.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSection (see page 6676)

ReadSectionValues (see page 6677)

73.1.4.4.10 TRegistryIniFile.ReadSectionValues Method

Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const UnicodeString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read all data values associated with a specified key into a string object.

Section is the name of the key from which to retrieve data-values.

Strings specifies the string list object to hold the retrieved values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself. Use the Values property of the string object to access a specific string in the object.

See Also

ReadSection (see page 6676)

ReadSections (see page 6676)

SectionExists (see page 4278)

73.1.4.4.11 TRegistryIniFile.ReadString Method

Retrieves a string value from an INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall UnicodeString ReadString(const UnicodeString Section, const UnicodeString
Ident, const UnicodeString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
```

```
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
```

```
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
    SettingsFile.Free;
  end;
```

```
end;

function TForm2.OpenIniFileInstance: TCustomIniFile;
begin
  {
  Open/create a new INI file that has the same name as our executable
  only with the INI extension.
  }

  case RadioGroup1.ItemIndex of
    0:
      begin
        { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
        Result := TRegistryIniFile.Create('Software\' + Application.Title);
      end;
    1:
      begin
        { Ini file mode selected }
        Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
    2:
      begin
        { Memory based Ini file mode selected }
        Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
      end;
  end;
end;
```

See Also

ReadBool (see page 4265)

ReadDate (see page 6671)

ReadDateTime (see page 6671)

ReadFloat (see page 6672)

ReadInteger (see page 6672)

ReadTime (see page 6680)

73.1.4.4.12 TRegistryIniFile.ReadTime Method

Retrieves a time value from an INI file.

Pascal

```
function ReadTime(const Section: string; const Name: string; Default: TDateTime): TDateTime;
override;
```

C++

```
virtual __fastcall TDateTime ReadTime(const UnicodeString Section, const UnicodeString Name,
TDateTime Default);
```

Description

Call ReadTime to read a time value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the time value. Default is the time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 4265)

ReadDate (see page 6671)

ReadDateTime (see page 6671)

ReadFloat (see page 6672)

ReadInteger (see page 6672)

ReadString (see page 6677)

73.1.4.4.13 TRegistryIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to flush buffered reads from and writes to the INI file to disk. UpdateFile is useful under Windows95, but has no effect under Windows NT as NT does not buffer INI file reads and writes.

See Also

FileName (see page 4291)

73.1.4.4.14 TRegistryIniFile.WriteBinaryStream Method

Writes a binary value to the registry.

Pascal

```
procedure WriteBinaryStream(const Section: string; const Name: string; Value: TStream);
override;
```

C++

```
virtual __fastcall WriteBinaryStream(const UnicodeString Section, const UnicodeString Name,
TStream Value);
```

Description

Call WriteBinaryStream to write a binary value from a specified stream to a specified key in the registry. WriteBinaryStream writes from the current position in the stream to the end.

Section is the key for the section that contains the subkey where the binary data should be written.

Name is the subkey under which the binary data is stored.

Value is a stream from which the binary value can be read.

Note: Attempting to write a value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteBinaryStream creates the section and key and sets its initial value to the stream's data.

See Also

WriteString (see page 6686)

ReadBinaryStream (see page 6670)

73.1.4.4.15 TRegistryIniFile.WriteDate Method

Writes a date value to an INI file.

Pascal

```
procedure WriteDate(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteDate(const UnicodeString Section, const UnicodeString Name, TDateTime
Value);
```

Description

Call WriteDate to write a date value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDate creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDateTime (see page 6682)

WriteFloat (see page 6683)

WriteInteger (see page 6683)

WriteString (see page 6686)

WriteTime (see page 6689)

73.1.4.4.16 TRegistryIniFile.WriteDateTime Method

Writes a date-time value to an INI file.

Pascal

```
procedure WriteDateTime(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteDateTime(const UnicodeString Section, const UnicodeString Name,
TDateTime Value);
```

Description

Call WriteDateTime to write a date-time value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date-time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDateTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 6682)

WriteFloat (see page 6683)

WriteInteger (see page 6683)

WriteString (see page 6686)

WriteTime (see page 6689)

73.1.4.4.17 TRegistryIniFile.WriteFloat Method

Writes a float value to an INI file.

Pascal

```
procedure WriteFloat(const Section: string; const Name: string; Value: Double); override;
```

C++

```
virtual __fastcall WriteFloat(const UnicodeString Section, const UnicodeString Name, Double
Value);
```

Description

Call WriteFloat to write a float value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the float value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteFloat creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 6682)

WriteDateTime (see page 6682)

WriteInteger (see page 6683)

WriteString (see page 6686)

WriteTime (see page 6689)

73.1.4.4.18 TRegistryIniFile.WriteInteger Method

Writes an integer value to an INI file.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint); override;
```

C++

```
virtual __fastcall WriteInteger(const UnicodeString Section, const UnicodeString Ident,
Longint Value);
```

Description

Call WriteInteger to write an integer value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the integer value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteInteger creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    */

    switch (RadioGroup1->ItemIndex)
    {
```

```
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
    SettingsFile.WriteString  (Name, 'Caption', Caption);
    SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
    SettingsFile.WriteDateTime(Name, 'LastRun', Now);
  finally
```

```
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

WriteBool (see page 4279)

WriteDate (see page 6682)

WriteDateTime (see page 6682)

WriteFloat (see page 6683)

WriteString (see page 6686)

WriteTime (see page 6689)

73.1.4.4.19 TRegistryIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: string; const Ident: string; const Value: string);
override;
```

C++

```
virtual __fastcall WriteString(const UnicodeString Section, const UnicodeString Ident, const
UnicodeString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

C++ Examples:

```
/*
This example demonstrates the usage of Ini files in order to store
and load form configuration between sessions. Example assumes
a RadioGroup and two buttons are present on the main form.
*/
void __fastcall TForm2::btStoreClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    // Store current form properties to be used in later sessions.
    try
    {
        SettingsFile->WriteInteger (Name, "Top", Top);
        SettingsFile->WriteInteger (Name, "Left", Left);
        SettingsFile->WriteInteger (Name, "Width", Width);
        SettingsFile->WriteInteger (Name, "Height", Height);
        SettingsFile->WriteString  (Name, "Caption", Caption);
        SettingsFile->WriteBool    (Name, "InitMax", WindowState == wsMaximized );
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

void __fastcall TForm2::btLoadClick(TObject *Sender)
{
    /* Open an instance */
    TCustomIniFile* SettingsFile = OpenIniFileInstance();

    try
    {
        /*
        Read all saved values from the last session. The section name
        is the name of the form. Also use form's properties as defaults
        */
        Top     = SettingsFile->ReadInteger(Name, "Top", Top );
        Left    = SettingsFile->ReadInteger(Name, "Left", Left );
        Width   = SettingsFile->ReadInteger(Name, "Width", Width );
        Height  = SettingsFile->ReadInteger(Name, "Height", Height );
        Caption = SettingsFile->ReadString (Name, "Caption", Caption);

        // Load last window state
        if (SettingsFile->ReadBool(Name, "InitMax", WindowState == wsMaximized))
            WindowState = wsMaximized;
        else
            WindowState = wsNormal;
    }
    catch(Exception* e)
    {
    }

    delete SettingsFile;
}

TCustomIniFile* __fastcall TForm2::OpenIniFileInstance()
{
    /*
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
```

```
    */

    switch (RadioGroup1->ItemIndex)
    {
        case 0:
            /* Registry mode selected: in HKEY_CURRENT_USER\Software\... */
            return new TRegistryIniFile(String("Software\\") + Application->Title);
        case 1:
            /* Ini file mode selected */
            return new TIniFile(ChangeFileExt(Application->ExeName, ".INI"));
        case 2:
            /* Memory based Ini file mode selected */
            return new TMemIniFile(ChangeFileExt(Application->ExeName, ".INI"));
      }
}
```

Delphi Examples:

```
{
This example demonstrates the use of Ini files in order to store
and load form configurations between sessions. This example requires
a RadioGroup and two buttons.
}
procedure TForm2.btLoadClick(Sender: TObject);
var
  SettingsFile : TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  try
    {
    Read all saved values from the last session. The section name
    is the name of the form. Also use form's properties as defaults
    }
    Top     := SettingsFile.ReadInteger(Name, 'Top', Top );
    Left    := SettingsFile.ReadInteger(Name, 'Left', Left );
    Width   := SettingsFile.ReadInteger(Name, 'Width', Width );
    Height  := SettingsFile.ReadInteger(Name, 'Height', Height );
    Caption := SettingsFile.ReadString (Name, 'Caption', Caption);

    { Load last window state }
    case SettingsFile.ReadBool(Name, 'InitMax', WindowState = wsMaximized) of
      true : WindowState := wsMaximized;
      false: WindowState := wsNormal;
    end;

  finally
    SettingsFile.Free;
  end;
end;

procedure TForm2.btStoreClick(Sender: TObject);
var
  SettingsFile: TCustomIniFile;
begin
  { Open an instance }
  SettingsFile := OpenIniFileInstance();

  {
  Store current form properties to be used in later sessions.
  }
  try
    SettingsFile.WriteInteger (Name, 'Top', Top);
    SettingsFile.WriteInteger (Name, 'Left', Left);
    SettingsFile.WriteInteger (Name, 'Width', Width);
    SettingsFile.WriteInteger (Name, 'Height', Height);
```

```
      SettingsFile.WriteString  (Name, 'Caption', Caption);
      SettingsFile.WriteBool    (Name, 'InitMax', WindowState = wsMaximized );
      SettingsFile.WriteDateTime(Name, 'LastRun', Now);
    finally
      SettingsFile.Free;
    end;

  end;

  function TForm2.OpenIniFileInstance: TCustomIniFile;
  begin
    {
    Open/create a new INI file that has the same name as our executable
    only with the INI extension.
    }

    case RadioGroup1.ItemIndex of
      0:
        begin
          { Registry mode selected: in HKEY_CURRENT_USER\Software\... }
          Result := TRegistryIniFile.Create('Software\' + Application.Title);
        end;
      1:
        begin
          { Ini file mode selected }
          Result := TIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
      2:
        begin
          { Memory based Ini file mode selected }
          Result := TMemIniFile.Create(ChangeFileExt(Application.ExeName, '.INI'));
        end;
    end;
  end;
```

See Also

WriteBool (see page 4279)

WriteDate (see page 6682)

WriteDateTime (see page 6682)

WriteFloat (see page 6683)

WriteInteger (see page 6683)

WriteTime (see page 6689)

73.1.4.4.20 TRegistryIniFile.WriteTime Method

Writes a time value to an INI file.

Pascal

```
procedure WriteTime(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteTime(const UnicodeString Section, const UnicodeString Name, TDateTime
Value);
```

Description

Call WriteTime to write a time value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 4279)

WriteDate (see page 6682)

WriteDateTime (see page 6682)

WriteFloat (see page 6683)

WriteInteger (see page 6683)

WriteString (see page 6686)

73.1.4.5 TRegistryIniFile Properties

The properties of the TRegistryIniFile class are listed here.

Public Properties

	Name	Description
	RegIniFile (see page 6690)	Contains and internal instance of a TRegIniFile (see page 6601) object.

73.1.4.5.1 TRegistryIniFile.RegIniFile Property

Contains and internal instance of a TRegIniFile (see page 6601) object.

Pascal

```
property RegIniFile: TRegIniFile;
```

C++

```
__property TRegIniFile RegIniFile;
```

Description

RegIniFile is used internally by the TRegistryIniFile (see page 6663) object as the gateway connection between TRegistryIniFile (see page 6663) methods and the system registry.

See Also

Create (see page 6668)

TRegIniFile (see page 6601)

73.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRegDataType (see page 6691)	TRegDataType is a type used by the GetDataInfo method of TRegistery.

Records

	Name	Description
	TRegDataInfo (see page 6691)	TRegDataInfo is a type used by the GetDataInfo method of TRegistery.
	TRegKeyInfo (see page 6692)	TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

73.2.1 Registry.TRegDataInfo Record

TRegDataInfo is a type used by the GetDataInfo method of TRegistery.

Pascal

```
TRegDataInfo = record
  RegData: TRegDataType;
  DataSize: Integer;
end;
```

C++

```
struct TRegDataInfo {
  TRegDataType RegData;
  int DataSize;
};
```

File

Registry

Description

TRegDataInfo is a type used by the GetDataInfo method of TRegistery. Records of type TRegDataInfo contain two fields: a field called RegData, of type TRegDataType (see page 6691), that specifies the data type of the data value; and a field called DataSize that specifies the size, in bytes, of the data value.

73.2.2 Registry.TRegDataType Enumeration

TRegDataType is a type used by the GetDataInfo method of TRegistery.

Pascal

```
TRegDataType = (
  rdUnknown,
  rdString,
  rdExpandString,
  rdInteger,
```

```
    rdBinary
  );
```

C++

```
  enum TRegDataType {
    rdUnknown,
    rdString,
    rdExpandString,
    rdInteger,
    rdBinary
  };
```

File

Registry

Description

TRegDataType is a type used by the GetDataInfo method of TRegistery.It can have the following values:

Value	Meaning
rdUnknown	Data value type is of no defined value type.
rdString	Data value type is a null-terminated string. It is a Unicode or ANSI string depending on whether the application calls the Unicode or ANSI string functions.
rdExpandString	Data value type is a null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It is a Unicode or ANSI string depending on whether the application calls the Unicode or ANSI string functions.
rdInteger	Data value type is a32-bit number.
rdBinary	Data value type is binary data, in any form.

73.2.3 Registry.TRegKeyInfo Record

TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

Pascal

```
  TRegKeyInfo = record
    NumSubKeys: Integer;
    MaxSubKeyLen: Integer;
    NumValues: Integer;
    MaxValueLen: Integer;
    MaxDataLen: Integer;
    FileTime: TFileTime;
  end;
```

C++

```
  struct TRegKeyInfo {
    int NumSubKeys;
    int MaxSubKeyLen;
    int NumValues;
    int MaxValueLen;
    int MaxDataLen;
    TFileTime FileTime;
  };
```

File

Registry

Description

TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery to return information about the current key:

Note: Number of subkeys

Note: Longest subkey name length, in characters

Note: Number of data values

Note: Longest data-value name length

Note: Longest data-value length

Note: Time of last write to the key

74 ReqFiles Namespace

This is namespace ReqFiles.

Topics

Name	Description
Classes (see page 6696)	The following table lists classes in this documentation.

Classes

	Name	Description
	TWebRequestFile (see page 6696)	TWebRequestFile describes the content of a file extracted from an HTTP request.
	TWebRequestFiles (see page 6700)	TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.
	TWebRequestFileStream (see page 6703)	This is class ReqFiles.TWebRequestFileStream.

74.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebRequestFile (see page 6696)	TWebRequestFile describes the content of a file extracted from an HTTP request.
	TWebRequestFiles (see page 6700)	TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.
	TWebRequestFileStream (see page 6703)	This is class ReqFiles.TWebRequestFileStream.

74.1.1 TWebRequestFile Class

TWebRequestFile describes the content of a file extracted from an HTTP request.

Class Hierarchy

Pascal

```
TWebRequestFile = class(TAbstractWebRequestFile);
```

C++

```
class TWebRequestFile : public TAbstractWebRequestFile;
```

File

ReqFiles

Description

TWebRequestFile describes and provides access to an uploaded file from an HTTP request. The following properties support this access:

FileName, which gives the name of the uploaded file.

FieldName, which gives the name of the adapter field whose value is the uploaded file.

ContentType, which describes how the file data is encoded.

Stream, which is a stream object from which you can read the file content.

Each TWebRequestFile object describes a single uploaded file. A collection of all the uploaded files from the current request is available as the Files property of TWebRequest (see page 3358). In addition, adapter fields that work with uploaded files surface those files using a TUpdateFileList object or IActionFieldValues interface.

See Also

TWebRequestFiles (see page 6700)

IActionFieldValues

TUpdateFileList

TAdapterFileField

74.1.1.1 TWebRequestFile Members

The following tables list the members exposed by TWebRequestFile.

Protected Methods

	Name	Description
	GetContentType (see page 6698)	Returns the value of the ContentType property.
	GetFieldName (see page 6699)	Returns the value of the FieldName property.
	GetFileName (see page 6699)	Returns the value of the FileName property.
	GetStream (see page 6699)	Returns the value of the Stream property.

Public Methods

	Name	Description
	Create (see page 6697)	Creates an initializes an instance of TWebRequestFile (see page 6696).
	Destroy (see page 6698)	Frees an instance of TWebRequestFile (see page 6696).

Public Properties

	Name	Description
	ContentType (see page 3291)	Describes how the file data is encoded.
	FieldName (see page 3291)	Indicates the name of the adapter field whose value is the contents of the file.
	FileName (see page 3291)	Indicates the name of the uploaded file.
	Stream (see page 3292)	Provides access to the contents of the uploaded file.

74.1.1.2 TWebRequestFile.Create Constructor

Creates an initializes an instance of TWebRequestFile (see page 6696).

Pascal

```
constructor Create(const AFieldName: AnsiString; const AFileName: AnsiString; const
AContentType: AnsiString; AContent: Pointer; AContentLength: Integer);
```

C++

```
__fastcall TWebRequestFile(const AnsiString AFieldName, const AnsiString AFileName, const
AnsiString AContentType, void * AContent, int AContentLength);
```

Description

Typically, applications do not call the TWebRequestFile (see page 6696) constructor. Instead, new Web request file objects are created by calling the Add method of a TWebRequestFiles (see page 6700) collection as the file uploads are extracted from the current Web request.

AFieldName is the name of the adapter field whose value this file represents. It becomes the value of the FieldName property.

AFileName is the name of the file this object represents. It becomes the value of the FileName property.

AContentType describes how the file content is encoded. It becomes the value of the ContentType property.

AContent is an array of bytes that is the content of the file. It is used to initialize the Stream property.

AContentLength is the number of bytes in AContent.

See Also

FieldName

FileName

ContentType

Stream

Add (see page 6702)

74.1.1.3 TWebRequestFile.Destroy Destructor

Frees an instance of TWebRequestFile (see page 6696).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestFile();
```

Description

Do not free instances of TWebRequestFile (see page 6696). TWebRequestFile (see page 6696) objects are created and destroyed by the TWebRequestFiles (see page 6700) object that lists the uploaded files from the current request.

See Also

TWebRequestFiles (see page 6700)

74.1.1.4 TWebRequestFile Methods

The methods of the TWebRequestFile class are listed here.

Protected Methods

	Name	Description
	GetContentType (see page 6698)	Returns the value of the ContentType property.
	GetFieldName (see page 6699)	Returns the value of the FieldName property.
	GetFileName (see page 6699)	Returns the value of the FileName property.
	GetStream (see page 6699)	Returns the value of the Stream property.

74.1.1.4.1 TWebRequestFile.GetContentType Method

Returns the value of the ContentType property.

Pascal

```
function GetContentType: AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetContentType();
```

Description

GetContentType is the protected read implementation for the ContentType property.

In TAbstractWebRequestFile, GetContentType is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ContentType property.

See Also

ContentType (see page 3291)

74.1.1.4.2 TWebRequestFile.GetFieldName Method

Returns the value of the FieldName property.

Pascal

```
function GetFieldName: AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetFieldName();
```

Description

GetFieldName is the protected read implementation for the FieldName property.

In TAbstractWebRequestFile, GetFieldName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FieldName property.

See Also

FieldName (see page 3291)

74.1.1.4.3 TWebRequestFile.GetFileName Method

Returns the value of the FileName property.

Pascal

```
function GetFileName: AnsiString; override;
```

C++

```
virtual __fastcall AnsiString GetFileName();
```

Description

GetFileName is the protected read implementation for the FileName property.

In TAbstractWebRequestFile, GetFileName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FileName property.

See Also

FileName (see page 3291)

74.1.1.4.4 TWebRequestFile.GetStream Method

Returns the value of the Stream property.

Pascal

```
function GetStream: TStream; override;
```

C++

```
virtual __fastcall TStream GetStream();
```

Description

GetStream is the protected read implementation for the Stream property.

In TAbstractWebRequestFile, GetStream is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Stream property.

See Also

Stream (see page 3292)

74.1.2 TWebRequestFiles Class

TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.

Class Hierarchy

Pascal

```
TWebRequestFiles = class(TAbstractWebRequestFiles);
```

C++

```
class TWebRequestFiles : public TAbstractWebRequestFiles;
```

File

ReqFiles

Description

TWebRequestFiles manages a set of Web request files objects (TWebRequestFile (see page 6696) objects). Each Web request file object in the collection describes an uploaded file that was extracted from the current HTTP request object.

See Also

TWebRequestFile (see page 6696)

TWebRequest (see page 3358)

74.1.2.1 TWebRequestFiles Members

The following tables list the members exposed by TWebRequestFiles.

Protected Methods

	Name	Description
	GetCount (see page 6703)	Returns the value of the Count property.
	GetItem (see page 6703)	Returns the value of the Items property.

Public Methods

	Name	Description
	Add (see page 6701)	This is the overview for the Add method overload.
	Create (see page 6701)	Creates and initializes an instance of TWebRequestFiles (see page 6700).
	Destroy (see page 6701)	Frees an instance of TWebRequestFiles (see page 6700).

Public Properties

	Name	Description
	Count (see page 3294)	Indicates the number of uploaded files in the collection.
	Items (see page 3295)	Lists the uploaded files in the collection.

74.1.2.2 TWebRequestFiles.Create Constructor

Creates and initializes an instance of TWebRequestFiles (see page 6700).

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebRequestFiles();
```

Description

Call Create to instantiate TWebRequestFiles (see page 6700) at runtime. TWebRequestFiles (see page 6700) instances are created automatically by the Web request object to implement its Files property.

See Also

Files (see page 3379)

74.1.2.3 TWebRequestFiles.Destroy Destructor

Frees an instance of TWebRequestFiles (see page 6700).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestFiles();
```

Description

Use the Free method to delete an instance of TWebRequestFiles (see page 6700). Free checks for a nil reference before calling Destroy.

See Also

System::TObject::Free

74.1.2.4 TWebRequestFiles Methods

The methods of the TWebRequestFiles class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6703)	Returns the value of the Count property.
	GetItem (see page 6703)	Returns the value of the Items property.

Public Methods

	Name	Description
	Add (see page 6701)	This is the overview for the Add method overload.

74.1.2.4.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TWebRequestFiles.Add (AnsiString, AnsiString, AnsiString, Pointer, Integer) (see page 6702)	Adds a new Web request file object to the collection.
	TWebRequestFiles.Add (TWebRequestFile) (see page 6702)	Adds a new Web request file object to the collection.

74.1.2.4.1.1 TWebRequestFiles.Add Method (AnsiString, AnsiString, AnsiString, Pointer, Integer)

Adds a new Web request file object to the collection.

Pascal

```
procedure Add(const AName: AnsiString; const AFileName: AnsiString; const AContentType:
AnsiString; AContent: Pointer; AContentLength: Integer); overload;
procedure Add(AFile: TWebRequestFile); overload;
```

C++

```
__fastcall Add(const AnsiString AName, const AnsiString AFileName, const AnsiString
AContentType, void * AContent, int AContentLength);
__fastcall Add(TWebRequestFile AFile);
```

Description

Call Add to add a TWebRequestFile (see page 6696) object to the collection. Add inserts the new Web request file object at the end of the Items property array and increments the value of the Count property.

Use the first syntax to cause TWebRequestFiles (see page 6700) to create the Web request file object you are adding. Use the second syntax to add an existing TWebRequestFile (see page 6696) object. Using either syntax, TWebRequestFiles (see page 6700) assumes ownership of the new Web request file object and is responsible for freeing it.

AName is the name of the adapter field whose value the new file represents. It becomes the value of the FieldName property of the new TWebRequestFile (see page 6696) object.

AFileName is the name of the file to add. It becomes the value of the FileName property of the new TWebRequestFile (see page 6696) object.

AContentType describes how the file content is encoded. It becomes the value of the ContentType property of the new TWebRequestFile (see page 6696) object.

AContent is an array of bytes that is the content of the file. It is used to initialize the Stream property of the new TWebRequestFile (see page 6696) object.

AContentLength is the number of bytes in AContent.

AFile is an existing TWebRequestFile (see page 6696) object that describes the file you are adding.

See Also

Count

Items

FieldName

FileName

ContentType

Stream

74.1.2.4.2 TWebRequestFiles.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TAbstractWebRequestFile, GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count property.

See Also

Count (see page 3294)

74.1.2.4.3 TWebRequestFiles.GetItem Method

Returns the value of the Items property.

Pascal

```
function GetItem(I: Integer): TAbstractWebRequestFile; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFile GetItem(int I);
```

Description

GetItem is the protected read implementation of the Items property.

I is the index of the Web request file object to return.

In TAbstractWebRequestFile, GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count property.

74.1.3 TWebRequestFileStream Class

Class Hierarchy

Pascal

```
TWebRequestFileStream = class(TCustomMemoryStream);
```

C++

```
class TWebRequestFileStream : public TCustomMemoryStream;
```

File

ReqFiles

Description

This is class ReqFiles.TWebRequestFileStream.

74.1.3.1 TWebRequestFileStream Members

The following tables list the members exposed by TWebRequestFileStream.

Public Methods

	Name	Description
	Create (see page 6704)	This is Create, a member of class TWebRequestFileStream.
	Write (see page 6704)	Introduces a pure virtual method for writing to the stream.

74.1.3.2 TWebRequestFileStream.Create Constructor

Pascal

```
constructor Create(ABuffer: Pointer; Size: Integer);
```

C++

```
__fastcall TWebRequestFileStream(void * ABuffer, int Size);
```

Description

This is Create, a member of class TWebRequestFileStream.

74.1.3.3 TWebRequestFileStream Methods

The methods of the TWebRequestFileStream class are listed here.

Public Methods

	Name	Description
	Write (see page 6704)	Introduces a pure virtual method for writing to the stream.

74.1.3.3.1 TWebRequestFileStream.Write Method

Introduces a pure virtual method for writing to the stream.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Each descendant stream object defines a Write method that writes data to its particular storage medium (such as memory or a disk file). Write attempts to write up to Count bytes from Buffer, and returns the number of bytes actually written.

All the other data-writing methods of a stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

See Also

WriteBuffer

WriteComponent

75 ReqMulti Namespace

This is namespace ReqMulti.

Topics

Name	Description
Classes (see page 6706)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMultipartContentParser (see page 6706)	TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

75.1 Classes

The following table lists classes in this documentation.

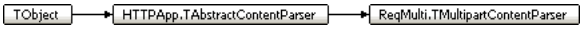
Classes

	Name	Description
	TMultipartContentParser (see page 6706)	TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

75.1.1 TMultipartContentParser Class

TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

Class Hierarchy



Pascal

```
TMultipartContentParser = class(TAbstractContentParser);
```

C++

```
class TMultipartContentParser : public TAbstractContentParser;
```

File

ReqMulti

Description

Web request objects use TMultipartContentParser object to parse the content of an HTTP request message from a multipart form when that request may contain uploaded files. The Web request uses this content parser to assign values for its ContentFields and Files properties.

TMultipartContentParser is only used when the current HTTP request object has a content type of 'multipart/form-data'. Multipart forms built using a WebSnap application automatically use this content type.

To allow your WebSnap application to use TMultipartContentParser in Delphi, add the ReqMulti (see page 6705) unit to the end of your project uses clause. In C++, include ReqMulti.hpp in the header of your project source file.

See Also

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TWebRequest (see page 3358)

TContentParser (see page 3301)

TWebRequest (see page 3358)

75.1.1.1 TMultipartContentParser Members

The following tables list the members exposed by TMultipartContentParser.

Protected Methods

	Name	Description
	BufferContent (see page 6708)	This is BufferContent, a member of class TMultipartContentParser.
	ExtractContentTypeFields (see page 6709)	This is ExtractContentTypeFields, a member of class TMultipartContentParser.
	ExtractMultipartContentFields (see page 6709)	This is ExtractMultipartContentFields, a member of class TMultipartContentParser.
V	GetContentFields (see page 6709)	Returns the named value strings that are included in the content of the current Web request.
V	GetFiles (see page 6709)	Returns a set of uploaded files that the parser extracts from the current Web request.
	ParseMultiPart (see page 6710)	This is ParseMultiPart, a member of class TMultipartContentParser.
	ParseMultiPartContent (see page 6710)	This is ParseMultiPartContent, a member of class TMultipartContentParser.
	ParseMultipartHeaders (see page 6710)	This is ParseMultipartHeaders, a member of class TMultipartContentParser.

Protected Properties

	Name	Description
R	WebRequest (see page 3258)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
V	CanParse (see page 6708)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
V	Create (see page 3256)	Creates and initializes a new TAbstractContentParser (see page 3255) object.
V	Destroy (see page 6707)	Frees the memory associated with the TMultipartContentParser (see page 6706) object.

75.1.1.2 TMultipartContentParser.Destroy Destructor

Frees the memory associated with the TMultipartContentParser (see page 6706) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultipartContentParser();
```

Description

Do not free content parser instances. The Web request object frees the content parser when it no longer needs it.

75.1.1.3 TMultipartContentParser Methods

The methods of the TMultipartContentParser class are listed here.

Protected Methods

	Name	Description
	BufferContent (see page 6708)	This is BufferContent, a member of class TMultipartContentParser.
	ExtractContentTypeFields (see page 6709)	This is ExtractContentTypeFields, a member of class TMultipartContentParser.
	ExtractMultipartContentFields (see page 6709)	This is ExtractMultipartContentFields, a member of class TMultipartContentParser.

	GetContentFields (see page 6709)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 6709)	Returns a set of uploaded files that the parser extracts from the current Web request.
	ParseMultiPart (see page 6710)	This is ParseMultiPart, a member of class TMultipartContentParser.
	ParseMultiPartContent (see page 6710)	This is ParseMultiPartContent, a member of class TMultipartContentParser.
	ParseMultipartHeaders (see page 6710)	This is ParseMultipartHeaders, a member of class TMultipartContentParser.

Public Methods

	Name	Description
	CanParse (see page 6708)	Indicates whether a specified request message contains information encoded in a way that this parser understands.

75.1.1.3.1 TMultipartContentParser.BufferContent Method

Pascal

```
procedure BufferContent;
```

C++

```
__fastcall BufferContent();
```

Description

This is BufferContent, a member of class TMultipartContentParser.

75.1.1.3.2 TMultipartContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a class method (Delphi) or static method (C++), which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered content parser until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TMultipartContentParser (see page 6706), CanParse returns true if AWebRequest has a content type of 'multipart/form-data'. If the Web request indicates some other content type, the Web request uses a different content parser class such as TContentParser (see page 3301).

See Also

ContentType (see page 3376)

TContentParser (see page 3301)

75.1.1.3.3 TMultipartContentParser.ExtractContentTypeFields Method

Pascal

```
procedure ExtractContentTypeFields(Strings: TStrings);
```

C++

```
__fastcall ExtractContentTypeFields(TStrings Strings);
```

Description

This is ExtractContentTypeFields, a member of class TMultipartContentParser.

75.1.1.3.4 TMultipartContentParser.ExtractMultipartContentFields Method

Pascal

```
procedure ExtractMultipartContentFields;
```

C++

```
__fastcall ExtractMultipartContentFields();
```

Description

This is ExtractMultipartContentFields, a member of class TMultipartContentParser.

75.1.1.3.5 TMultipartContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; override;
```

C++

```
virtual __fastcall TStrings GetContentFields();
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TAbstractContentParser, GetContentFields is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract name/value pairs from the current request and add them to the returned string list.

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 3375)

WebRequest (see page 3258)

ExtractContentFields (see page 3363)

75.1.1.3.6 TMultipartContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles();
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TAbstractContentParser, GetFiles is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract uploaded files from the current request and add them to the Web request files object.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

Files (see page 3379)

WebRequest (see page 3258)

TAbstractWebRequestFile (see page 3288)

75.1.1.3.7 TMultipartContentParser.ParseMultiPart Method

Pascal

```
procedure ParseMultiPart(Part: Pointer; Size: Integer);
```

C++

```
__fastcall ParseMultiPart(void * Part, int Size);
```

Description

This is ParseMultiPart, a member of class TMultipartContentParser.

75.1.1.3.8 TMultipartContentParser.ParseMultiPartContent Method

Pascal

```
procedure ParseMultiPartContent;
```

C++

```
__fastcall ParseMultiPartContent();
```

Description

This is ParseMultiPartContent, a member of class TMultipartContentParser.

75.1.1.3.9 TMultipartContentParser.ParseMultipartHeaders Method

Pascal

```
procedure ParseMultipartHeaders(Parser: THTTPParser; AContent: Pointer; AContentLength:
Integer);
```

C++

```
__fastcall ParseMultipartHeaders(THTTPParser Parser, void * AContent, int AContentLength);
```

Description

This is ParseMultipartHeaders, a member of class TMultipartContentParser.

76 Ribbon Namespace

This is namespace Ribbon.

Topics

Name	Description
Classes (see page 6715)	The following table lists classes in this documentation.
Interfaces (see page 6931)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 6936)	The following table lists structs, records, enums in this documentation.
Types (see page 6940)	The following table lists types in this documentation.

Classes

	Name	Description
	ERibbonException (see page 6715)	This is class Ribbon.ERibbonException.
	TApplicationMenu (see page 6716)	Contains properties associated with the Application (see page 2612) Menu (see page 6721).
	TCustomActionControlBar (see page 6721)	This is class Ribbon.TCustomActionControlBar.
	TCustomRibbon (see page 6724)	Base class for all ribbon controls.
	TCustomRibbonColorMap (see page 6767)	This is class Ribbon.TCustomRibbonColorMap.
	TCustomRibbonGroup (see page 6771)	Align and display all controls on ribbon.
	TCustomRibbonPage (see page 6790)	Holds groups of controls.
	TCustomRibbonQuickAccessPopupToolbar (see page 6804)	This is class Ribbon.TCustomRibbonQuickAccessPopupToolbar.
	TCustomRibbonQuickAccessToolbar (see page 6807)	Quck Access Toolbar base class.
	TCustomTabData (see page 6819)	Encapsulates all functions and status to manage scrolling of tabs and selected group.
	THelpButton (see page 6825)	Button to activate help.
	TOffice2007Button (see page 6826)	This is class Ribbon.TOffice2007Button.
	TQuickAccessToolbar (see page 6835)	Quick Access Toolbar appearance.
	TQuickAccessToolbarGroupItem (see page 6841)	Display Quick Access Toolbar commands to display entire Ribbon group.
	TQuickAccessToolbarGroups (see page 6843)	This is class Ribbon.TQuickAccessToolbarGroups.
	TRibbon (see page 6845)	Implement ribbon control.
	TRibbonCustomizeDlg (see page 6864)	This is class Ribbon.TRibbonCustomizeDlg.
	TRibbonGroup (see page 6865)	Control used to display the user selectable commands for the ribbon.
	TRibbonGroupCollapsedControl (see page 6871)	Control used to represent a collapsed group on a tab.
	TRibbonGroupDockForm (see page 6884)	Display a collapsed group.
	TRibbonGroupScrollButton (see page 6886)	This is class Ribbon.TRibbonGroupScrollButton.
	TRibbonLunaColorMap (see page 6890)	Color map class for Luna.

	TRibbonObsidianColorMap (see page 6896)	Color map class for Obsidian.
	TRibbonPage (see page 6901)	Tab page for ribbon control.
	TRibbonQuickAccessPopupToolbar (see page 6911)	This is class Ribbon.TRibbonQuickAccessPopupToolbar.
	TRibbonQuickAccessToolbar (see page 6911)	Quck Access Toolbar.
	TRibbonSilverColorMap (see page 6914)	Color map class for Silver.
	TRibbonTabItem (see page 6920)	This is class Ribbon.TRibbonTabItem.
	TRibbonTabs (see page 6925)	Collection of information about ribbon pages.

Interfaces

	Name	Description
	IRibbon (see page 6931)	Interface for ribbon control.
	IRibbonGroup (see page 6933)	Interface for any ribbon-specific group.
	IRibbonQuickAccessToolbar (see page 6934)	This is class Ribbon.IRibbonQuickAccessToolbar.

Structs, Records, Enums

	Name	Description
	TCommandType (see page 6936)	This is record Ribbon.TCommandType.
	TIconSize (see page 6936)	This is record Ribbon.TIconSize.
	TRibbonGroupVariant (see page 6937)	This is record Ribbon.TRibbonGroupVariant.
	TRibbonMetric (see page 6937)	This is record Ribbon.TRibbonMetric.
	TRibbonScrollButton (see page 6938)	This is record Ribbon.TRibbonScrollButton.
	TTabActiveScrollButton (see page 6938)	This is record Ribbon.TTabActiveScrollButton.
	TTabSeparatorState (see page 6939)	Position of Quick Access Toolbar.

Types

Name	Description
TCustomRibbonClass (see page 6940)	This is type Ribbon.TCustomRibbonClass.
TRecentItemClickEvent (see page 6940)	This is type Ribbon.TRecentItemClickEvent.
TRibbonScrollButtons (see page 6941)	This is type Ribbon.TRibbonScrollButtons.

76.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERibbonException (see page 6715)	This is class Ribbon.ERibbonException.
	TApplicationMenu (see page 6716)	Contains properties associated with the Application (see page 2612) Menu (see page 6721).
	TCustomActionControlBar (see page 6721)	This is class Ribbon.TCustomActionControlBar.
	TCustomRibbon (see page 6724)	Base class for all ribbon controls.
	TCustomRibbonColorMap (see page 6767)	This is class Ribbon.TCustomRibbonColorMap.
	TCustomRibbonGroup (see page 6771)	Align and display all controls on ribbon.
	TCustomRibbonPage (see page 6790)	Holds groups of controls.
	TCustomRibbonQuickAccessPopupToolbar (see page 6804)	This is class Ribbon.TCustomRibbonQuickAccessPopupToolbar.
	TCustomRibbonQuickAccessToolbar (see page 6807)	Quck Access Toolbar base class.
	TCustomTabData (see page 6819)	Encapsulates all functions and status to manage scrolling of tabs and selected group.
	THelpButton (see page 6825)	Button to activate help.
	TOffice2007Button (see page 6826)	This is class Ribbon.TOffice2007Button.
	TQuickAccessToolbar (see page 6835)	Quick Access Toolbar appearance.
	TQuickAccessToolbarGroupItem (see page 6841)	Display Quick Access Toolbar commands to display entire Ribbon (see page 6713) group.
	TQuickAccessToolbarGroups (see page 6843)	This is class Ribbon.TQuickAccessToolbarGroups.
	TRibbon (see page 6845)	Implement ribbon control.
	TRibbonCustomizeDlg (see page 6864)	This is class Ribbon.TRibbonCustomizeDlg.
	TRibbonGroup (see page 6865)	Control used to display the user selectable commands for the ribbon.
	TRibbonGroupCollapsedControl (see page 6871)	Control used to represent a collapsed group on a tab.
	TRibbonGroupDockForm (see page 6884)	Display a collapsed group.
	TRibbonGroupScrollButton (see page 6886)	This is class Ribbon.TRibbonGroupScrollButton.
	TRibbonLunaColorMap (see page 6890)	Color map class for Luna.
	TRibbonObsidianColorMap (see page 6896)	Color map class for Obsidian.
	TRibbonPage (see page 6901)	Tab page for ribbon control.
	TRibbonQuickAccessPopupToolbar (see page 6911)	This is class Ribbon.TRibbonQuickAccessPopupToolbar.
	TRibbonQuickAccessToolbar (see page 6911)	Quck Access Toolbar.
	TRibbonSilverColorMap (see page 6914)	Color map class for Silver.
	TRibbonTabItem (see page 6920)	This is class Ribbon.TRibbonTabItem.
	TRibbonTabs (see page 6925)	Collection of information about ribbon pages.

76.1.1 ERibbonException Class

Class Hierarchy

Pascal

```
ERibbonException = class(Exception);
```

C++

```
class ERibbonException : public Exception;
```

File

Ribbon

Description

This is class Ribbon.ERibbonException.

76.1.2 TApplicationMenu Class

Contains properties associated with the Application (see page 2612) Menu (see page 6721).

Class Hierarchy

Pascal

```
TApplicationMenu = class(TPersistent);
```

C++

```
class TApplicationMenu : public TPersistent;
```

File

Ribbon

Description

TApplicationMenu Contains properties associated with the Application (see page 2612) Menu (see page 6721) on a ribbon.

Use the properties to change the appearance and behavior of the Application (see page 2612) Menu (see page 6721) as displayed on the ribbon.

See Also

Caption (see page 6719)

Icon (see page 6720)

KeyTip (see page 6720)

76.1.2.1 TApplicationMenu Members

The following tables list the members exposed by TApplicationMenu.

Protected Methods

	Name	Description
	GetActionClientItem (see page 6718)	This is GetActionClientItem, a member of class TApplicationMenu.
	Initialize (see page 6719)	This is Initialize, a member of class TApplicationMenu.

Public Methods

	Name	Description
	Assign (see page 6718)	Copies the contents of another, similar object.
	Create (see page 6717)	This is Create, a member of class TApplicationMenu.

	Destroy (see page 6717)	Destroys the TPersistent instance and frees its memory.
	Show (see page 6719)	This is Show, a member of class TApplicationMenu.

Published Properties

	Name	Description
	Caption (see page 6719)	Text displayed above Application (see page 2612) Menu (see page 6721).
	CommandType (see page 6720)	This is CommandType, a member of class TApplicationMenu.
	Icon (see page 6720)	Icon to display on the Application (see page 2612) Button.
	IconSize (see page 6720)	This is IconSize, a member of class TApplicationMenu.
	KeyTip (see page 6720)	KeyTip displayed in KeyTip mode.
	Menu (see page 6721)	This is Menu, a member of class TApplicationMenu.
	ScreenTip (see page 6721)	This is ScreenTip, a member of class TApplicationMenu.

76.1.2.2 TApplicationMenu.Create Constructor

Pascal

```
constructor Create(ARibbon: TCustomRibbon); reintroduce;
```

C++

```
__fastcall TApplicationMenu(TCustomRibbon ARibbon);
```

Description

This is Create, a member of class TApplicationMenu.

76.1.2.3 TApplicationMenu.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TApplicationMenu();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free

System::TObject::Destroy

76.1.2.4 TApplicationMenu Methods

The methods of the TApplicationMenu class are listed here.

Protected Methods

	Name	Description
	GetActionClientItem (see page 6718)	This is GetActionClientItem, a member of class TApplicationMenu.
	Initialize (see page 6719)	This is Initialize, a member of class TApplicationMenu.

Public Methods

	Name	Description
	Assign (see page 6718)	Copies the contents of another, similar object.
	Show (see page 6719)	This is Show, a member of class TApplicationMenu.

76.1.2.4.1 TApplicationMenu.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

76.1.2.4.2 TApplicationMenu.GetActionClientItem Method

Pascal

```
function GetActionClientItem: TActionClientItem;
```

C++

```
__fastcall TActionClientItem GetActionClientItem();
```

Description

This is GetActionClientItem, a member of class TApplicationMenu.

76.1.2.4.3 TApplicationMenu.Initialize Method

Pascal

```
procedure Initialize(AManager: TCustomScreenTipsManager);
```

C++

```
__fastcall Initialize(TCustomScreenTipsManager AManager);
```

Description

This is Initialize, a member of class TApplicationMenu.

76.1.2.4.4 TApplicationMenu.Show Method

Pascal

```
procedure Show(const Visible: Boolean);
```

C++

```
__fastcall Show(const Boolean Visible);
```

Description

This is Show, a member of class TApplicationMenu.

76.1.2.5 TApplicationMenu Properties

The properties of the TApplicationMenu class are listed here.

Published Properties

	Name	Description
	Caption (see page 6719)	Text displayed above Application (see page 2612) Menu (see page 6721).
	CommandType (see page 6720)	This is CommandType, a member of class TApplicationMenu.
	Icon (see page 6720)	Icon to display on the Application (see page 2612) Button.
	IconSize (see page 6720)	This is IconSize, a member of class TApplicationMenu.
	KeyTip (see page 6720)	KeyTip displayed in KeyTip mode.
	Menu (see page 6721)	This is Menu, a member of class TApplicationMenu.
	ScreenTip (see page 6721)	This is ScreenTip, a member of class TApplicationMenu.

76.1.2.5.1 TApplicationMenu.Caption Property

Text displayed above Application (see page 2612) Menu (see page 6721).

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Caption is the text displayed above the right panel in the Application (see page 2612) Menu (see page 6721).

76.1.2.5.2 TApplicationMenu.CommandType Property

Pascal

```
property CommandType: TCommandType;
```

C++

```
__property TCommandType CommandType;
```

Description

This is CommandType, a member of class TApplicationMenu.

76.1.2.5.3 TApplicationMenu.Icon Property

Icon to display on the Application (see page 2612) Button.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Icon is the icon to display on the Application (see page 2612) Button. If no icon is assigned, the default application icon is used. The dimensions of the assigned icon are 24 x 24.

76.1.2.5.4 TApplicationMenu.IconSize Property

Pascal

```
property IconSize: TIconSize;
```

C++

```
__property TIconSize IconSize;
```

Description

This is IconSize, a member of class TApplicationMenu.

76.1.2.5.5 TApplicationMenu.KeyTip Property

KeyTip displayed in KeyTip mode.

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

KeyTip is the KeyTip that is displayed when the user has entered KeyTip mode.

76.1.2.5.6 TApplicationMenu.Menu Property

Pascal

```
property Menu: TRibbonApplicationMenuBar;
```

C++

```
__property TRibbonApplicationMenuBar Menu;
```

Description

This is Menu, a member of class TApplicationMenu.

76.1.2.5.7 TApplicationMenu.ScreenTip Property

Pascal

```
property ScreenTip: TScreenTipItem;
```

C++

```
__property TScreenTipItem ScreenTip;
```

Description

This is ScreenTip, a member of class TApplicationMenu.

76.1.3 TCustomActionControlBar Class

Class Hierarchy

Pascal

```
TCustomActionControlBar = class(TCustomActionToolbar);
```

C++

```
class TCustomActionControlBar : public TCustomActionToolbar;
```

File

Ribbon

Description

This is class Ribbon.TCustomActionControlBar.

76.1.3.1 TCustomActionControlBar Members

The following tables list the members exposed by TCustomActionControlBar.

Protected Methods

	Name	Description
	CMControlChange (see page 6722)	This is CMControlChange, a member of class TCustomActionControlBar.
V	CreateControls (see page 6723)	This is CreateControls, a member of class TCustomActionControlBar.
V	GetControlClass (see page 6723)	This is GetControlClass, a member of class TCustomActionControlBar.

Protected Properties

	Name	Description
	CreatingControls (see page 6723)	This is CreatingControls, a member of class TCustomActionControlBar.

Public Methods

	Name	Description
	Create (see page 6722)	This is Create, a member of class TCustomActionControlBar.
	CreateControl (see page 6722)	This is CreateControl, a member of class TCustomActionControlBar.

76.1.3.2 TCustomActionControlBar.Create Constructor

Pascal

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

C++

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

Description

This is Create, a member of class TCustomActionControlBar.

76.1.3.3 TCustomActionControlBar Methods

The methods of the TCustomActionControlBar class are listed here.

Protected Methods

	Name	Description
	CMControlChange (see page 6722)	This is CMControlChange, a member of class TCustomActionControlBar.
	CreateControls (see page 6723)	This is CreateControls, a member of class TCustomActionControlBar.
	GetControlClass (see page 6723)	This is GetControlClass, a member of class TCustomActionControlBar.

Public Methods

	Name	Description
	CreateControl (see page 6722)	This is CreateControl, a member of class TCustomActionControlBar.

76.1.3.3.1 TCustomActionControlBar.CMControlChange Method

Pascal

```
procedure CMControlChange(var Message: TCMControlChange); message CM_CONTROLCHANGE;
```

C++

```
__fastcall CMControlChange(TCMControlChange Message);
```

Description

This is CMControlChange, a member of class TCustomActionControlBar.

76.1.3.3.2 TCustomActionControlBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionControlBar.

76.1.3.3.3 TCustomActionControlBar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomActionControlBar.

76.1.3.3.4 TCustomActionControlBar.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TCustomActionControlBar.

76.1.3.4 TCustomActionControlBar Properties

The properties of the TCustomActionControlBar class are listed here.

Protected Properties

	Name	Description
	CreatingControls (see page 6723)	This is CreatingControls, a member of class TCustomActionControlBar.

76.1.3.4.1 TCustomActionControlBar.CreatingControls Property

Pascal

```
property CreatingControls: Boolean;
```

C++

```
__property Boolean CreatingControls;
```

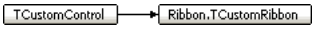
Description

This is CreatingControls, a member of class TCustomActionControlBar.

76.1.4 TCustomRibbon Class

Base class for all ribbon controls.

Class Hierarchy



Pascal

```
TCustomRibbon = class(TCustomControl);
```

C++

```
class TCustomRibbon : public TCustomControl;
```

File

Ribbon

Description

TCustomRibbon is the base class for all ribbon controls. A ribbon can hold controls, such as application menus, Quick Access Toolbars, and pages holding controls.

The following constants control ribbon appearance:

Value	Meaning
cRibbonHideWidth (see page 6731)	Parent width at which the Ribbon (see page 6713) must automatically hide.
cRibbonHideHeight (see page 6730)	Parent height at which the Ribbon (see page 6713) must automatically hide.
cRibbonQuickAccessToolbarTop (see page 6732)	Top position of the Quick Access Toolbar.
cRibbonQuickAccessToolbarLeft (see page 6732)	Left position of the Quick Access Toolbar.
cRibbonHeight (see page 6730)	Default height of the ribbon.
cRibbonQATHeight (see page 6732)	Height of the bottom aligned Quick Access Toolbar.
cRibbonUnthemedCaptionHeight (see page 6733)	Height of the caption section of the ribbon.
cRibbonFirstTabOffSet (see page 6730)	Position of the first tab that is displayed.
cRibbonTabSpacing (see page 6733)	Amount of spacing between tabs.
cRibbonTabHeight (see page 6732)	Height of the tabs.
cRibbonMinimizedHeight (see page 6731)	Minimized (see page 6763) height of the ribbon.

The following messages are used:

Value	Meaning
UM_DISPLAYKEYTIPS (see page 6733)	Message sent to display the KeyTips.
UM_CHECKSIZE (see page 6733)	Message sent after a `WMWindowPosChanged` (see page 6758) message.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TApplicationMenu (see page 6716)

TCustomRibbonGroup (see page 6771)

TCustomRibbonPage (see page 6790)

TQuickAccessToolbar (see page 6835)

TRibbonGroup (see page 6865)

TRibbonPage (see page 6901)

76.1.4.1 TCustomRibbon Members

The following tables list the members exposed by TCustomRibbon.

Protected Events

	Name	Description
	OnLeftTabScrollButtonClick (see page 6766)	This is OnLeftTabScrollButtonClick, a member of class TCustomRibbon.
	OnRightTabScrollButtonClick (see page 6767)	This is OnRightTabScrollButtonClick, a member of class TCustomRibbon.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 6736)	This is CMBidimodechanged, a member of class TCustomRibbon.
	CMDesignHitTest (see page 6737)	This is CMDesignHitTest, a member of class TCustomRibbon.
	CMFontchanged (see page 6737)	This is CMFontchanged, a member of class TCustomRibbon.
	CMMouseenter (see page 6737)	This is CMMouseenter, a member of class TCustomRibbon.
	CMMouseleave (see page 6737)	This is CMMouseleave, a member of class TCustomRibbon.
	CMTextchanged (see page 6738)	This is CMTextchanged, a member of class TCustomRibbon.
	CMThemeChanged (see page 6738)	This is CMThemeChanged, a member of class TCustomRibbon.
	ContextMenuPopupHandler (see page 6738)	This is ContextMenuPopupHandler, a member of class TCustomRibbon.
	CreateContextMenu (see page 6738)	This is CreateContextMenu, a member of class TCustomRibbon.
	CustomFrameWndProc (see page 6739)	This is CustomFrameWndProc, a member of class TCustomRibbon.
	DblClick (see page 6739)	OnDblClick event dispatcher.
	DefineProperties (see page 6739)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DisplayKeyTips (see page 6740)	This is DisplayKeyTips, a member of class TCustomRibbon.
	DoAddToQuickAccessToolbar (see page 6740)	This is DoAddToQuickAccessToolbar, a member of class TCustomRibbon.
	DoChangeQuickAccessToolbarPosition (see page 6740)	This is DoChangeQuickAccessToolbarPosition, a member of class TCustomRibbon.
	DoContextPopup (see page 6741)	OnContextPopup event dispatcher.
	DoCustomizeQuickAccessToolbar (see page 6741)	This is DoCustomizeQuickAccessToolbar, a member of class TCustomRibbon.
	DoGroupStyleChanged (see page 6741)	This is DoGroupStyleChanged, a member of class TCustomRibbon.
	DoHelpButtonClick (see page 6741)	This is DoHelpButtonClick, a member of class TCustomRibbon.
	DoMinimizeRibbon (see page 6742)	This is DoMinimizeRibbon, a member of class TCustomRibbon.
	DoMouseWheelDown (see page 6742)	OnMouseWheelDown event dispatcher.
	DoMouseWheelUp (see page 6743)	OnMouseWheelUp event dispatcher.
	DoRecentItemClick (see page 6743)	This is DoRecentItemClick, a member of class TCustomRibbon.
	DoScrollTabsLeft (see page 6743)	This is DoScrollTabsLeft, a member of class TCustomRibbon.
	DoScrollTabsRight (see page 6743)	This is DoScrollTabsRight, a member of class TCustomRibbon.
	DoTabChange (see page 6744)	This is DoTabChange, a member of class TCustomRibbon.

	DrawCaption (see page 6744)	This is DrawCaption, a member of class TCustomRibbon.
	DrawQuickAccessToolbar (see page 6744)	This is DrawQuickAccessToolbar, a member of class TCustomRibbon.
	DrawSystemButtons (see page 6744)	This is DrawSystemButtons, a member of class TCustomRibbon.
	DrawTab (see page 6745)	This is DrawTab, a member of class TCustomRibbon.
	GetContextMenuOptions (see page 6745)	This is GetContextMenuOptions, a member of class TCustomRibbon.
	GetHelpButtonRect (see page 6745)	This is GetHelpButtonRect, a member of class TCustomRibbon.
	GetRibbonHeight (see page 6746)	This is GetRibbonHeight, a member of class TCustomRibbon.
	GetTabAt (see page 6746)	This is GetTabAt, a member of class TCustomRibbon.
	GetTabRect (see page 6746)	This is GetTabRect, a member of class TCustomRibbon.
	GetTabWidth (see page 6747)	This is GetTabWidth, a member of class TCustomRibbon.
	HasApplicationMenu (see page 6747)	This is HasApplicationMenu, a member of class TCustomRibbon.
	HasQuickAccessToolbar (see page 6747)	This is HasQuickAccessToolbar, a member of class TCustomRibbon.
	Idle (see page 6747)	This is Idle, a member of class TCustomRibbon.
	KeyDown (see page 6749)	Respond to key press events.
	Loaded (see page 6749)	Initializes the control after it is loaded from a stream.
	MouseDown (see page 6750)	OnMouseDown event dispatcher.
	MouseInSystemMenu (see page 6750)	This is MouseInSystemMenu, a member of class TCustomRibbon.
	MouseMove (see page 6750)	Respond to mouse moving over control area..
	MouseUp (see page 6751)	OnMouseUp event dispatcher.
	Notification (see page 6752)	Responds to notifications that components are being created or destroyed.
	Paint (see page 6752)	Renders the image of a custom control.
	RebuildRibbonMetrics (see page 6753)	This is RebuildRibbonMetrics, a member of class TCustomRibbon.
	Resize (see page 6754)	Respond to control resize.
	SetParent (see page 6754)	Sets the parent of the control.
	UMCheckResize (see page 6756)	This is UMCheckResize, a member of class TCustomRibbon.
	UMDisplayKeyTips (see page 6756)	This is UMDisplayKeyTips, a member of class TCustomRibbon.
	UpdateQATPosition (see page 6757)	This is UpdateQATPosition, a member of class TCustomRibbon.
	UpdateSystemMenuInfo (see page 6757)	This is UpdateSystemMenuInfo, a member of class TCustomRibbon.
	ValidTab (see page 6757)	This is ValidTab, a member of class TCustomRibbon.
	WMEraseBkgnd (see page 6757)	This is WMEraseBkgnd, a member of class TCustomRibbon.
	WMGetDlgCode (see page 6758)	This is WMGetDlgCode, a member of class TCustomRibbon.
	WMKillFocus (see page 6758)	This is WMKillFocus, a member of class TCustomRibbon.
	WMSettingChange (see page 6758)	This is WMSettingChange, a member of class TCustomRibbon.
	WMWindowPosChanged (see page 6758)	This is WMWindowPosChanged, a member of class TCustomRibbon.

Protected Properties

	Name	Description
	AllowKeyboardNavigation (see page 6760)	This is AllowKeyboardNavigation, a member of class TCustomRibbon.
	HideRibbon (see page 6761)	This is HideRibbon, a member of class TCustomRibbon.
	HotTabIndex (see page 6762)	This is HotTabIndex, a member of class TCustomRibbon.
	KeyTipsVisible (see page 6762)	This is KeyTipsVisible, a member of class TCustomRibbon.
	Menu (see page 6762)	This is Menu, a member of class TCustomRibbon.
	ScaleFactor (see page 6763)	This is ScaleFactor, a member of class TCustomRibbon.

Public Constants

	Name	Description
	cRibbonFirstTabOffSet (see page 6730)	This is cRibbonFirstTabOffSet, a member of class TCustomRibbon.
	cRibbonGroupCaptionHeight (see page 6730)	This is cRibbonGroupCaptionHeight, a member of class TCustomRibbon.
	cRibbonGroupHeight (see page 6730)	This is cRibbonGroupHeight, a member of class TCustomRibbon.
	cRibbonHeight (see page 6730)	This is cRibbonHeight, a member of class TCustomRibbon.
	cRibbonHideHeight (see page 6730)	This is cRibbonHideHeight, a member of class TCustomRibbon.
	cRibbonHideWidth (see page 6731)	This is cRibbonHideWidth, a member of class TCustomRibbon.
	cRibbonMinimizedHeight (see page 6731)	This is cRibbonMinimizedHeight, a member of class TCustomRibbon.
	cRibbonMinimumCaptionWidth (see page 6731)	This is cRibbonMinimumCaptionWidth, a member of class TCustomRibbon.
	cRibbonPageHeight (see page 6731)	This is cRibbonPageHeight, a member of class TCustomRibbon.

	cRibbonQATHeight (see page 6732)	This is cRibbonQATHeight, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarLeft (see page 6732)	This is cRibbonQuickAccessToolbarLeft, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarTop (see page 6732)	This is cRibbonQuickAccessToolbarTop, a member of class TCustomRibbon.
	cRibbonTabHeight (see page 6732)	This is cRibbonTabHeight, a member of class TCustomRibbon.
	cRibbonTabScrollButtonWidth (see page 6732)	This is cRibbonTabScrollButtonWidth, a member of class TCustomRibbon.
	cRibbonTabSpacing (see page 6733)	This is cRibbonTabSpacing, a member of class TCustomRibbon.
	cRibbonUnthemedCaptionHeight (see page 6733)	This is cRibbonUnthemedCaptionHeight, a member of class TCustomRibbon.
	UM_CHECKSIZE (see page 6733)	This is UM_CHECKSIZE, a member of class TCustomRibbon.
	UM_DISPLAYKEYTIPS (see page 6733)	This is UM_DISPLAYKEYTIPS, a member of class TCustomRibbon.

Public Events

	Name	Description
	OnHelpButtonClick (see page 6766)	This is OnHelpButtonClick, a member of class TCustomRibbon.
	OnRecentItemClick (see page 6766)	This is OnRecentItemClick, a member of class TCustomRibbon.
	OnTabChange (see page 6767)	This is OnTabChange, a member of class TCustomRibbon.

Public Methods

	Name	Description
	AddRecentItem (see page 6735)	Add new entry into Recent Items list.
	AddTab (see page 6736)	This is AddTab, a member of class TCustomRibbon.
	BeginUpdate (see page 6736)	This is BeginUpdate, a member of class TCustomRibbon.
	CheckRibbonSize (see page 6736)	This is CheckRibbonSize, a member of class TCustomRibbon.
	ClearRecentItems (see page 6736)	This is ClearRecentItems, a member of class TCustomRibbon.
V	Create (see page 6728)	Creates an instance of TCustomControl.
	CustomFrameSupported (see page 6738)	This is CustomFrameSupported, a member of class TCustomRibbon.
	DeleteTab (see page 6740)	Delete tab at specified Index.
V	Destroy (see page 6728)	Destroys an instance of TCustomControl.
	DoMouseWheelEx (see page 6742)	This is DoMouseWheelEx, a member of class TCustomRibbon.
	EndUpdate (see page 6745)	This is EndUpdate, a member of class TCustomRibbon.
V	GetIcon (see page 6745)	Return Icon used by Application (see page 2612) Menu (see page 6762) Button.
	GetRibbonMetric (see page 6746)	This is GetRibbonMetric, a member of class TCustomRibbon.
	InvalidateHeaderArea (see page 6747)	Invalidate header section of Ribbon (see page 6713).
	InvalidateScrollButtons (see page 6748)	This is InvalidateScrollButtons, a member of class TCustomRibbon.
	InvalidateSystemMenuArea (see page 6748)	This is InvalidateSystemMenuArea, a member of class TCustomRibbon.
	InvalidateTabArea (see page 6748)	Invalidate area that displays tabs.
	MoveGroup (see page 6751)	Move group to new page.
	RemoveAllTabs (see page 6753)	Remove all Tabs (see page 6764) from Ribbon (see page 6713).
	RemoveRecentItem (see page 6753)	This is RemoveRecentItem, a member of class TCustomRibbon.
	RemoveTab (see page 6753)	Remove tab with specified caption.
V	SetBounds (see page 6754)	Sets the windowed control's boundary properties all at once.
	TabHandledKeyTips (see page 6755)	Validate **KeyTip** input when **KeyTip**s State is **ktsTabs**.
	TabHideKeyTips (see page 6755)	Hide K**eyTip**s.
	TabShowKeyTips (see page 6756)	Display **KeyTip**s for **ktsTabs** State.

Public Nested Types

Name	Description
TTabChangeEvent (see page 6765)	This is nested type Ribbon.TCustomRibbon.TTabChangeEvent.

Public Properties

	Name	Description
	ActionManager (see page 6759)	This is ActionManager, a member of class TCustomRibbon.
	Active (see page 6759)	Indicates if form that ribbon is placed on is active form.
	ActivePage (see page 6760)	Page currently displayed by ribbon control.

	ApplicationMenu (see page 6760)	Change appearance of ribbon's associated TRibbonApplicationMenuBar (see page 7302) control.
	ColorMap (see page 6761)	This is ColorMap, a member of class TCustomRibbon.
	DocumentName (see page 6761)	Text in Ribbon (see page 6713) caption.
	HideTabs (see page 6762)	Shows or hides Tab portion of Ribbon (see page 6713).
	Minimized (see page 6763)	Show Ribbon (see page 6713) in minimized state where tabs can be selected but are not visible.
	QuickAccessToolbar (see page 6763)	Properties associated with TQuickAccessToolbar (see page 6835).
	ScreenTips (see page 6763)	This is ScreenTips, a member of class TCustomRibbon.
	ShowHelpButton (see page 6764)	Determines whether ribbon displays Help Button.
	Style (see page 6764)	This is Style, a member of class TCustomRibbon.
	TabIndex (see page 6764)	Change the active page of ribbon.
	Tabs (see page 6764)	Collection of TRibbonTabs (see page 6925) visible on ribbon.
	UseCustomFrame (see page 6765)	Indicates whether to use a custom frame for ribbon controls.

76.1.4.2 TCustomRibbon.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

76.1.4.3 TCustomRibbon.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRibbon();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

76.1.4.4 TCustomRibbon Constants

The constants of the TCustomRibbon class are listed here.

Public Constants

	Name	Description
	cRibbonFirstTabOffSet (see page 6730)	This is cRibbonFirstTabOffSet, a member of class TCustomRibbon.
	cRibbonGroupCaptionHeight (see page 6730)	This is cRibbonGroupCaptionHeight, a member of class TCustomRibbon.
	cRibbonGroupHeight (see page 6730)	This is cRibbonGroupHeight, a member of class TCustomRibbon.
	cRibbonHeight (see page 6730)	This is cRibbonHeight, a member of class TCustomRibbon.
	cRibbonHideHeight (see page 6730)	This is cRibbonHideHeight, a member of class TCustomRibbon.
	cRibbonHideWidth (see page 6731)	This is cRibbonHideWidth, a member of class TCustomRibbon.
	cRibbonMinimizedHeight (see page 6731)	This is cRibbonMinimizedHeight, a member of class TCustomRibbon.
	cRibbonMinimumCaptionWidth (see page 6731)	This is cRibbonMinimumCaptionWidth, a member of class TCustomRibbon.
	cRibbonPageHeight (see page 6731)	This is cRibbonPageHeight, a member of class TCustomRibbon.
	cRibbonQATHeight (see page 6732)	This is cRibbonQATHeight, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarLeft (see page 6732)	This is cRibbonQuickAccessToolbarLeft, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarTop (see page 6732)	This is cRibbonQuickAccessToolbarTop, a member of class TCustomRibbon.
	cRibbonTabHeight (see page 6732)	This is cRibbonTabHeight, a member of class TCustomRibbon.
	cRibbonTabScrollButtonWidth (see page 6732)	This is cRibbonTabScrollButtonWidth, a member of class TCustomRibbon.
	cRibbonTabSpacing (see page 6733)	This is cRibbonTabSpacing, a member of class TCustomRibbon.
	cRibbonUnthemedCaptionHeight (see page 6733)	This is cRibbonUnthemedCaptionHeight, a member of class TCustomRibbon.
	UM_CHECKSIZE (see page 6733)	This is UM_CHECKSIZE, a member of class TCustomRibbon.
	UM_DISPLAYKEYTIPS (see page 6733)	This is UM_DISPLAYKEYTIPS, a member of class TCustomRibbon.

76.1.4.4.1 TCustomRibbon.cRibbonFirstTabOffSet Constant

Pascal

```
const cRibbonFirstTabOffSet = 47;
```

C++

```
cRibbonFirstTabOffSet = 47;
```

Description

This is cRibbonFirstTabOffSet, a member of class TCustomRibbon.

76.1.4.4.2 TCustomRibbon.cRibbonGroupCaptionHeight Constant

Pascal

```
const cRibbonGroupCaptionHeight = 16;
```

C++

```
cRibbonGroupCaptionHeight = 16;
```

Description

This is cRibbonGroupCaptionHeight, a member of class TCustomRibbon.

76.1.4.4.3 TCustomRibbon.cRibbonGroupHeight Constant

Pascal

```
const cRibbonGroupHeight = 86;
```

C++

```
cRibbonGroupHeight = 86;
```

Description

This is cRibbonGroupHeight, a member of class TCustomRibbon.

76.1.4.4.4 TCustomRibbon.cRibbonHeight Constant

Pascal

```
const cRibbonHeight = 117;
```

C++

```
cRibbonHeight = 117;
```

Description

This is cRibbonHeight, a member of class TCustomRibbon.

76.1.4.4.5 TCustomRibbon.cRibbonHideHeight Constant

Pascal

```
const cRibbonHideHeight = 250;
```

C++

```
cRibbonHideHeight = 250;
```

Description

This is cRibbonHideHeight, a member of class TCustomRibbon.

76.1.4.4.6 TCustomRibbon.cRibbonHideWidth Constant

Pascal

```
const cRibbonHideWidth = 300;
```

C++

```
cRibbonHideWidth = 300;
```

Description

This is cRibbonHideWidth, a member of class TCustomRibbon.

76.1.4.4.7 TCustomRibbon.cRibbonMinimizedHeight Constant

Pascal

```
const cRibbonMinimizedHeight = 27;
```

C++

```
cRibbonMinimizedHeight = 27;
```

Description

This is cRibbonMinimizedHeight, a member of class TCustomRibbon.

76.1.4.4.8 TCustomRibbon.cRibbonMinimumCaptionWidth Constant

Pascal

```
const cRibbonMinimumCaptionWidth = 50;
```

C++

```
cRibbonMinimumCaptionWidth = 50;
```

Description

This is cRibbonMinimumCaptionWidth, a member of class TCustomRibbon.

76.1.4.4.9 TCustomRibbon.cRibbonPageHeight Constant

Pascal

```
const cRibbonPageHeight = 93;
```

C++

```
cRibbonPageHeight = 93;
```

Description

This is cRibbonPageHeight, a member of class TCustomRibbon.

76.1.4.4.10 TCustomRibbon.cRibbonQATHeight Constant

Pascal

```
const cRibbonQATHeight = 26;
```

C++

```
cRibbonQATHeight = 26;
```

Description

This is cRibbonQATHeight, a member of class TCustomRibbon.

76.1.4.4.11 TCustomRibbon.cRibbonQuickAccessToolbarLeft Constant

Pascal

```
const cRibbonQuickAccessToolbarLeft = 34;
```

C++

```
cRibbonQuickAccessToolbarLeft = 34;
```

Description

This is cRibbonQuickAccessToolbarLeft, a member of class TCustomRibbon.

76.1.4.4.12 TCustomRibbon.cRibbonQuickAccessToolbarTop Constant

Pascal

```
const cRibbonQuickAccessToolbarTop = 5;
```

C++

```
cRibbonQuickAccessToolbarTop = 5;
```

Description

This is cRibbonQuickAccessToolbarTop, a member of class TCustomRibbon.

76.1.4.4.13 TCustomRibbon.cRibbonTabHeight Constant

Pascal

```
const cRibbonTabHeight = 23;
```

C++

```
cRibbonTabHeight = 23;
```

Description

This is cRibbonTabHeight, a member of class TCustomRibbon.

76.1.4.4.14 TCustomRibbon.cRibbonTabScrollButtonWidth Constant

Pascal

```
const cRibbonTabScrollButtonWidth = 12;
```

C++

```
cRibbonTabScrollButtonWidth = 12;
```

Description

This is cRibbonTabScrollButtonWidth, a member of class TCustomRibbon.

76.1.4.4.15 TCustomRibbon.cRibbonTabSpacing Constant

Pascal

```
const cRibbonTabSpacing = 6;
```

C++

```
cRibbonTabSpacing = 6;
```

Description

This is cRibbonTabSpacing, a member of class TCustomRibbon.

76.1.4.4.16 TCustomRibbon.cRibbonUnthemedCaptionHeight Constant

Pascal

```
const cRibbonUnthemedCaptionHeight = 30;
```

C++

```
cRibbonUnthemedCaptionHeight = 30;
```

Description

This is cRibbonUnthemedCaptionHeight, a member of class TCustomRibbon.

76.1.4.4.17 TCustomRibbon.UM_CHECKSIZE Constant

Pascal

```
const UM_CHECKSIZE = WM_USER + 2;
```

C++

```
UM_CHECKSIZE = WM_USER + 2;
```

Description

This is UM_CHECKSIZE, a member of class TCustomRibbon.

76.1.4.4.18 TCustomRibbon.UM_DISPLAYKEYTIPS Constant

Pascal

```
const UM_DISPLAYKEYTIPS = WM_USER + 1;
```

C++

```
UM_DISPLAYKEYTIPS = WM_USER + 1;
```

Description

This is UM_DISPLAYKEYTIPS, a member of class TCustomRibbon.

76.1.4.5 TCustomRibbon Methods

The methods of the TCustomRibbon class are listed here.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 6736)	This is CMBidimodechanged, a member of class TCustomRibbon.
	CMDesignHitTest (see page 6737)	This is CMDesignHitTest, a member of class TCustomRibbon.
	CMFontchanged (see page 6737)	This is CMFontchanged, a member of class TCustomRibbon.
	CMMouseenter (see page 6737)	This is CMMouseenter, a member of class TCustomRibbon.
	CMMouseleave (see page 6737)	This is CMMouseleave, a member of class TCustomRibbon.
	CMTextchanged (see page 6738)	This is CMTextchanged, a member of class TCustomRibbon.
	CMThemeChanged (see page 6738)	This is CMThemeChanged, a member of class TCustomRibbon.
V	ContextMenuPopupHandler (see page 6738)	This is ContextMenuPopupHandler, a member of class TCustomRibbon.
V	CreateContextMenu (see page 6738)	This is CreateContextMenu, a member of class TCustomRibbon.
V	CustomFrameWndProc (see page 6739)	This is CustomFrameWndProc, a member of class TCustomRibbon.
V	DblClick (see page 6739)	OnDblClick event dispatcher.
V	DefineProperties (see page 6739)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DisplayKeyTips (see page 6740)	This is DisplayKeyTips, a member of class TCustomRibbon.
V	DoAddToQuickAccessToolbar (see page 6740)	This is DoAddToQuickAccessToolbar, a member of class TCustomRibbon.
V	DoChangeQuickAccessToolbarPosition (see page 6740)	This is DoChangeQuickAccessToolbarPosition, a member of class TCustomRibbon.
V	DoContextPopup (see page 6741)	OnContextPopup event dispatcher.
V	DoCustomizeQuickAccessToolbar (see page 6741)	This is DoCustomizeQuickAccessToolbar, a member of class TCustomRibbon.
V	DoGroupStyleChanged (see page 6741)	This is DoGroupStyleChanged, a member of class TCustomRibbon.
V	DoHelpButtonClick (see page 6741)	This is DoHelpButtonClick, a member of class TCustomRibbon.
V	DoMinimizeRibbon (see page 6742)	This is DoMinimizeRibbon, a member of class TCustomRibbon.
V	DoMouseWheelDown (see page 6742)	OnMouseWheelDown event dispatcher.
V	DoMouseWheelUp (see page 6743)	OnMouseWheelUp event dispatcher.
V	DoRecentItemClick (see page 6743)	This is DoRecentItemClick, a member of class TCustomRibbon.
V	DoScrollTabsLeft (see page 6743)	This is DoScrollTabsLeft, a member of class TCustomRibbon.
V	DoScrollTabsRight (see page 6743)	This is DoScrollTabsRight, a member of class TCustomRibbon.
V	DoTabChange (see page 6744)	This is DoTabChange, a member of class TCustomRibbon.
V	DrawCaption (see page 6744)	This is DrawCaption, a member of class TCustomRibbon.
V	DrawQuickAccessToolbar (see page 6744)	This is DrawQuickAccessToolbar, a member of class TCustomRibbon.
	DrawSystemButtons (see page 6744)	This is DrawSystemButtons, a member of class TCustomRibbon.
V	DrawTab (see page 6745)	This is DrawTab, a member of class TCustomRibbon.
	GetContextMenuOptions (see page 6745)	This is GetContextMenuOptions, a member of class TCustomRibbon.
V	GetHelpButtonRect (see page 6745)	This is GetHelpButtonRect, a member of class TCustomRibbon.
	GetRibbonHeight (see page 6746)	This is GetRibbonHeight, a member of class TCustomRibbon.
V	GetTabAt (see page 6746)	This is GetTabAt, a member of class TCustomRibbon.
V	GetTabRect (see page 6746)	This is GetTabRect, a member of class TCustomRibbon.
V	GetTabWidth (see page 6747)	This is GetTabWidth, a member of class TCustomRibbon.
	HasApplicationMenu (see page 6747)	This is HasApplicationMenu, a member of class TCustomRibbon.
	HasQuickAccessToolbar (see page 6747)	This is HasQuickAccessToolbar, a member of class TCustomRibbon.
	Idle (see page 6747)	This is Idle, a member of class TCustomRibbon.
V	KeyDown (see page 6749)	Respond to key press events.
V	Loaded (see page 6749)	Initializes the control after it is loaded from a stream.
V	MouseDown (see page 6750)	OnMouseDown event dispatcher.
	MouseInSystemMenu (see page 6750)	This is MouseInSystemMenu, a member of class TCustomRibbon.
V	MouseMove (see page 6750)	Respond to mouse moving over control area..
V	MouseUp (see page 6751)	OnMouseUp event dispatcher.
V	Notification (see page 6752)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 6752)	Renders the image of a custom control.
	RebuildRibbonMetrics (see page 6753)	This is RebuildRibbonMetrics, a member of class TCustomRibbon.
V	Resize (see page 6754)	Respond to control resize.
V	SetParent (see page 6754)	Sets the parent of the control.

	UMCheckResize (see page 6756)	This is UMCheckResize, a member of class TCustomRibbon.
	UMDisplayKeyTips (see page 6756)	This is UMDisplayKeyTips, a member of class TCustomRibbon.
	UpdateQATPosition (see page 6757)	This is UpdateQATPosition, a member of class TCustomRibbon.
	UpdateSystemMenuInfo (see page 6757)	This is UpdateSystemMenuInfo, a member of class TCustomRibbon.
	ValidTab (see page 6757)	This is ValidTab, a member of class TCustomRibbon.
	WMEraseBkgnd (see page 6757)	This is WMEraseBkgnd, a member of class TCustomRibbon.
	WMGetDlgCode (see page 6758)	This is WMGetDlgCode, a member of class TCustomRibbon.
	WMKillFocus (see page 6758)	This is WMKillFocus, a member of class TCustomRibbon.
	WMSettingChange (see page 6758)	This is WMSettingChange, a member of class TCustomRibbon.
	WMWindowPosChanged (see page 6758)	This is WMWindowPosChanged, a member of class TCustomRibbon.

Public Methods

	Name	Description
	AddRecentItem (see page 6735)	Add new entry into Recent Items list.
	AddTab (see page 6736)	This is AddTab, a member of class TCustomRibbon.
	BeginUpdate (see page 6736)	This is BeginUpdate, a member of class TCustomRibbon.
	CheckRibbonSize (see page 6736)	This is CheckRibbonSize, a member of class TCustomRibbon.
	ClearRecentItems (see page 6736)	This is ClearRecentItems, a member of class TCustomRibbon.
	CustomFrameSupported (see page 6738)	This is CustomFrameSupported, a member of class TCustomRibbon.
	DeleteTab (see page 6740)	Delete tab at specified Index.
	DoMouseWheelEx (see page 6742)	This is DoMouseWheelEx, a member of class TCustomRibbon.
	EndUpdate (see page 6745)	This is EndUpdate, a member of class TCustomRibbon.
	GetIcon (see page 6745)	Return Icon used by Application (see page 2612) Menu (see page 6762) Button.
	GetRibbonMetric (see page 6746)	This is GetRibbonMetric, a member of class TCustomRibbon.
	InvalidateHeaderArea (see page 6747)	Invalidate header section of Ribbon (see page 6713).
	InvalidateScrollButtons (see page 6748)	This is InvalidateScrollButtons, a member of class TCustomRibbon.
	InvalidateSystemMenuArea (see page 6748)	This is InvalidateSystemMenuArea, a member of class TCustomRibbon.
	InvalidateTabArea (see page 6748)	Invalidate area that displays tabs.
	MoveGroup (see page 6751)	Move group to new page.
	RemoveAllTabs (see page 6753)	Remove all Tabs (see page 6764) from Ribbon (see page 6713).
	RemoveRecentItem (see page 6753)	This is RemoveRecentItem, a member of class TCustomRibbon.
	RemoveTab (see page 6753)	Remove tab with specified caption.
	SetBounds (see page 6754)	Sets the windowed control's boundary properties all at once.
	TabHandledKeyTips (see page 6755)	Validate **KeyTip** input when **KeyTip**s State is **ktsTabs**.
	TabHideKeyTips (see page 6755)	Hide K**eyTip**s.
	TabShowKeyTips (see page 6756)	Display **KeyTip**s for **ktsTabs** State.

76.1.4.5.1 TCustomRibbon.AddRecentItem Method

Add new entry into Recent Items list.

Pascal

```
function AddRecentItem(const FileName: string): Integer;
```

C++

```
__fastcall int AddRecentItem(const UnicodeString FileName);
```

Description

Call AddRecentItem to add a new entry into the Recent Items list to the right of the Application (see page 2612) Menu (see page 6762). The new entry is automatically added to the top of the Recent items list, and if an entry with the same filename already exists, it is deleted.

This method should only be called when the command type for the application menu is set to **ctRecent**.

76.1.4.5.2 TCustomRibbon.AddTab Method

Pascal

```
function AddTab(const APage: TCustomRibbonPage): TRibbonTabItem;
```

C++

```
__fastcall TRibbonTabItem AddTab(const TCustomRibbonPage APage);
```

Description

This is AddTab, a member of class TCustomRibbon.

76.1.4.5.3 TCustomRibbon.BeginUpdate Method

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

This is BeginUpdate, a member of class TCustomRibbon.

76.1.4.5.4 TCustomRibbon.CheckRibbonSize Method

Pascal

```
procedure CheckRibbonSize;
```

C++

```
__fastcall CheckRibbonSize();
```

Description

This is CheckRibbonSize, a member of class TCustomRibbon.

76.1.4.5.5 TCustomRibbon.ClearRecentItems Method

Pascal

```
procedure ClearRecentItems;
```

C++

```
__fastcall ClearRecentItems();
```

Description

This is ClearRecentItems, a member of class TCustomRibbon.

76.1.4.5.6 TCustomRibbon.CMBidimodechanged Method

Pascal

```
procedure CMBidimodechanged(var Message: TMessage); message CM_BIDIMODECHANGED;
```

C++

```
__fastcall CMBidimodechanged(TMessage Message);
```

Description

This is CMBidimodechanged, a member of class TCustomRibbon.

76.1.4.5.7 TCustomRibbon.CMDesignHitTest Method

Pascal

```
procedure CMDesignHitTest(var Message: TCMDesignHitTest); message CM_DESIGNHITTEST;
```

C++

```
__fastcall CMDesignHitTest(TCMDesignHitTest Message);
```

Description

This is CMDesignHitTest, a member of class TCustomRibbon.

76.1.4.5.8 TCustomRibbon.CMFontchanged Method

Pascal

```
procedure CMFontchanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontchanged(TMessage Message);
```

Description

This is CMFontchanged, a member of class TCustomRibbon.

76.1.4.5.9 TCustomRibbon.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TCustomRibbon.

76.1.4.5.10 TCustomRibbon.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TCustomRibbon.

76.1.4.5.11 TCustomRibbon.CMTextchanged Method

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This is CMTextchanged, a member of class TCustomRibbon.

76.1.4.5.12 TCustomRibbon.CMThemeChanged Method

Pascal

```
procedure CMThemeChanged(var Message: TMessage); message CM_THEMECHANGED;
```

C++

```
__fastcall CMThemeChanged(TMessage Message);
```

Description

This is CMThemeChanged, a member of class TCustomRibbon.

76.1.4.5.13 TCustomRibbon.ContextMenuPopupHandler Method

Pascal

```
procedure ContextMenuPopupHandler(Sender: TObject); virtual;
```

C++

```
virtual __fastcall ContextMenuPopupHandler(TObject * Sender);
```

Description

This is ContextMenuPopupHandler, a member of class TCustomRibbon.

76.1.4.5.14 TCustomRibbon.CreateContextMenu Method

Pascal

```
procedure CreateContextMenu; virtual;
```

C++

```
virtual __fastcall CreateContextMenu();
```

Description

This is CreateContextMenu, a member of class TCustomRibbon.

76.1.4.5.15 TCustomRibbon.CustomFrameSupported Method

Pascal

```
function CustomFrameSupported: Boolean;
```

C++

```
__fastcall Boolean CustomFrameSupported();
```

Description

This is CustomFrameSupported, a member of class TCustomRibbon.

76.1.4.5.16 TCustomRibbon.CustomFrameWndProc Method

Pascal

```
procedure CustomFrameWndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall CustomFrameWndProc(TMessage Message);
```

Description

This is CustomFrameWndProc, a member of class TCustomRibbon.

76.1.4.5.17 TCustomRibbon.DblClick Method

OnDblClick event dispatcher.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click

OnDblClick

76.1.4.5.18 TCustomRibbon.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty

DefineProperty

DefineProperties

IsControl

76.1.4.5.19 TCustomRibbon.DeleteTab Method

Delete tab at specified Index.

Pascal

```
procedure DeleteTab(const Index: Integer);
```

C++

```
__fastcall DeleteTab(const int Index);
```

Description

DeleteTab deletes the tab at the specified Index. The Page and any associated Groups are also deleted

76.1.4.5.20 TCustomRibbon.DisplayKeyTips Method

Pascal

```
procedure DisplayKeyTips(const State: TKeyTipState); virtual;
```

C++

```
virtual __fastcall DisplayKeyTips(const TKeyTipState State);
```

Description

This is DisplayKeyTips, a member of class TCustomRibbon.

76.1.4.5.21 TCustomRibbon.DoAddToQuickAccessToolbar Method

Pascal

```
procedure DoAddToQuickAccessToolbar; virtual;
```

C++

```
virtual __fastcall DoAddToQuickAccessToolbar();
```

Description

This is DoAddToQuickAccessToolbar, a member of class TCustomRibbon.

76.1.4.5.22 TCustomRibbon.DoChangeQuickAccessToolbarPosition Method

Pascal

```
procedure DoChangeQuickAccessToolbarPosition; virtual;
```

C++

```
virtual __fastcall DoChangeQuickAccessToolbarPosition();
```

Description

This is DoChangeQuickAccessToolbarPosition, a member of class TCustomRibbon.

76.1.4.5.23 TCustomRibbon.DoContextPopup Method

OnContextPopup event dispatcher.

Pascal

```
procedure DoContextPopup(MousePos: TPoint; var Handled: Boolean); override;
```

C++

```
virtual __fastcall DoContextPopup(TPoint MousePos, Boolean Handled);
```

Description

OnContextPopup calls any event handler attached to the OnContextPopup event. When creating a component that descends from TControl, override OnContextPopup to provide other responses in addition to the inherited event-handler call when the user invokes the control's context menu.

MousePos indicates the position where the user right-clicked the control. If the user invoked the context menu using the keyboard, MousePos is (-1,-1).

Handled returns a Boolean value indicating whether the control should continue with its default response to the user action.

See Also

OnContextPopup

76.1.4.5.24 TCustomRibbon.DoCustomizeQuickAccessToolbar Method

Pascal

```
procedure DoCustomizeQuickAccessToolbar; virtual;
```

C++

```
virtual __fastcall DoCustomizeQuickAccessToolbar();
```

Description

This is DoCustomizeQuickAccessToolbar, a member of class TCustomRibbon.

76.1.4.5.25 TCustomRibbon.DoGroupStyleChanged Method

Pascal

```
procedure DoGroupStyleChanged; virtual;
```

C++

```
virtual __fastcall DoGroupStyleChanged();
```

Description

This is DoGroupStyleChanged, a member of class TCustomRibbon.

76.1.4.5.26 TCustomRibbon.DoHelpButtonClick Method

Pascal

```
procedure DoHelpButtonClick; virtual;
```

C++

```
virtual __fastcall DoHelpButtonClick();
```

Description

This is DoHelpButtonClick, a member of class TCustomRibbon.

76.1.4.5.27 TCustomRibbon.DoMinimizeRibbon Method

Pascal

```
procedure DoMinimizeRibbon; virtual;
```

C++

```
virtual __fastcall DoMinimizeRibbon();
```

Description

This is DoMinimizeRibbon, a member of class TCustomRibbon.

76.1.4.5.28 TCustomRibbon.DoMouseWheelDown Method

OnMouseWheelDown event dispatcher.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelDown to generate an OnMouseWheelDown event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelDown returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelDown to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated downward.

See Also

OnMouseWheelDown

DoMouseWheel

76.1.4.5.29 TCustomRibbon.DoMouseWheelEx Method

Pascal

```
function DoMouseWheelEx(Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint): Boolean;
```

C++

```
__fastcall Boolean DoMouseWheelEx(TShiftState Shift, int WheelDelta, TPoint MousePos);
```

Description

This is DoMouseWheelEx, a member of class TCustomRibbon.

76.1.4.5.30 TCustomRibbon.DoMouseWheelUp Method

OnMouseWheelUp event dispatcher.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelUp to generate an OnMouseWheelUp event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelUp returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelUp to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated upward.

See Also

OnMouseWheelUp

DoMouseWheel

76.1.4.5.31 TCustomRibbon.DoRecentItemClick Method

Pascal

```
procedure DoRecentItemClick(const FileName: string; const Index: Integer); virtual;
```

C++

```
virtual __fastcall DoRecentItemClick(const UnicodeString FileName, const int Index);
```

Description

This is DoRecentItemClick, a member of class TCustomRibbon.

76.1.4.5.32 TCustomRibbon.DoScrollTabsLeft Method

Pascal

```
procedure DoScrollTabsLeft; virtual;
```

C++

```
virtual __fastcall DoScrollTabsLeft();
```

Description

This is DoScrollTabsLeft, a member of class TCustomRibbon.

76.1.4.5.33 TCustomRibbon.DoScrollTabsRight Method

Pascal

```
procedure DoScrollTabsRight; virtual;
```

C++

```
virtual __fastcall DoScrollTabsRight();
```

Description

This is DoScrollTabsRight, a member of class TCustomRibbon.

76.1.4.5.34 TCustomRibbon.DoTabChange Method

Pascal

```
procedure DoTabChange(const NewIndex: Integer; const OldIndex: Integer; var AllowChange:
Boolean); virtual;
```

C++

```
virtual __fastcall DoTabChange(const int NewIndex, const int OldIndex, Boolean AllowChange);
```

Description

This is DoTabChange, a member of class TCustomRibbon.

76.1.4.5.35 TCustomRibbon.DrawCaption Method

Pascal

```
procedure DrawCaption; virtual;
```

C++

```
virtual __fastcall DrawCaption();
```

Description

This is DrawCaption, a member of class TCustomRibbon.

76.1.4.5.36 TCustomRibbon.DrawQuickAccessToolbar Method

Pascal

```
procedure DrawQuickAccessToolbar; virtual;
```

C++

```
virtual __fastcall DrawQuickAccessToolbar();
```

Description

This is DrawQuickAccessToolbar, a member of class TCustomRibbon.

76.1.4.5.37 TCustomRibbon.DrawSystemButtons Method

Pascal

```
procedure DrawSystemButtons;
```

C++

```
__fastcall DrawSystemButtons();
```

Description

This is DrawSystemButtons, a member of class TCustomRibbon.

76.1.4.5.38 TCustomRibbon.DrawTab Method

Pascal

```
procedure DrawTab(const Index: Integer); virtual;
```

C++

```
virtual __fastcall DrawTab(const int Index);
```

Description

This is DrawTab, a member of class TCustomRibbon.

76.1.4.5.39 TCustomRibbon.EndUpdate Method

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

This is EndUpdate, a member of class TCustomRibbon.

76.1.4.5.40 TCustomRibbon.GetContextMenuOptions Method

Pascal

```
function GetContextMenuOptions: TContextMenuOptions;
```

C++

```
__fastcall TContextMenuOptions GetContextMenuOptions();
```

Description

This is GetContextMenuOptions, a member of class TCustomRibbon.

76.1.4.5.41 TCustomRibbon.GetHelpButtonRect Method

Pascal

```
function GetHelpButtonRect: TRect; virtual;
```

C++

```
virtual __fastcall TRect GetHelpButtonRect();
```

Description

This is GetHelpButtonRect, a member of class TCustomRibbon.

76.1.4.5.42 TCustomRibbon.GetIcon Method

Return Icon used by Application (see page 2612) Menu (see page 6762) Button.

Pascal

```
function GetIcon(out IconSize: TIconSize): TIcon; virtual;
```

C++

```
virtual __fastcall TIcon GetIcon(TIconSize IconSize);
```

Description

GetIcon returns the Icon used by the Application (see page 2612) Menu (see page 6762) Button. If there is no Icon assigned to the TApplicationMenu.Icon property, the application's icon is used.

See Also

TApplicationMenu (see page 6716)

Icon (see page 6720)

76.1.4.5.43 TCustomRibbon.GetRibbonHeight Method

Pascal

```
function GetRibbonHeight: Integer;
```

C++

```
__fastcall int GetRibbonHeight();
```

Description

This is GetRibbonHeight, a member of class TCustomRibbon.

76.1.4.5.44 TCustomRibbon.GetRibbonMetric Method

Pascal

```
function GetRibbonMetric(const Metric: TRibbonMetric): Integer;
```

C++

```
__fastcall int GetRibbonMetric(const TRibbonMetric Metric);
```

Description

This is GetRibbonMetric, a member of class TCustomRibbon.

76.1.4.5.45 TCustomRibbon.GetTabAt Method

Pascal

```
function GetTabAt(const X: Integer; const Y: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetTabAt(const int X, const int Y);
```

Description

This is GetTabAt, a member of class TCustomRibbon.

76.1.4.5.46 TCustomRibbon.GetTabRect Method

Pascal

```
function GetTabRect(const Index: Integer): TRect; virtual;
```

C++

```
virtual __fastcall TRect GetTabRect(const int Index);
```

Description

This is GetTabRect, a member of class TCustomRibbon.

76.1.4.5.47 TCustomRibbon.GetTabWidth Method

Pascal

```
function GetTabWidth(const Index: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetTabWidth(const int Index);
```

Description

This is GetTabWidth, a member of class TCustomRibbon.

76.1.4.5.48 TCustomRibbon.HasApplicationMenu Method

Pascal

```
function HasApplicationMenu: Boolean;
```

C++

```
__fastcall Boolean HasApplicationMenu();
```

Description

This is HasApplicationMenu, a member of class TCustomRibbon.

76.1.4.5.49 TCustomRibbon.HasQuickAccessToolbar Method

Pascal

```
function HasQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean HasQuickAccessToolbar();
```

Description

This is HasQuickAccessToolbar, a member of class TCustomRibbon.

76.1.4.5.50 TCustomRibbon.Idle Method

Pascal

```
procedure Idle(const Msg: TMsg);
```

C++

```
__fastcall Idle(const TMsg Msg);
```

Description

This is Idle, a member of class TCustomRibbon.

76.1.4.5.51 TCustomRibbon.InvalidateHeaderArea Method

Invalidate header section of Ribbon (see page 6713).

Pascal

```
procedure InvalidateHeaderArea;
```

C++

```
__fastcall InvalidateHeaderArea();
```

Description

InvalidateHeaderArea invalidates the header section of the Ribbon (see page 6713). Invalidating the header of the ribbon is especially important when using a custom frame. When using a custom frame, the header section of the ribbon needs to react visually to additional non-client specific messages.

See Also

InvalidateTabArea (see page 6748)

76.1.4.5.52 TCustomRibbon.InvalidateScrollButtons Method

Pascal

```
procedure InvalidateScrollButtons;
```

C++

```
__fastcall InvalidateScrollButtons();
```

Description

This is InvalidateScrollButtons, a member of class TCustomRibbon.

76.1.4.5.53 TCustomRibbon.InvalidateSystemMenuArea Method

Pascal

```
procedure InvalidateSystemMenuArea;
```

C++

```
__fastcall InvalidateSystemMenuArea();
```

Description

This is InvalidateSystemMenuArea, a member of class TCustomRibbon.

76.1.4.5.54 TCustomRibbon.InvalidateTabArea Method

Invalidate area that displays tabs.

Pascal

```
procedure InvalidateTabArea;
```

C++

```
__fastcall InvalidateTabArea();
```

Description

InvalidateTabArea invalidates the area that displays the tabs.

See Also

InvalidateHeaderArea (see page 6747)

76.1.4.5.55 TCustomRibbon.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

76.1.4.5.56 TCustomRibbon.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

76.1.4.5.57 TCustomRibbon.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

76.1.4.5.58 TCustomRibbon.MouseInSystemMenu Method

Pascal

```
function MouseInSystemMenu(const LPt: TPoint): Boolean;
```

C++

```
__fastcall Boolean MouseInSystemMenu(const TPoint LPt);
```

Description

This is MouseInSystemMenu, a member of class TCustomRibbon.

76.1.4.5.59 TCustomRibbon.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

76.1.4.5.60 TCustomRibbon.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

76.1.4.5.61 TCustomRibbon.MoveGroup Method

Move group to new page.

Pascal

```
procedure MoveGroup(const GroupName: string; const PageName: string);
```

C++

```
__fastcall MoveGroup(const UnicodeString GroupName, const UnicodeString PageName);
```

Description

MoveGroup moves a group to a new page. The group to be moved is specified in the **GroupName** parameter. The destination page is specified in the **PageName** parameter.

76.1.4.5.62 TCustomRibbon.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

76.1.4.5.63 TCustomRibbon.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

76.1.4.5.64 TCustomRibbon.RebuildRibbonMetrics Method

Pascal

```
procedure RebuildRibbonMetrics;
```

C++

```
__fastcall RebuildRibbonMetrics();
```

Description

This is RebuildRibbonMetrics, a member of class TCustomRibbon.

76.1.4.5.65 TCustomRibbon.RemoveAllTabs Method

Remove all Tabs (see page 6764) from Ribbon (see page 6713).

Pascal

```
procedure RemoveAllTabs;
```

C++

```
__fastcall RemoveAllTabs();
```

Description

RemoveAllTabs removes all Tabs (see page 6764) from the Ribbon (see page 6713). All associated pages and groups are deleted.

See Also

RemoveTab (see page 6753)

76.1.4.5.66 TCustomRibbon.RemoveRecentItem Method

Pascal

```
function RemoveRecentItem(const FileName: string): Boolean;
```

C++

```
__fastcall Boolean RemoveRecentItem(const UnicodeString FileName);
```

Description

This is RemoveRecentItem, a member of class TCustomRibbon.

76.1.4.5.67 TCustomRibbon.RemoveTab Method

Remove tab with specified caption.

Pascal

```
procedure RemoveTab(const ACaption: string);
```

C++

```
__fastcall RemoveTab(const UnicodeString ACaption);
```

Description

RemoveTab removes the tab with the specified caption. RemoveTab deletes the associated page and any groups on that page.

See Also

RemoveAllTabs (see page 6753)

76.1.4.5.68 TCustomRibbon.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize

76.1.4.5.69 TCustomRibbon.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

76.1.4.5.70 TCustomRibbon.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

76.1.4.5.71 TCustomRibbon.TabHandledKeyTips Method

Validate **KeyTip** input when **KeyTip**s State is **ktsTabs**.

Pascal

```
procedure TabHandledKeyTips(const KeyTip: string; var KeyTips: TKeyTipList);
```

C++

```
__fastcall TabHandledKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

TabHandledKeyTips validates **KeyTip** input when the **KeyTip**s State is **ktsTabs**. **KeyTip**s can be in either **ktsTabs** or **ktsControls** state. When the **KeyTip**s state is **ktsTabs**, the user has the option of selecting which Tab to switch to and display the **KeyTip**s for the controls on that tab. When the State **ktsTabs**, the Application (see page 2612) Button and Quick Access Toolbar are also displaying their **KeyTip**s.

The parameters are:

- KeyTip: The **KeyTip** the user has input that needs to be validated against.
- Full: A list of TKeyTipMatchItem (see page 6962) that contain details if any **KeyTip**s were matched.

See Also

TKeyTipMatchItem (see page 6962)

TKeyTipState (see page 7288)

76.1.4.5.72 TCustomRibbon.TabHideKeyTips Method

Hide K**eyTip**s.

Pascal

```
procedure TabHideKeyTips(const KeyTip: string; KeyTips: TKeyTipList);
```

C++

```
__fastcall TabHideKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

TabHideKeyTips hides the **KeyTip**s. If the **KeyTips** parameter is `nil`, all **KeyTip**s are hidden. Otherwise **KeyTips** contains TKeyTipMatchItem (see page 6962) that are not supposed to be hidden at this time. This would occur when there are a number of Partial **KeyTip** matches where the first character of a **KeyTip** matches many **KeyTip**s.

See Also

TKeyTipMatchItem (see page 6962)

TabShowKeyTips (see page 6756)

76.1.4.5.73 TCustomRibbon.TabShowKeyTips Method

Display **KeyTip**s for **ktsTabs** State.

Pascal

```
procedure TabShowKeyTips;
```

C++

```
__fastcall TabShowKeyTips();
```

Description

TabShowKeyTips displays the **KeyTip**s for the **ktsTabs** State. When called, **KeyTip**s are displayed for the Tabs (see page 6764), Application (see page 2612) Button and Quick Access Toolbar.

See Also

TabHideKeyTips (see page 6755)

TKeyTipState (see page 7288)

76.1.4.5.74 TCustomRibbon.UMCheckResize Method

Pascal

```
procedure UMCheckResize(var Message: TMessage); message UM_CHECKSIZE;
```

C++

```
__fastcall UMCheckResize(TMessage Message);
```

Description

This is UMCheckResize, a member of class TCustomRibbon.

76.1.4.5.75 TCustomRibbon.UMDisplayKeyTips Method

Pascal

```
procedure UMDisplayKeyTips(var Message: TMessage); message UM_DISPLAYKEYTIPS;
```

C++

```
__fastcall UMDisplayKeyTips(TMessage Message);
```

Description

This is UMDisplayKeyTips, a member of class TCustomRibbon.

76.1.4.5.76 TCustomRibbon.UpdateQATPosition Method

Pascal

```
procedure UpdateQATPosition;
```

C++

```
__fastcall UpdateQATPosition();
```

Description

This is UpdateQATPosition, a member of class TCustomRibbon.

76.1.4.5.77 TCustomRibbon.UpdateSystemMenuInfo Method

Pascal

```
procedure UpdateSystemMenuInfo;
```

C++

```
__fastcall UpdateSystemMenuInfo();
```

Description

This is UpdateSystemMenuInfo, a member of class TCustomRibbon.

76.1.4.5.78 TCustomRibbon.ValidTab Method

Pascal

```
function ValidTab(const Index: Integer): Boolean;
```

C++

```
__fastcall Boolean ValidTab(const int Index);
```

Description

This is ValidTab, a member of class TCustomRibbon.

76.1.4.5.79 TCustomRibbon.WMEraseBkgnd Method

Pascal

```
procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
```

C++

```
__fastcall WMEraseBkgnd(TWMEraseBkgnd Message);
```

Description

This is WMEraseBkgnd, a member of class TCustomRibbon.

76.1.4.5.80 TCustomRibbon.WMGetDlgCode Method

Pascal

```
procedure WMGetDlgCode(var Message: TWMGetDlgCode); message WM_GETDLGCODE;
```

C++

```
__fastcall WMGetDlgCode(TWMGetDlgCode Message);
```

Description

This is WMGetDlgCode, a member of class TCustomRibbon.

76.1.4.5.81 TCustomRibbon.WMKillFocus Method

Pascal

```
procedure WMKillFocus(var Message: TWMKillFocus); message WM_KILLFOCUS;
```

C++

```
__fastcall WMKillFocus(TWMKillFocus Message);
```

Description

This is WMKillFocus, a member of class TCustomRibbon.

76.1.4.5.82 TCustomRibbon.WMSettingChange Method

Pascal

```
procedure WMSettingChange(var Message: TWMSettingChange); message WM_SETTINGCHANGE;
```

C++

```
__fastcall WMSettingChange(TWMSettingChange Message);
```

Description

This is WMSettingChange, a member of class TCustomRibbon.

76.1.4.5.83 TCustomRibbon.WMWindowPosChanged Method

Pascal

```
procedure WMWindowPosChanged(var Message: TWMWindowPosChanged); message WM_WINDOWPOSCHANGED;
```

C++

```
__fastcall WMWindowPosChanged(TWMWindowPosChanged Message);
```

Description

This is WMWindowPosChanged, a member of class TCustomRibbon.

76.1.4.6 TCustomRibbon Properties

The properties of the TCustomRibbon class are listed here.

Protected Properties

	Name	Description
	AllowKeyboardNavigation (see page 6760)	This is AllowKeyboardNavigation, a member of class TCustomRibbon.
	HideRibbon (see page 6761)	This is HideRibbon, a member of class TCustomRibbon.
	HotTabIndex (see page 6762)	This is HotTabIndex, a member of class TCustomRibbon.
	KeyTipsVisible (see page 6762)	This is KeyTipsVisible, a member of class TCustomRibbon.
R	Menu (see page 6762)	This is Menu, a member of class TCustomRibbon.
R	ScaleFactor (see page 6763)	This is ScaleFactor, a member of class TCustomRibbon.

Public Properties

	Name	Description
	ActionManager (see page 6759)	This is ActionManager, a member of class TCustomRibbon.
	Active (see page 6759)	Indicates if form that ribbon is placed on is active form.
	ActivePage (see page 6760)	Page currently displayed by ribbon control.
	ApplicationMenu (see page 6760)	Change appearance of ribbon's associated TRibbonApplicationMenuBar (see page 7302) control.
	ColorMap (see page 6761)	This is ColorMap, a member of class TCustomRibbon.
	DocumentName (see page 6761)	Text in Ribbon (see page 6713) caption.
	HideTabs (see page 6762)	Shows or hides Tab portion of Ribbon (see page 6713).
	Minimized (see page 6763)	Show Ribbon (see page 6713) in minimized state where tabs can be selected but are not visible.
	QuickAccessToolbar (see page 6763)	Properties associated with TQuickAccessToolbar (see page 6835).
	ScreenTips (see page 6763)	This is ScreenTips, a member of class TCustomRibbon.
	ShowHelpButton (see page 6764)	Determines whether ribbon displays Help Button.
	Style (see page 6764)	This is Style, a member of class TCustomRibbon.
	TabIndex (see page 6764)	Change the active page of ribbon.
	Tabs (see page 6764)	Collection of TRibbonTabs (see page 6925) visible on ribbon.
	UseCustomFrame (see page 6765)	Indicates whether to use a custom frame for ribbon controls.

76.1.4.6.1 TCustomRibbon.ActionManager Property

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

This is ActionManager, a member of class TCustomRibbon.

76.1.4.6.2 TCustomRibbon.Active Property

Indicates if form that ribbon is placed on is active form.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Active indicates if the form that the ribbon is placed on is the active form of the desktop.

There should be no need for a user to set the Active property unless they are not using the custom frame functionality of the

ribbon. The appearance of the ribbon changes depending on the value of this property.

If the form that the ribbon is placed on does not descend from TRibbonForm, the user should update the Active property when the form gets activated and deactivated.

See Also

ActivePage (see page 6760)

76.1.4.6.3 TCustomRibbon.ActivePage Property

Page currently displayed by ribbon control.

Pascal

```
property ActivePage: TCustomRibbonPage;
```

C++

```
__property TCustomRibbonPage ActivePage;
```

Description

ActivePage is the page currently displayed by the ribbon control.

Read ActivePage to determine which page is selected by the user. The user selects a page by clicking the mouse on its tab or selects the page's associated keytip. Set ActivePage to bring a page to the foreground. Only one page can be active at a time.

Use TRibbonTabs.SelectNext and TRibbonTabs.SelectPrior to select the next and prior pages in the ribbon. If the active page does not change after a call to SelectNext or SelectPrior, the return value of those functions is False.

To change the active page of the ribbon by Index, set the TabIndex (see page 6764) property to the desired page index.

See Also

TRibbonTabs (see page 6925)

Active (see page 6759)

TabIndex (see page 6764)

SelectNext (see page 6928)

SelectPrior (see page 6928)

76.1.4.6.4 TCustomRibbon.AllowKeyboardNavigation Property

Pascal

```
property AllowKeyboardNavigation: Boolean;
```

C++

```
__property Boolean AllowKeyboardNavigation;
```

Description

This is AllowKeyboardNavigation, a member of class TCustomRibbon.

76.1.4.6.5 TCustomRibbon.ApplicationMenu Property

Change appearance of ribbon's associated TRibbonApplicationMenuBar (see page 7302) control.

Pascal

```
property ApplicationMenu: TApplicationMenu;
```

C++

```
__property TApplicationMenu ApplicationMenu;
```

Description

ApplicationMenu changes the appearance for the ribbon's associated TRibbonApplicationMenuBar (see page 7302) control.

Changing the values of the fields in ApplicationMenu changes the characteristics of the Application (see page 2612) Menu (see page 6762) for the ribbon. The Application (see page 2612) Menu (see page 6762) on the ribbon is represented by the round button in the top left corner.

See Also

TApplicationMenu (see page 6716)

TRibbonApplicationMenuBar (see page 7302)

76.1.4.6.6 TCustomRibbon.ColorMap Property

Pascal

```
property ColorMap: TCustomActionBarColorMap;
```

C++

```
__property TCustomActionBarColorMap ColorMap;
```

Description

This is ColorMap, a member of class TCustomRibbon.

76.1.4.6.7 TCustomRibbon.DocumentName Property

Text in Ribbon (see page 6713) caption.

Pascal

```
property DocumentName: string;
```

C++

```
__property UnicodeString DocumentName;
```

Description

DocumentName appears in the Ribbon (see page 6713) caption.

See Also

CaptionFont

76.1.4.6.8 TCustomRibbon.HideRibbon Property

Pascal

```
property HideRibbon: Boolean;
```

C++

```
__property Boolean HideRibbon;
```

Description

This is HideRibbon, a member of class TCustomRibbon.

76.1.4.6.9 TCustomRibbon.HideTabs Property

Shows or hides Tab portion of Ribbon (see page 6713).

Pascal

```
property HideTabs: Boolean;
```

C++

```
__property Boolean HideTabs;
```

Description

HideTabs shows or hides the Tab portion of the Ribbon (see page 6713). When HideTabs is True, the user cannot access any of the Tabs (see page 6764) on the Ribbon (see page 6713).

76.1.4.6.10 TCustomRibbon.HotTabIndex Property

Pascal

```
property HotTabIndex: Integer;
```

C++

```
__property int HotTabIndex;
```

Description

This is HotTabIndex, a member of class TCustomRibbon.

76.1.4.6.11 TCustomRibbon.KeyTipsVisible Property

Pascal

```
property KeyTipsVisible: Boolean;
```

C++

```
__property Boolean KeyTipsVisible;
```

Description

This is KeyTipsVisible, a member of class TCustomRibbon.

76.1.4.6.12 TCustomRibbon.Menu Property

Pascal

```
property Menu: TRibbonApplicationMenuBar;
```

C++

```
__property TRibbonApplicationMenuBar Menu;
```

Description

This is Menu, a member of class TCustomRibbon.

76.1.4.6.13 TCustomRibbon.Minimized Property

Show Ribbon (see page 6713) in minimized state where tabs can be selected but are not visible.

Pascal

```
property Minimized: Boolean;
```

C++

```
__property Boolean Minimized;
```

Description

Minimized indicates whether to show the Ribbon (see page 6713) in a minimized state where tabs can be selected but are not visible. When Minimized is True, Tabs (see page 6764) still function. You can access the tabs by selecting the desired Tab with the mouse or switching to the tab by using tab allocated `KeyTips`.

76.1.4.6.14 TCustomRibbon.QuickAccessToolbar Property

Properties associated with TQuickAccessToolbar (see page 6835).

Pascal

```
property QuickAccessToolbar: TQuickAccessToolbar;
```

C++

```
__property TQuickAccessToolbar QuickAccessToolbar;
```

Description

QuickAccessToolbar has the properties associated with the TQuickAccessToolbar (see page 6835) on the ribbon. Use these properties to change the appearance of the Quick Access Toolbar as displayed on the ribbon.

See Also

TQuickAccessToolbar (see page 6835)

76.1.4.6.15 TCustomRibbon.ScaleFactor Property

Pascal

```
property ScaleFactor: Single;
```

C++

```
__property Single ScaleFactor;
```

Description

This is ScaleFactor, a member of class TCustomRibbon.

76.1.4.6.16 TCustomRibbon.ScreenTips Property

Pascal

```
property ScreenTips: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTips;
```

Description

This is ScreenTips, a member of class TCustomRibbon.

76.1.4.6.17 TCustomRibbon.ShowHelpButton Property

Determines whether ribbon displays Help Button.

Pascal

```
property ShowHelpButton: Boolean;
```

C++

```
__property Boolean ShowHelpButton;
```

Description

ShowHelpButton indicates whether or not the ribbon displays a Help Button in the top right hand corner.

See Also

THelpButton (see page 6825)

76.1.4.6.18 TCustomRibbon.Style Property

Pascal

```
property Style: TRibbonStyleActionBars;
```

C++

```
__property TRibbonStyleActionBars Style;
```

Description

This is Style, a member of class TCustomRibbon.

76.1.4.6.19 TCustomRibbon.TabIndex Property

Change the active page of ribbon.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

TabIndex changes the active page of the ribbon. Set the TabIndex property to the desired page index.

See Also

Tabs (see page 6764)

76.1.4.6.20 TCustomRibbon.Tabs Property

Collection of TRibbonTabs (see page 6925) visible on ribbon.

Pascal

```
property Tabs: TRibbonTabs;
```

C++

```
__property TRibbonTabs Tabs;
```

Description

Tabs contains a collection of TRibbonTabs (see page 6925) visible on the ribbon.

Tabs is a linked list of TRibbonTabs (see page 6925) objects. TRibbonTabs (see page 6925) is a simple data structure that contains the caption for the tab, the keytip for the tab as well as a reference to the Tab object.

See Also

TRibbonTabs (see page 6925)

TabIndex (see page 6764)

76.1.4.6.21 TCustomRibbon.UseCustomFrame Property

Indicates whether to use a custom frame for ribbon controls.

Pascal

```
property UseCustomFrame: Boolean;
```

C++

```
__property Boolean UseCustomFrame;
```

Description

UseCustomFrame indicates whether or not to use a custom frame, such as TRibbonForm, for ribbon controls.

See Also

TRibbonForm

76.1.4.7 TCustomRibbon Nested Types

The nested types of the TCustomRibbon class are listed here.

Public Nested Types

Name	Description
TTabChangeEvent (see page 6765)	This is nested type Ribbon.TCustomRibbon.TTabChangeEvent.

76.1.4.7.1 TCustomRibbon.TTabChangeEvent Nested Type

Pascal

```
TTabChangeEvent = procedure (Sender: TObject; const NewIndex, OldIndex: Integer; var
AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; const NewIndex, OldIndex: Integer; var AllowChange: Boolean) (
TTabChangeEvent)();
```

Description

This is nested type Ribbon.TCustomRibbon.TTabChangeEvent.

76.1.4.8 TCustomRibbon Events

The events of the TCustomRibbon class are listed here.

Protected Events

	Name	Description
	OnLeftTabScrollButtonClick (see page 6766)	This is OnLeftTabScrollButtonClick, a member of class TCustomRibbon.
	OnRightTabScrollButtonClick (see page 6767)	This is OnRightTabScrollButtonClick, a member of class TCustomRibbon.

Public Events

	Name	Description
	OnHelpButtonClick (see page 6766)	This is OnHelpButtonClick, a member of class TCustomRibbon.
	OnRecentItemClick (see page 6766)	This is OnRecentItemClick, a member of class TCustomRibbon.
	OnTabChange (see page 6767)	This is OnTabChange, a member of class TCustomRibbon.

76.1.4.8.1 TCustomRibbon.OnHelpButtonClick Event

Pascal

```
property OnHelpButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHelpButtonClick;
```

Description

This is OnHelpButtonClick, a member of class TCustomRibbon.

76.1.4.8.2 TCustomRibbon.OnLeftTabScrollButtonClick Event

Pascal

```
property OnLeftTabScrollButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLeftTabScrollButtonClick;
```

Description

This is OnLeftTabScrollButtonClick, a member of class TCustomRibbon.

76.1.4.8.3 TCustomRibbon.OnRecentItemClick Event

Pascal

```
property OnRecentItemClick: TRecentItemClickEvent;
```

C++

```
__property TRecentItemClickEvent OnRecentItemClick;
```

Description

This is OnRecentItemClick, a member of class TCustomRibbon.

76.1.4.8.4 TCustomRibbon.OnRightTabScrollButtonClick Event

Pascal

```
property OnRightTabScrollButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRightTabScrollButtonClick;
```

Description

This is OnRightTabScrollButtonClick, a member of class TCustomRibbon.

76.1.4.8.5 TCustomRibbon.OnTabChange Event

Pascal

```
property OnTabChange: TTabChangeEvent;
```

C++

```
__property TTabChangeEvent OnTabChange;
```

Description

This is OnTabChange, a member of class TCustomRibbon.

76.1.5 TCustomRibbonColorMap Class

Class Hierarchy

Pascal

```
TCustomRibbonColorMap = class(TXPColorMap);
```

C++

```
class TCustomRibbonColorMap : public TXPColorMap;
```

File

Ribbon

Description

This is class Ribbon.TCustomRibbonColorMap.

76.1.5.1 TCustomRibbonColorMap Members

The following tables list the members exposed by TCustomRibbonColorMap.

Public Properties

	Name	Description
	ActiveTabFontColor (see page 6768)	This is ActiveTabFontColor, a member of class TCustomRibbonColorMap.
	BevelMinimizedInner (see page 6768)	This is BevelMinimizedInner, a member of class TCustomRibbonColorMap.
	BevelMinimizedOuter (see page 6769)	This is BevelMinimizedOuter, a member of class TCustomRibbonColorMap.

	CaptionFontColor (see page 6769)	This is CaptionFontColor, a member of class TCustomRibbonColorMap.
	ControlColor (see page 6769)	This is ControlColor, a member of class TCustomRibbonColorMap.
	ControlFrameColor (see page 6769)	This is ControlFrameColor, a member of class TCustomRibbonColorMap.
	ControlInactiveColor (see page 6769)	This is ControlInactiveColor, a member of class TCustomRibbonColorMap.
	DocumentFontColor (see page 6770)	This is DocumentFontColor, a member of class TCustomRibbonColorMap.
	GroupFontColor (see page 6770)	This is GroupFontColor, a member of class TCustomRibbonColorMap.
	InactiveTabFontColor (see page 6770)	This is InactiveTabFontColor, a member of class TCustomRibbonColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6770)	This is QuickAccessToolbarMoreBorderColor, a member of class TCustomRibbonColorMap.
	QuickAccessToolbarMoreColor (see page 6771)	This is QuickAccessToolbarMoreColor, a member of class TCustomRibbonColorMap.

76.1.5.2 TCustomRibbonColorMap Properties

The properties of the TCustomRibbonColorMap class are listed here.

Public Properties

	Name	Description
	ActiveTabFontColor (see page 6768)	This is ActiveTabFontColor, a member of class TCustomRibbonColorMap.
	BevelMinimizedInner (see page 6768)	This is BevelMinimizedInner, a member of class TCustomRibbonColorMap.
	BevelMinimizedOuter (see page 6769)	This is BevelMinimizedOuter, a member of class TCustomRibbonColorMap.
	CaptionFontColor (see page 6769)	This is CaptionFontColor, a member of class TCustomRibbonColorMap.
	ControlColor (see page 6769)	This is ControlColor, a member of class TCustomRibbonColorMap.
	ControlFrameColor (see page 6769)	This is ControlFrameColor, a member of class TCustomRibbonColorMap.
	ControlInactiveColor (see page 6769)	This is ControlInactiveColor, a member of class TCustomRibbonColorMap.
	DocumentFontColor (see page 6770)	This is DocumentFontColor, a member of class TCustomRibbonColorMap.
	GroupFontColor (see page 6770)	This is GroupFontColor, a member of class TCustomRibbonColorMap.
	InactiveTabFontColor (see page 6770)	This is InactiveTabFontColor, a member of class TCustomRibbonColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6770)	This is QuickAccessToolbarMoreBorderColor, a member of class TCustomRibbonColorMap.
	QuickAccessToolbarMoreColor (see page 6771)	This is QuickAccessToolbarMoreColor, a member of class TCustomRibbonColorMap.

76.1.5.2.1 TCustomRibbonColorMap.ActiveTabFontColor Property

Pascal

```
property ActiveTabFontColor: TColor;
```

C++

```
__property TColor ActiveTabFontColor;
```

Description

This is ActiveTabFontColor, a member of class TCustomRibbonColorMap.

76.1.5.2.2 TCustomRibbonColorMap.BevelMinimizedInner Property

Pascal

```
property BevelMinimizedInner: TColor;
```

C++

```
__property TColor BevelMinimizedInner;
```

Description

This is BevelMinimizedInner, a member of class TCustomRibbonColorMap.

76.1.5.2.3 TCustomRibbonColorMap.BevelMinimizedOuter Property

Pascal

```
property BevelMinimizedOuter: TColor;
```

C++

```
__property TColor BevelMinimizedOuter;
```

Description

This is BevelMinimizedOuter, a member of class TCustomRibbonColorMap.

76.1.5.2.4 TCustomRibbonColorMap.CaptionFontColor Property

Pascal

```
property CaptionFontColor: TColor;
```

C++

```
__property TColor CaptionFontColor;
```

Description

This is CaptionFontColor, a member of class TCustomRibbonColorMap.

76.1.5.2.5 TCustomRibbonColorMap.ControlColor Property

Pascal

```
property ControlColor: TColor;
```

C++

```
__property TColor ControlColor;
```

Description

This is ControlColor, a member of class TCustomRibbonColorMap.

76.1.5.2.6 TCustomRibbonColorMap.ControlFrameColor Property

Pascal

```
property ControlFrameColor: TColor;
```

C++

```
__property TColor ControlFrameColor;
```

Description

This is ControlFrameColor, a member of class TCustomRibbonColorMap.

76.1.5.2.7 TCustomRibbonColorMap.ControlInactiveColor Property

Pascal

```
property ControlInactiveColor: TColor;
```

C++

```
__property TColor ControlInactiveColor;
```

Description

This is ControlInactiveColor, a member of class TCustomRibbonColorMap.

76.1.5.2.8 TCustomRibbonColorMap.DocumentFontColor Property

Pascal

```
property DocumentFontColor: TColor;
```

C++

```
__property TColor DocumentFontColor;
```

Description

This is DocumentFontColor, a member of class TCustomRibbonColorMap.

76.1.5.2.9 TCustomRibbonColorMap.GroupFontColor Property

Pascal

```
property GroupFontColor: TColor;
```

C++

```
__property TColor GroupFontColor;
```

Description

This is GroupFontColor, a member of class TCustomRibbonColorMap.

76.1.5.2.10 TCustomRibbonColorMap.InactiveTabFontColor Property

Pascal

```
property InactiveTabFontColor: TColor;
```

C++

```
__property TColor InactiveTabFontColor;
```

Description

This is InactiveTabFontColor, a member of class TCustomRibbonColorMap.

76.1.5.2.11 TCustomRibbonColorMap.QuickAccessToolbarMoreBorderColor Property

Pascal

```
property QuickAccessToolbarMoreBorderColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreBorderColor;
```

Description

This is QuickAccessToolbarMoreBorderColor, a member of class TCustomRibbonColorMap.

76.1.5.2.12 TCustomRibbonColorMap.QuickAccessToolbarMoreColor Property

Pascal

```
property QuickAccessToolbarMoreColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreColor;
```

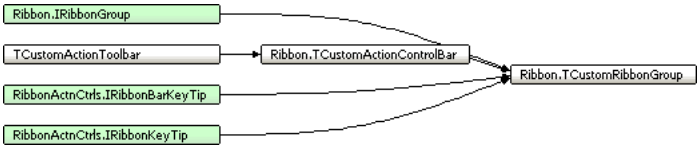
Description

This is QuickAccessToolbarMoreColor, a member of class TCustomRibbonColorMap.

76.1.6 TCustomRibbonGroup Class

Align and display all controls on ribbon.

Class Hierarchy



Pascal

```
TCustomRibbonGroup = class(TCustomActionControlBar, IRibbonGroup, IRibbonBarKeyTip,
IRibbonKeyTip);
```

C++

```
class TCustomRibbonGroup : public TCustomActionControlBar, public IRibbonGroup, public
IRibbonBarKeyTip, public IRibbonKeyTip;
```

File

Ribbon

Description

TCustomRibbonGroup aligns and displays all controls available for use on the ribbon. Controls are displayed on a Ribbon (see page 6713) Group.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TGroupAlign

76.1.6.1 TCustomRibbonGroup Members

The following tables list the members exposed by TCustomRibbonGroup.

Enumerations

	Name	Description
	TGroupAlign (see page 6774)	This is record Ribbon.TCustomRibbonGroup.TGroupAlign.

Protected Methods

	Name	Description
	AlignControls (see page 6775)	This is AlignControls, a member of class TCustomRibbonGroup.
	AutoSizingChanged (see page 6776)	This is AutoSizingChanged, a member of class TCustomRibbonGroup.
	CMControlChange (see page 6722)	This is CMControlChange, a member of class TCustomActionControlBar.
	CMFontchanged (see page 6777)	This is CMFontchanged, a member of class TCustomRibbonGroup.
	CMItemDropPoint (see page 6777)	This is CMItemDropPoint, a member of class TCustomRibbonGroup.
	CMMouseenter (see page 6778)	This is CMMouseenter, a member of class TCustomRibbonGroup.
	CMMouseleave (see page 6778)	This is CMMouseleave, a member of class TCustomRibbonGroup.
	CMTextchanged (see page 6778)	This is CMTextchanged, a member of class TCustomRibbonGroup.
	CreateControls (see page 6779)	This is CreateControls, a member of class TCustomRibbonGroup.
	DoDropItem (see page 6779)	This is DoDropItem, a member of class TCustomRibbonGroup.
	DoEndDrag (see page 6779)	This is DoEndDrag, a member of class TCustomRibbonGroup.
	DrawShowDialogButton (see page 6780)	This is DrawShowDialogButton, a member of class TCustomRibbonGroup.
	GetCaptionHeight (see page 6780)	This is GetCaptionHeight, a member of class TCustomRibbonGroup.
	GetControlClass (see page 6723)	This is GetControlClass, a member of class TCustomActionControlBar.
	GetGroupHeight (see page 6780)	This is GetGroupHeight, a member of class TCustomRibbonGroup.
	HideControls (see page 6781)	This is HideControls, a member of class TCustomRibbonGroup.
	MouseDown (see page 6782)	This is MouseDown, a member of class TCustomRibbonGroup.
	MouseMove (see page 6782)	This is MouseMove, a member of class TCustomRibbonGroup.
	MouseUp (see page 6783)	This is MouseUp, a member of class TCustomRibbonGroup.
	Notification (see page 6783)	This is Notification, a member of class TCustomRibbonGroup.
	Paint (see page 6783)	This is Paint, a member of class TCustomRibbonGroup.
	Resize (see page 6783)	This is Resize, a member of class TCustomRibbonGroup.
	SetAutoSizing (see page 6783)	This is SetAutoSizing, a member of class TCustomRibbonGroup.
	SetParent (see page 6784)	This is SetParent, a member of class TCustomRibbonGroup.
	SetupDropDownBtn (see page 6785)	This is SetupDropDownBtn, a member of class TCustomRibbonGroup.
	ShowControls (see page 6785)	This is ShowControls, a member of class TCustomRibbonGroup.

Protected Properties

	Name	Description
	CreatingControls (see page 6723)	This is CreatingControls, a member of class TCustomActionControlBar.

Public Methods

	Name	Description
	AlignGroupControls (see page 6775)	Align controls in Ribbon (see page 6713) Group
	Assign (see page 6775)	This is Assign, a member of class TCustomRibbonGroup.
	BarHandleKeyTips (see page 6776)	Iterate over any visible **KeyTip (see page 6788)**s and perform validation to determine if one of **KeyTip (see page 6788)**s is partial or full match.
	BarHideKeyTips (see page 6776)	Hide any visible KeyTips that commands are displaying.
	BarShowKeyTips (see page 6777)	Show K**eyTip**s for commands.
	Clone (see page 6777)	This is Clone, a member of class TCustomRibbonGroup.
	Create (see page 6773)	This is Create, a member of class TCustomRibbonGroup.
	CreateControl (see page 6778)	Recreate specific control.
	Destroy (see page 6773)	This is Destroy, a member of class TCustomRibbonGroup.
	DockDrop (see page 6779)	This is DockDrop, a member of class TCustomRibbonGroup.
	DragDrop (see page 6779)	This is DragDrop, a member of class TCustomRibbonGroup.
	GetKeyTip (see page 6780)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 6781)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6781)	Hide **KeyTip** for command.
	KeyTipClick (see page 6782)	Handle clicking a **KeyTip**.
	SetBounds (see page 6784)	This is SetBounds, a member of class TCustomRibbonGroup.
	SetKeyTip (see page 6784)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6785)	Shows **KeyTip** for command in specified position.

V	StyleChanged (see page 6785)	This is StyleChanged, a member of class TCustomRibbonGroup.

Public Nested Types

Name	Description
TRowPosition (see page 6790)	This is nested type Ribbon.TCustomRibbonGroup.TRowPosition.

Public Properties

	Name	Description
R	CanCollapse (see page 6786)	Determine whether group can be displayed collapsed.
R	CollapsedControl (see page 6786)	This is CollapsedControl, a member of class TCustomRibbonGroup.
	DialogAction (see page 6786)	Determines whether dialog button displayed in group.
	GroupAlign (see page 6787)	Determines how controls on group are aligned.
	GroupIndex (see page 6787)	Index of group within current tab.
	GroupVariant (see page 6787)	Current size variant ribbon group is using.
R	GroupWidth (see page 6788)	This is GroupWidth, a member of class TCustomRibbonGroup.
R	HasGallery (see page 6788)	Indicates if group has an in ribbon Gallery.
	Items (see page 6788)	This is Items, a member of class TCustomRibbonGroup.
	KeyTip (see page 6788)	Key tip for Dialog button of group.
R	Page (see page 6789)	Reference to Page group is on.
R	RequiredWidth (see page 6789)	Width required to display group.
	Rows (see page 6789)	Number of Rows in group has for controls.

76.1.6.2 TCustomRibbonGroup.Create Constructor

Pascal

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

C++

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

Description

This is Create, a member of class TCustomRibbonGroup.

76.1.6.3 TCustomRibbonGroup.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRibbonGroup();
```

Description

This is Destroy, a member of class TCustomRibbonGroup.

76.1.6.4 TCustomRibbonGroup Enumerations

The enumerations of the TCustomRibbonGroup class are listed here.

Enumerations

	Name	Description
	TGroupAlign (see page 6774)	This is record Ribbon.TCustomRibbonGroup.TGroupAlign.

76.1.6.4.1 Ribbon.TCustomRibbonGroup.TGroupAlign Enumeration

Pascal

```
TGroupAlign = (
  gaHorizontal,
  gaVertical
);
```

C++

```
enum TGroupAlign {
  gaHorizontal,
  gaVertical
};
```

File

Ribbon

Description

This is record Ribbon.TCustomRibbonGroup.TGroupAlign.

76.1.6.5 TCustomRibbonGroup Methods

The methods of the TCustomRibbonGroup class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 6775)	This is AlignControls, a member of class TCustomRibbonGroup.
	AutoSizingChanged (see page 6776)	This is AutoSizingChanged, a member of class TCustomRibbonGroup.
	CMFontchanged (see page 6777)	This is CMFontchanged, a member of class TCustomRibbonGroup.
	CMItemDropPoint (see page 6777)	This is CMItemDropPoint, a member of class TCustomRibbonGroup.
	CMMouseenter (see page 6778)	This is CMMouseenter, a member of class TCustomRibbonGroup.
	CMMouseleave (see page 6778)	This is CMMouseleave, a member of class TCustomRibbonGroup.
	CMTextchanged (see page 6778)	This is CMTextchanged, a member of class TCustomRibbonGroup.
	CreateControls (see page 6779)	This is CreateControls, a member of class TCustomRibbonGroup.
	DoDropItem (see page 6779)	This is DoDropItem, a member of class TCustomRibbonGroup.
	DoEndDrag (see page 6779)	This is DoEndDrag, a member of class TCustomRibbonGroup.
	DrawShowDialogButton (see page 6780)	This is DrawShowDialogButton, a member of class TCustomRibbonGroup.
	GetCaptionHeight (see page 6780)	This is GetCaptionHeight, a member of class TCustomRibbonGroup.
	GetGroupHeight (see page 6780)	This is GetGroupHeight, a member of class TCustomRibbonGroup.
	HideControls (see page 6781)	This is HideControls, a member of class TCustomRibbonGroup.
	MouseDown (see page 6782)	This is MouseDown, a member of class TCustomRibbonGroup.
	MouseMove (see page 6782)	This is MouseMove, a member of class TCustomRibbonGroup.
	MouseUp (see page 6783)	This is MouseUp, a member of class TCustomRibbonGroup.
	Notification (see page 6783)	This is Notification, a member of class TCustomRibbonGroup.
	Paint (see page 6783)	This is Paint, a member of class TCustomRibbonGroup.
	Resize (see page 6783)	This is Resize, a member of class TCustomRibbonGroup.
	SetAutoSizing (see page 6783)	This is SetAutoSizing, a member of class TCustomRibbonGroup.
	SetParent (see page 6784)	This is SetParent, a member of class TCustomRibbonGroup.
	SetupDropDownBtn (see page 6785)	This is SetupDropDownBtn, a member of class TCustomRibbonGroup.
	ShowControls (see page 6785)	This is ShowControls, a member of class TCustomRibbonGroup.

Public Methods

	Name	Description
	AlignGroupControls (see page 6775)	Align controls in Ribbon (see page 6713) Group
	Assign (see page 6775)	This is Assign, a member of class TCustomRibbonGroup.
	BarHandleKeyTips (see page 6776)	Iterate over any visible **KeyTip (see page 6788)**s and perform validation to determine if one of **KeyTip (see page 6788)**s is partial or full match.
	BarHideKeyTips (see page 6776)	Hide any visible KeyTips that commands are displaying.
	BarShowKeyTips (see page 6777)	Show K**eyTip**s for commands.
	Clone (see page 6777)	This is Clone, a member of class TCustomRibbonGroup.
	CreateControl (see page 6778)	Recreate specific control.
	DockDrop (see page 6779)	This is DockDrop, a member of class TCustomRibbonGroup.
	DragDrop (see page 6779)	This is DragDrop, a member of class TCustomRibbonGroup.
	GetKeyTip (see page 6780)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 6781)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6781)	Hide **KeyTip** for command.
	KeyTipClick (see page 6782)	Handle clicking a **KeyTip**.
	SetBounds (see page 6784)	This is SetBounds, a member of class TCustomRibbonGroup.
	SetKeyTip (see page 6784)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6785)	Shows **KeyTip** for command in specified position.
	StyleChanged (see page 6785)	This is StyleChanged, a member of class TCustomRibbonGroup.

76.1.6.5.1 TCustomRibbonGroup.AlignControls Method

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

This is AlignControls, a member of class TCustomRibbonGroup.

76.1.6.5.2 TCustomRibbonGroup.AlignGroupControls Method

Align controls in Ribbon (see page 6713) Group

Pascal

```
procedure AlignGroupControls; virtual;
```

C++

```
virtual __fastcall AlignGroupControls();
```

Description

AlignGroupControls aligns the controls in the Ribbon (see page 6713) Group.

76.1.6.5.3 TCustomRibbonGroup.Assign Method

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

This is Assign, a member of class TCustomRibbonGroup.

76.1.6.5.4 TCustomRibbonGroup.AutoSizingChanged Method

Pascal

```
procedure AutoSizingChanged; override;
```

C++

```
virtual __fastcall AutoSizingChanged();
```

Description

This is AutoSizingChanged, a member of class TCustomRibbonGroup.

76.1.6.5.5 TCustomRibbonGroup.BarHandleKeyTips Method

Iterate over any visible **KeyTip (see page 6788)**s and perform validation to determine if one of **KeyTip (see page 6788)**s is partial or full match.

Pascal

```
procedure BarHandleKeyTips(const KeyTip: string; var KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHandleKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

BarHandleKeyTips iterates over any visible **KeyTip (see page 6788)**s and performs validation to determine if one of the **KeyTip (see page 6788)**s is a partial or full match. **KeyTip (see page 6788)**s that match are returned in the **List** parameter.

See Also

BarHideKeyTips (see page 6776)

BarShowKeyTips (see page 6777)

76.1.6.5.6 TCustomRibbonGroup.BarHideKeyTips Method

Hide any visible KeyTips that commands are displaying.

Pascal

```
procedure BarHideKeyTips(KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHideKeyTips(TKeyTipList KeyTips);
```

Description

BarHideKeyTips hides any visible KeyTips that commands are displaying. The **KeyTips** parameter contains a list of TKeyTipMatchItem (see page 6962) objects that indicate the **KeyTip (see page 6788)** associate with that command should not be close. When the **KeyTips** parameter is `nil`, all **KeyTip (see page 6788)**s are hidden.

See Also

TKeyTipMatchItem (see page 6962)

BarHandleKeyTips (see page 6776)

BarShowKeyTips (see page 6777)

76.1.6.5.7 TCustomRibbonGroup.BarShowKeyTips Method

Show K**eyTip**s for commands.

Pascal

```
procedure BarShowKeyTips;
```

C++

```
__fastcall BarShowKeyTips();
```

Description

BarShowKeyTips shows **KeyTip (see page 6788)**s for the commands.

See Also

BarHandleKeyTips (see page 6776)

BarHideKeyTips (see page 6776)

76.1.6.5.8 TCustomRibbonGroup.Clone Method

Pascal

```
function Clone: TCustomRibbonGroup;
```

C++

```
__fastcall TCustomRibbonGroup Clone();
```

Description

This is Clone, a member of class TCustomRibbonGroup.

76.1.6.5.9 TCustomRibbonGroup.CMFontchanged Method

Pascal

```
procedure CMFontchanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontchanged(TMessage Message);
```

Description

This is CMFontchanged, a member of class TCustomRibbonGroup.

76.1.6.5.10 TCustomRibbonGroup.CMItemDropPoint Method

Pascal

```
procedure CMItemDropPoint(var Message: TCMItemMsg); message CM_ITEMDROPPOINT;
```

C++

```
__fastcall CMItemDropPoint(TCMItemMsg Message);
```

Description

This is CMItemDropPoint, a member of class TCustomRibbonGroup.

76.1.6.5.11 TCustomRibbonGroup.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TCustomRibbonGroup.

76.1.6.5.12 TCustomRibbonGroup.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TCustomRibbonGroup.

76.1.6.5.13 TCustomRibbonGroup.CMTextchanged Method

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This is CMTextchanged, a member of class TCustomRibbonGroup.

76.1.6.5.14 TCustomRibbonGroup.CreateControl Method

Recreate specific control.

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

CreateControl recreates a specific control.

See Also

AlignGroupControls (see page 6934)

76.1.6.5.15 TCustomRibbonGroup.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomRibbonGroup.

76.1.6.5.16 TCustomRibbonGroup.DockDrop Method

Pascal

```
procedure DockDrop(Source: TDragDockObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DockDrop(TDragDockObject Source, int X, int Y);
```

Description

This is DockDrop, a member of class TCustomRibbonGroup.

76.1.6.5.17 TCustomRibbonGroup.DoDropItem Method

Pascal

```
procedure DoDropItem(Source: TActionItemDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropItem(TActionItemDragObject Source, const int X, const int Y);
```

Description

This is DoDropItem, a member of class TCustomRibbonGroup.

76.1.6.5.18 TCustomRibbonGroup.DoEndDrag Method

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

This is DoEndDrag, a member of class TCustomRibbonGroup.

76.1.6.5.19 TCustomRibbonGroup.DragDrop Method

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

This is DragDrop, a member of class TCustomRibbonGroup.

76.1.6.5.20 TCustomRibbonGroup.DrawShowDialogButton Method

Pascal

```
procedure DrawShowDialogButton; virtual;
```

C++

```
virtual __fastcall DrawShowDialogButton();
```

Description

This is DrawShowDialogButton, a member of class TCustomRibbonGroup.

76.1.6.5.21 TCustomRibbonGroup.GetCaptionHeight Method

Pascal

```
function GetCaptionHeight: Integer;
```

C++

```
__fastcall int GetCaptionHeight();
```

Description

This is GetCaptionHeight, a member of class TCustomRibbonGroup.

76.1.6.5.22 TCustomRibbonGroup.GetGroupHeight Method

Pascal

```
function GetGroupHeight: Integer;
```

C++

```
__fastcall int GetGroupHeight();
```

Description

This is GetGroupHeight, a member of class TCustomRibbonGroup.

76.1.6.5.23 TCustomRibbonGroup.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.6.5.24 TCustomRibbonGroup.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.6.5.25 TCustomRibbonGroup.HideControls Method

Pascal

```
procedure HideControls; virtual;
```

C++

```
virtual __fastcall HideControls();
```

Description

This is HideControls, a member of class TCustomRibbonGroup.

76.1.6.5.26 TCustomRibbonGroup.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.6.5.27 TCustomRibbonGroup.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

76.1.6.5.28 TCustomRibbonGroup.MouseDown Method

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseDown, a member of class TCustomRibbonGroup.

76.1.6.5.29 TCustomRibbonGroup.MouseMove Method

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

This is MouseMove, a member of class TCustomRibbonGroup.

76.1.6.5.30 TCustomRibbonGroup.MouseUp Method

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseUp, a member of class TCustomRibbonGroup.

76.1.6.5.31 TCustomRibbonGroup.Notification Method

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This is Notification, a member of class TCustomRibbonGroup.

76.1.6.5.32 TCustomRibbonGroup.Paint Method

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

This is Paint, a member of class TCustomRibbonGroup.

76.1.6.5.33 TCustomRibbonGroup.Resize Method

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

This is Resize, a member of class TCustomRibbonGroup.

76.1.6.5.34 TCustomRibbonGroup.SetAutoSizing Method

Pascal

```
procedure SetAutoSizing(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSizing(const Boolean Value);
```

Description

This is SetAutoSizing, a member of class TCustomRibbonGroup.

76.1.6.5.35 TCustomRibbonGroup.SetBounds Method

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

This is SetBounds, a member of class TCustomRibbonGroup.

76.1.6.5.36 TCustomRibbonGroup.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

76.1.6.5.37 TCustomRibbonGroup.SetParent Method

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

This is SetParent, a member of class TCustomRibbonGroup.

76.1.6.5.38 TCustomRibbonGroup.SetupDropDownBtn Method

Pascal

```
procedure SetupDropDownBtn; override;
```

C++

```
virtual __fastcall SetupDropDownBtn();
```

Description

This is SetupDropDownBtn, a member of class TCustomRibbonGroup.

76.1.6.5.39 TCustomRibbonGroup.ShowControls Method

Pascal

```
procedure ShowControls; virtual;
```

C++

```
virtual __fastcall ShowControls();
```

Description

This is ShowControls, a member of class TCustomRibbonGroup.

76.1.6.5.40 TCustomRibbonGroup.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

76.1.6.5.41 TCustomRibbonGroup.StyleChanged Method

Pascal

```
procedure StyleChanged(AStyle: TRibbonStyleActionBars); virtual;
```

C++

```
virtual __fastcall StyleChanged(TRibbonStyleActionBars AStyle);
```

Description

This is StyleChanged, a member of class TCustomRibbonGroup.

76.1.6.6 TCustomRibbonGroup Properties

The properties of the TCustomRibbonGroup class are listed here.

Public Properties

	Name	Description
R	CanCollapse (see page 6786)	Determine whether group can be displayed collapsed.
R	CollapsedControl (see page 6786)	This is CollapsedControl, a member of class TCustomRibbonGroup.
	DialogAction (see page 6786)	Determines whether dialog button displayed in group.
	GroupAlign (see page 6787)	Determines how controls on group are aligned.
	GroupIndex (see page 6787)	Index of group within current tab.
	GroupVariant (see page 6787)	Current size variant ribbon group is using.
R	GroupWidth (see page 6788)	This is GroupWidth, a member of class TCustomRibbonGroup.
R	HasGallery (see page 6788)	Indicates if group has an in ribbon Gallery.
	Items (see page 6788)	This is Items, a member of class TCustomRibbonGroup.
	KeyTip (see page 6788)	Key tip for Dialog button of group.
R	Page (see page 6789)	Reference to Page group is on.
R	RequiredWidth (see page 6789)	Width required to display group.
	Rows (see page 6789)	Number of Rows in group has for controls.

76.1.6.6.1 TCustomRibbonGroup.CanCollapse Property

Determine whether group can be displayed collapsed.

Pascal

```
property CanCollapse: Boolean;
```

C++

```
__property Boolean CanCollapse;
```

Description

CanCollapse determines whether the group can be displayed in the collapsed state. If there is only one control on the group, that group cannot be displayed as collapsed.

76.1.6.6.2 TCustomRibbonGroup.CollapsedControl Property

Pascal

```
property CollapsedControl: TRibbonGroupCollapsedControl;
```

C++

```
__property TRibbonGroupCollapsedControl CollapsedControl;
```

Description

This is CollapsedControl, a member of class TCustomRibbonGroup.

76.1.6.6.3 TCustomRibbonGroup.DialogAction Property

Determines whether dialog button displayed in group.

Pascal

```
property DialogAction: TContainedAction;
```

C++

```
__property TContainedAction DialogAction;
```

Description

DialogAction determines whether the dialog button is displayed in the lower right hand corner of the group.

See Also

KeyTip (see page 6788)

76.1.6.6.4 TCustomRibbonGroup.GroupAlign Property

Determines how controls on group are aligned.

Pascal

```
property GroupAlign: TGroupAlign;
```

C++

```
__property TGroupAlign GroupAlign;
```

Description

GroupAlign contains a TGroupAlign (see page 6774) that determines how controls on a group are aligned.

If alignment is **gaHorizontal**, controls are aligned left to right, top to bottom. To make a control start in a new row, set the `NewRow` property to true.

If alignment is **gaVertical**, controls are aligned top to bottom, left to right. Controls are automatically moved to the next column in the group if three controls are already in the column. You can also force a control to a new column by setting the `NewCol` property to true.

See Also

TGroupAlign

76.1.6.6.5 TCustomRibbonGroup.GroupIndex Property

Index of group within current tab.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

GroupIndex is the index of the group within the current tab. Set the GroupIndex property to change the position of the group within the current tab.

76.1.6.6.6 TCustomRibbonGroup.GroupVariant Property

Current size variant ribbon group is using.

Pascal

```
property GroupVariant: TRibbonGroupVariant;
```

C++

```
__property TRibbonGroupVariant GroupVariant;
```

Description

GroupVariant is the current size variant the ribbon group is using.

See Also

RequiredWidth (see page 6789)

76.1.6.6.7 TCustomRibbonGroup.GroupWidth Property

Pascal

```
property GroupWidth: Integer;
```

C++

```
__property int GroupWidth;
```

Description

This is GroupWidth, a member of class TCustomRibbonGroup.

76.1.6.6.8 TCustomRibbonGroup.HasGallery Property

Indicates if group has an in ribbon Gallery.

Pascal

```
property HasGallery: Boolean;
```

C++

```
__property Boolean HasGallery;
```

Description

HasGallery returns True if the group has an in ribbon Gallery to display, false otherwise.

76.1.6.6.9 TCustomRibbonGroup.Items Property

Pascal

```
property Items;
```

C++

```
__property Items;
```

Description

This is Items, a member of class TCustomRibbonGroup.

76.1.6.6.10 TCustomRibbonGroup.KeyTip Property

Key tip for Dialog button of group.

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

KeyTip is the key tip for the Dialog button of the group. It is only valid when DialogAction (see page 6786) has been set.

See Also

DialogAction (see page 6786)

76.1.6.6.11 TCustomRibbonGroup.Page Property

Reference to Page group is on.

Pascal

```
property Page: TRibbonPage;
```

C++

```
__property TRibbonPage Page;
```

Description

Page is a reference to the Page that the group is on.

76.1.6.6.12 TCustomRibbonGroup.RequiredWidth Property

Width required to display group.

Pascal

```
property RequiredWidth: Integer;
```

C++

```
__property int RequiredWidth;
```

Description

RequiredWidth is the width required to display the group in its large Variant.

See Also

GroupVariant (see page 6787)

76.1.6.6.13 TCustomRibbonGroup.Rows Property

Number of Rows in group has for controls.

Pascal

```
property Rows: Integer;
```

C++

```
__property int Rows;
```

Description

Rows is the number of Rows the group has for displaying controls. A group must have at least 1 row and a maximum of 3.

76.1.6.7 TCustomRibbonGroup Nested Types

The nested types of the TCustomRibbonGroup class are listed here.

Public Nested Types

Name	Description
TRowPosition (see page 6790)	This is nested type Ribbon.TCustomRibbonGroup.TRowPosition.

76.1.6.7.1 TCustomRibbonGroup.TRowPosition Nested Type

Pascal

```
TRowPosition = array [1..3] of Integer;
```

C++

```
array [1..3] of Integer TRowPosition;
```

Description

This is nested type Ribbon.TCustomRibbonGroup.TRowPosition.

76.1.7 TCustomRibbonPage Class

Holds groups of controls.

Class Hierarchy

Pascal

```
TCustomRibbonPage = class(TCustomControl);
```

C++

```
class TCustomRibbonPage : public TCustomControl;
```

File

Ribbon

Description

TCustomRibbonPage holds the groups of controls that are displayed.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

76.1.7.1 TCustomRibbonPage Members

The following tables list the members exposed by TCustomRibbonPage.

Protected Methods

	Name	Description
	CMControlListChanging (see page 6795)	This is CMControlListChanging, a member of class TCustomRibbonPage.
	CMTextchanged (see page 6795)	This is CMTextchanged, a member of class TCustomRibbonPage.
V	CreateScrollButton (see page 6795)	This is CreateScrollButton, a member of class TCustomRibbonPage.
V	DoScrollButtonClick (see page 6795)	This is DoScrollButtonClick, a member of class TCustomRibbonPage.
	GetPageHeight (see page 6797)	This is GetPageHeight, a member of class TCustomRibbonPage.
	GetPageTop (see page 6797)	This is GetPageTop, a member of class TCustomRibbonPage.
V	Loaded (see page 6798)	Initializes the control after it is loaded from a stream.
V	Notification (see page 6798)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 6799)	Renders the image of a custom control.
V	RequestAlign (see page 6799)	Instructs the parent of a control to reposition the control, enforcing its Align property.
V	Resize (see page 6800)	Respond to control resize.
V	SetParent (see page 6801)	Sets the parent of the control.
	UpdateScrollButtons (see page 6802)	This is UpdateScrollButtons, a member of class TCustomRibbonPage.

Protected Properties

	Name	Description
	GroupOffset (see page 6803)	This is GroupOffset, a member of class TCustomRibbonPage.

Public Constants

	Name	Description
	cRibbonGroupLeftOffset (see page 6793)	This is cRibbonGroupLeftOffset, a member of class TCustomRibbonPage.
	cRibbonGroupSpacing (see page 6793)	This is cRibbonGroupSpacing, a member of class TCustomRibbonPage.

Public Methods

	Name	Description
	AddGroup (see page 6794)	This is AddGroup, a member of class TCustomRibbonPage.
	AlignGroups (see page 6794)	This is AlignGroups, a member of class TCustomRibbonPage.
	CollapseGroup (see page 6795)	This is CollapseGroup, a member of class TCustomRibbonPage.
V	Create (see page 6792)	Creates an instance of TCustomControl.
V	Destroy (see page 6792)	Destroys an instance of TCustomControl.
	FindGroup (see page 6796)	This is FindGroup, a member of class TCustomRibbonPage.
	FindGroupAt (see page 6796)	This is FindGroupAt, a member of class TCustomRibbonPage.
	GetGroupPos (see page 6796)	This is GetGroupPos, a member of class TCustomRibbonPage.
	GetNextVisibleGroup (see page 6796)	This is GetNextVisibleGroup, a member of class TCustomRibbonPage.
	HandleKeyTips (see page 6797)	This is HandleKeyTips, a member of class TCustomRibbonPage.
	HideKeyTips (see page 6797)	This is HideKeyTips, a member of class TCustomRibbonPage.
V	Invalidate (see page 6797)	Schedules a control repaint.
	RibbonStyle (see page 6800)	Return style class to use when drawing specific Ribbon (see page 6804) elements.
V	SetBounds (see page 6800)	Sets the windowed control's boundary properties all at once.
	ShowKeyTips (see page 6802)	This is ShowKeyTips, a member of class TCustomRibbonPage.
	UpdateActionManager (see page 6802)	This is UpdateActionManager, a member of class TCustomRibbonPage.

Public Properties

	Name	Description
R	CollapseGroupCount (see page 6803)	This is CollapseGroupCount, a member of class TCustomRibbonPage.
R	GroupCount (see page 6803)	This is GroupCount, a member of class TCustomRibbonPage.
R	Groups (see page 6803)	This is Groups, a member of class TCustomRibbonPage.
	Index (see page 6803)	This is Index, a member of class TCustomRibbonPage.
	KeyTip (see page 6804)	This is KeyTip, a member of class TCustomRibbonPage.
	Ribbon (see page 6804)	This is Ribbon, a member of class TCustomRibbonPage.

76.1.7.2 TCustomRibbonPage.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

76.1.7.3 TCustomRibbonPage.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRibbonPage();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free

delete

Create

76.1.7.4 TCustomRibbonPage Constants

The constants of the TCustomRibbonPage class are listed here.

Public Constants

	Name	Description
	cRibbonGroupLeftOffset (see page 6793)	This is cRibbonGroupLeftOffset, a member of class TCustomRibbonPage.
	cRibbonGroupSpacing (see page 6793)	This is cRibbonGroupSpacing, a member of class TCustomRibbonPage.

76.1.7.4.1 TCustomRibbonPage.cRibbonGroupLeftOffset Constant

Pascal

```
const cRibbonGroupLeftOffset = 4;
```

C++

```
cRibbonGroupLeftOffset = 4;
```

Description

This is cRibbonGroupLeftOffset, a member of class TCustomRibbonPage.

76.1.7.4.2 TCustomRibbonPage.cRibbonGroupSpacing Constant

Pascal

```
const cRibbonGroupSpacing = 2;
```

C++

```
cRibbonGroupSpacing = 2;
```

Description

This is cRibbonGroupSpacing, a member of class TCustomRibbonPage.

76.1.7.5 TCustomRibbonPage Methods

The methods of the TCustomRibbonPage class are listed here.

Protected Methods

	Name	Description
	CMControlListChanging (see page 6795)	This is CMControlListChanging, a member of class TCustomRibbonPage.
	CMTextchanged (see page 6795)	This is CMTextchanged, a member of class TCustomRibbonPage.
	CreateScrollButton (see page 6795)	This is CreateScrollButton, a member of class TCustomRibbonPage.

	DoScrollButtonClick (see page 6795)	This is DoScrollButtonClick, a member of class TCustomRibbonPage.
	GetPageHeight (see page 6797)	This is GetPageHeight, a member of class TCustomRibbonPage.
	GetPageTop (see page 6797)	This is GetPageTop, a member of class TCustomRibbonPage.
	Loaded (see page 6798)	Initializes the control after it is loaded from a stream.
	Notification (see page 6798)	Responds to notifications that components are being created or destroyed.
	Paint (see page 6799)	Renders the image of a custom control.
	RequestAlign (see page 6799)	Instructs the parent of a control to reposition the control, enforcing its Align property.
	Resize (see page 6800)	Respond to control resize.
	SetParent (see page 6801)	Sets the parent of the control.
	UpdateScrollButtons (see page 6802)	This is UpdateScrollButtons, a member of class TCustomRibbonPage.

Public Methods

	Name	Description
	AddGroup (see page 6794)	This is AddGroup, a member of class TCustomRibbonPage.
	AlignGroups (see page 6794)	This is AlignGroups, a member of class TCustomRibbonPage.
	CollapseGroup (see page 6795)	This is CollapseGroup, a member of class TCustomRibbonPage.
	FindGroup (see page 6796)	This is FindGroup, a member of class TCustomRibbonPage.
	FindGroupAt (see page 6796)	This is FindGroupAt, a member of class TCustomRibbonPage.
	GetGroupPos (see page 6796)	This is GetGroupPos, a member of class TCustomRibbonPage.
	GetNextVisibleGroup (see page 6796)	This is GetNextVisibleGroup, a member of class TCustomRibbonPage.
	HandleKeyTips (see page 6797)	This is HandleKeyTips, a member of class TCustomRibbonPage.
	HideKeyTips (see page 6797)	This is HideKeyTips, a member of class TCustomRibbonPage.
	Invalidate (see page 6797)	Schedules a control repaint.
	RibbonStyle (see page 6800)	Return style class to use when drawing specific Ribbon (see page 6804) elements.
	SetBounds (see page 6800)	Sets the windowed control's boundary properties all at once.
	ShowKeyTips (see page 6802)	This is ShowKeyTips, a member of class TCustomRibbonPage.
	UpdateActionManager (see page 6802)	This is UpdateActionManager, a member of class TCustomRibbonPage.

76.1.7.5.1 TCustomRibbonPage.AddGroup Method

Pascal

```
procedure AddGroup(const Group: TCustomRibbonGroup);
```

C++

```
__fastcall AddGroup(const TCustomRibbonGroup Group);
```

Description

This is AddGroup, a member of class TCustomRibbonPage.

76.1.7.5.2 TCustomRibbonPage.AlignGroups Method

Pascal

```
procedure AlignGroups;
```

C++

```
__fastcall AlignGroups();
```

Description

This is AlignGroups, a member of class TCustomRibbonPage.

76.1.7.5.3 TCustomRibbonPage.CMControlListChanging Method

Pascal

```
procedure CMControlListChanging(var Message: TCMControlListChanging); message
CM_CONTROLLISTCHANGING;
```

C++

```
__fastcall CMControlListChanging(TCMControlListChanging Message);
```

Description

This is CMControlListChanging, a member of class TCustomRibbonPage.

76.1.7.5.4 TCustomRibbonPage.CMTextchanged Method

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This is CMTextchanged, a member of class TCustomRibbonPage.

76.1.7.5.5 TCustomRibbonPage.CollapseGroup Method

Pascal

```
procedure CollapseGroup(const Group: TCustomRibbonGroup);
```

C++

```
__fastcall CollapseGroup(const TCustomRibbonGroup Group);
```

Description

This is CollapseGroup, a member of class TCustomRibbonPage.

76.1.7.5.6 TCustomRibbonPage.CreateScrollButton Method

Pascal

```
function CreateScrollButton(const Direction: TScrollDirection): TRibbonGroupScrollButton;
virtual;
```

C++

```
virtual __fastcall TRibbonGroupScrollButton CreateScrollButton(const TScrollDirection
Direction);
```

Description

This is CreateScrollButton, a member of class TCustomRibbonPage.

76.1.7.5.7 TCustomRibbonPage.DoScrollButtonClick Method

Pascal

```
procedure DoScrollButtonClick(const Direction: TScrollDirection); virtual;
```

C++

```
virtual __fastcall DoScrollButtonClick(const TScrollDirection Direction);
```

Description

This is DoScrollButtonClick, a member of class TCustomRibbonPage.

76.1.7.5.8 TCustomRibbonPage.FindGroup Method

Pascal

```
function FindGroup(const GroupCaption: string): TControl;
```

C++

```
__fastcall TControl * FindGroup(const UnicodeString GroupCaption);
```

Description

This is FindGroup, a member of class TCustomRibbonPage.

76.1.7.5.9 TCustomRibbonPage.FindGroupAt Method

Pascal

```
function FindGroupAt(const Pt: TPoint): TCustomRibbonGroup;
```

C++

```
__fastcall TCustomRibbonGroup FindGroupAt(const TPoint Pt);
```

Description

This is FindGroupAt, a member of class TCustomRibbonPage.

76.1.7.5.10 TCustomRibbonPage.GetGroupPos Method

Pascal

```
function GetGroupPos(Ignore: TControl): Integer;
```

C++

```
__fastcall int GetGroupPos(TControl * Ignore);
```

Description

This is GetGroupPos, a member of class TCustomRibbonPage.

76.1.7.5.11 TCustomRibbonPage.GetNextVisibleGroup Method

Pascal

```
function GetNextVisibleGroup: TCustomRibbonGroup;
```

C++

```
__fastcall TCustomRibbonGroup GetNextVisibleGroup();
```

Description

This is GetNextVisibleGroup, a member of class TCustomRibbonPage.

76.1.7.5.12 TCustomRibbonPage.GetPageHeight Method

Pascal

```
function GetPageHeight: Integer;
```

C++

```
__fastcall int GetPageHeight();
```

Description

This is GetPageHeight, a member of class TCustomRibbonPage.

76.1.7.5.13 TCustomRibbonPage.GetPageTop Method

Pascal

```
function GetPageTop: Integer;
```

C++

```
__fastcall int GetPageTop();
```

Description

This is GetPageTop, a member of class TCustomRibbonPage.

76.1.7.5.14 TCustomRibbonPage.HandleKeyTips Method

Pascal

```
procedure HandleKeyTips(const KeyTip: string; var KeyTips: TKeyTipList);
```

C++

```
__fastcall HandleKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

This is HandleKeyTips, a member of class TCustomRibbonPage.

76.1.7.5.15 TCustomRibbonPage.HideKeyTips Method

Pascal

```
procedure HideKeyTips(KeyTips: TKeyTipList);
```

C++

```
__fastcall HideKeyTips(TKeyTipList KeyTips);
```

Description

This is HideKeyTips, a member of class TCustomRibbonPage.

76.1.7.5.16 TCustomRibbonPage.Invalidate Method

Schedules a control repaint.

Pascal

```
procedure Invalidate; override;
```

C++

```
virtual __fastcall Invalidate();
```

Description

Invalidate informs a control that its entire surface needs to be repainted. Calling Invalidate can prevent flicker caused by a series of partial repaints. There is no performance penalty for calling Invalidate multiple times before the control is actually repainted.

The actual repaint does not occur until the control is updated. To force an immediate repaint, call Repaint instead.

See Also

Repaint

Update

76.1.7.5.17 TCustomRibbonPage.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

76.1.7.5.18 TCustomRibbonPage.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other

objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

76.1.7.5.19 TCustomRibbonPage.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

76.1.7.5.20 TCustomRibbonPage.RequestAlign Method

Instructs the parent of a control to reposition the control, enforcing its Align property.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

Controls call RequestAlign internally when changes are made to the size or position of the control, so that they remain properly aligned within the parent control. Do not call RequestAlign in application code.

See Also

Align

Parent

76.1.7.5.21 TCustomRibbonPage.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize

76.1.7.5.22 TCustomRibbonPage.RibbonStyle Method

Return style class to use when drawing specific Ribbon (see page 6804) elements.

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

RibbonStyle returns the style class to use when drawing the specific Ribbon (see page 6804) elements.

See Also

Chapter Title

Procedure Title

ClassName

PropertyName

MethodName

EventName

RoutineName

TypeName

VariableName

ConstantName

76.1.7.5.23 TCustomRibbonPage.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
```

```
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

76.1.7.5.24 TCustomRibbonPage.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

76.1.7.5.25 TCustomRibbonPage.ShowKeyTips Method

Pascal

```
procedure ShowKeyTips;
```

C++

```
__fastcall ShowKeyTips();
```

Description

This is ShowKeyTips, a member of class TCustomRibbonPage.

76.1.7.5.26 TCustomRibbonPage.UpdateActionManager Method

Pascal

```
procedure UpdateActionManager(Value: TCustomActionManager);
```

C++

```
__fastcall UpdateActionManager(TCustomActionManager Value);
```

Description

This is UpdateActionManager, a member of class TCustomRibbonPage.

76.1.7.5.27 TCustomRibbonPage.UpdateScrollButtons Method

Pascal

```
procedure UpdateScrollButtons;
```

C++

```
__fastcall UpdateScrollButtons();
```

Description

This is UpdateScrollButtons, a member of class TCustomRibbonPage.

76.1.7.6 TCustomRibbonPage Properties

The properties of the TCustomRibbonPage class are listed here.

Protected Properties

	Name	Description
	GroupOffset (see page 6803)	This is GroupOffset, a member of class TCustomRibbonPage.

Public Properties

	Name	Description
R	CollapseGroupCount (see page 6803)	This is CollapseGroupCount, a member of class TCustomRibbonPage.
R	GroupCount (see page 6803)	This is GroupCount, a member of class TCustomRibbonPage.
R	Groups (see page 6803)	This is Groups, a member of class TCustomRibbonPage.
	Index (see page 6803)	This is Index, a member of class TCustomRibbonPage.
	KeyTip (see page 6804)	This is KeyTip, a member of class TCustomRibbonPage.
	Ribbon (see page 6804)	This is Ribbon, a member of class TCustomRibbonPage.

76.1.7.6.1 TCustomRibbonPage.CollapseGroupCount Property

Pascal

```
property CollapseGroupCount: Integer;
```

C++

```
__property int CollapseGroupCount;
```

Description

This is CollapseGroupCount, a member of class TCustomRibbonPage.

76.1.7.6.2 TCustomRibbonPage.GroupCount Property

Pascal

```
property GroupCount: Integer;
```

C++

```
__property int GroupCount;
```

Description

This is GroupCount, a member of class TCustomRibbonPage.

76.1.7.6.3 TCustomRibbonPage.GroupOffset Property

Pascal

```
property GroupOffset: Integer;
```

C++

```
__property int GroupOffset;
```

Description

This is GroupOffset, a member of class TCustomRibbonPage.

76.1.7.6.4 TCustomRibbonPage.Groups Property

Pascal

```
property Groups [Index: Integer]: TCustomRibbonGroup;
```

C++

```
__property TCustomRibbonGroup Groups[int Index];
```

Description

This is Groups, a member of class TCustomRibbonPage.

76.1.7.6.5 TCustomRibbonPage.Index Property

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

This is Index, a member of class TCustomRibbonPage.

76.1.7.6.6 TCustomRibbonPage.KeyTip Property

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

This is KeyTip, a member of class TCustomRibbonPage.

76.1.7.6.7 TCustomRibbonPage.Ribbon Property

Pascal

```
property Ribbon: TCustomRibbon;
```

C++

```
__property TCustomRibbon Ribbon;
```

Description

This is Ribbon, a member of class TCustomRibbonPage.

76.1.8 TCustomRibbonQuickAccessPopupToolbar Class

Class Hierarchy

Pascal

```
TCustomRibbonQuickAccessPopupToolbar = class(TCustomActionPopupMenu);
```

C++

```
class TCustomRibbonQuickAccessPopupToolbar : public TCustomActionPopupMenu;
```

File

Ribbon

Description

This is class Ribbon.TCustomRibbonQuickAccessPopupToolbar.

76.1.8.1 TCustomRibbonQuickAccessPopupToolbar Members

The following tables list the members exposed by TCustomRibbonQuickAccessPopupToolbar.

Protected Methods

	Name	Description
	CreateWnd (see page 6806)	Creates the underlying window.
	DrawBackground (see page 6806)	This is DrawBackground, a member of class TCustomRibbonQuickAccessPopupToolbar.
	GetControlClass (see page 6806)	This is GetControlClass, a member of class TCustomRibbonQuickAccessPopupToolbar.
	NCPaint (see page 6807)	This is NCPaint, a member of class TCustomRibbonQuickAccessPopupToolbar.
	ValidOrientations (see page 6807)	This is ValidOrientations, a member of class TCustomRibbonQuickAccessPopupToolbar.

Public Methods

	Name	Description
	Create (see page 6805)	Creates and initializes a TCustomActionPopupMenu instance.
	CreateControl (see page 6806)	This is CreateControl, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.2 TCustomRibbonQuickAccessPopupToolbar.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

76.1.8.3 TCustomRibbonQuickAccessPopupToolbar Methods

The methods of the TCustomRibbonQuickAccessPopupToolbar class are listed here.

Protected Methods

	Name	Description
	CreateWnd (see page 6806)	Creates the underlying window.
	DrawBackground (see page 6806)	This is DrawBackground, a member of class TCustomRibbonQuickAccessPopupToolbar.
	GetControlClass (see page 6806)	This is GetControlClass, a member of class TCustomRibbonQuickAccessPopupToolbar.
	NCPaint (see page 6807)	This is NCPaint, a member of class TCustomRibbonQuickAccessPopupToolbar.
	ValidOrientations (see page 6807)	This is ValidOrientations, a member of class TCustomRibbonQuickAccessPopupToolbar.

Public Methods

	Name	Description
	CreateControl (see page 6806)	This is CreateControl, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.3.1 TCustomRibbonQuickAccessPopupToolbar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.3.2 TCustomRibbonQuickAccessPopupToolbar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

76.1.8.3.3 TCustomRibbonQuickAccessPopupToolbar.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.3.4 TCustomRibbonQuickAccessPopupToolbar.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.3.5 TCustomRibbonQuickAccessPopupToolbar.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.8.3.6 TCustomRibbonQuickAccessPopupToolbar.ValidOrientations Method

Pascal

```
function ValidOrientations: TBarOrientations; override;
```

C++

```
virtual __fastcall TBarOrientations ValidOrientations();
```

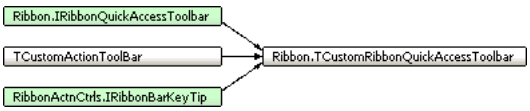
Description

This is ValidOrientations, a member of class TCustomRibbonQuickAccessPopupToolbar.

76.1.9 TCustomRibbonQuickAccessToolbar Class

Quck Access Toolbar base class.

Class Hierarchy



Pascal

```
TCustomRibbonQuickAccessToolbar = class(TCustomActionToolBar, IRibbonQuickAccessToolbar,
IRibbonBarKeyTip);
```

C++

```
class TCustomRibbonQuickAccessToolbar : public TCustomActionToolBar, public
IRibbonQuickAccessToolbar, public IRibbonBarKeyTip;
```

File

Ribbon

Description

TCustomRibbonQuickAccessToolbar is the base class Quck Access Toolbar.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TRibbonQuickAccessToolbar (see page 6911)

76.1.9.1 TCustomRibbonQuickAccessToolbar Members

The following tables list the members exposed by TCustomRibbonQuickAccessToolbar.

Protected Methods

	Name	Description
	CMFontchanged (see page 6811)	This is CMFontchanged, a member of class TCustomRibbonQuickAccessToolbar.
V	CreateControls (see page 6811)	This is CreateControls, a member of class TCustomRibbonQuickAccessToolbar.
V	DoContextPopup (see page 6812)	OnContextPopup event dispatcher.
V	DoDropActions (see page 6812)	This is DoDropActions, a member of class TCustomRibbonQuickAccessToolbar.
V	DoDropItem (see page 6813)	This is DoDropItem, a member of class TCustomRibbonQuickAccessToolbar.
V	DoMoreCommands (see page 6813)	This is DoMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.
V	DoScrollBtnClick (see page 6813)	This is DoScrollBtnClick, a member of class TCustomRibbonQuickAccessToolbar.
V	DrawBackground (see page 6814)	This is DrawBackground, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarHeight (see page 6814)	This is GetBarHeight, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarWidth (see page 6815)	This is GetBarWidth, a member of class TCustomRibbonQuickAccessToolbar.
V	GetScrollBtnClass (see page 6815)	This is GetScrollBtnClass, a member of class TCustomRibbonQuickAccessToolbar.
V	Notification (see page 6815)	Responds to notifications that components are being created or destroyed.
V	RequestAlign (see page 6815)	Instructs the parent of a control to reposition the control, enforcing its Align property.
V	SetParent (see page 6816)	Sets the parent of the control.
V	SetupDropDownBtn (see page 6816)	This is SetupDropDownBtn, a member of class TCustomRibbonQuickAccessToolbar.
V	ShowCustomizeMenu (see page 6817)	This is ShowCustomizeMenu, a member of class TCustomRibbonQuickAccessToolbar.
V	ShowMoreControls (see page 6817)	This is ShowMoreControls, a member of class TCustomRibbonQuickAccessToolbar.
V	UpdateQATWidth (see page 6817)	This is UpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.

Public Methods

	Name	Description
	BarHandleKeyTips (see page 6810)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 6810)	Hide any visible **KeyTip** windows.
	BarShowKeyTips (see page 6811)	Show **KeyTip**s for any commands that support them.
V	Create (see page 6809)	Creates and initializes a TCustomActionToolBar instance.
V	CreateControl (see page 6811)	This is CreateControl, a member of class TCustomRibbonQuickAccessToolbar.
	Customize (see page 6812)	This is Customize, a member of class TCustomRibbonQuickAccessToolbar.
V	DoDropCategory (see page 6813)	This is DoDropCategory, a member of class TCustomRibbonQuickAccessToolbar.
	DoUpdateQATWidth (see page 6813)	This is DoUpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.
V	DragDrop (see page 6814)	Generates an OnDragDrop event.

Public Nested Types

Name	Description
TMoreCommandsEvent (see page 6819)	This is nested type Ribbon.TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent.

Public Properties

	Name	Description
	Items (see page 6818)	Lists the action clients that correspond to the child controls of this action band.
	OnMoreCommands (see page 6818)	This is OnMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.
R	Ribbon (see page 6818)	This is Ribbon, a member of class TCustomRibbonQuickAccessToolbar.
	ScrollMode (see page 6818)	This is ScrollMode, a member of class TCustomRibbonQuickAccessToolbar.
	ShowBorder (see page 6819)	This is ShowBorder, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.2 TCustomRibbonQuickAccessToolbar.Create Constructor

Creates and initializes a TCustomActionToolBar instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionToolBar and initializes its properties.

```
__fastcall virtual TCustomActionToolBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall (HWND ParentWindow) : Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

76.1.9.3 TCustomRibbonQuickAccessToolbar Methods

The methods of the TCustomRibbonQuickAccessToolbar class are listed here.

Protected Methods

	Name	Description
	CMFontchanged (see page 6811)	This is CMFontchanged, a member of class TCustomRibbonQuickAccessToolbar.
V	CreateControls (see page 6811)	This is CreateControls, a member of class TCustomRibbonQuickAccessToolbar.
V	DoContextPopup (see page 6812)	OnContextPopup event dispatcher.
V	DoDropActions (see page 6812)	This is DoDropActions, a member of class TCustomRibbonQuickAccessToolbar.
V	DoDropItem (see page 6813)	This is DoDropItem, a member of class TCustomRibbonQuickAccessToolbar.
V	DoMoreCommands (see page 6813)	This is DoMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.
V	DoScrollBtnClick (see page 6813)	This is DoScrollBtnClick, a member of class TCustomRibbonQuickAccessToolbar.
V	DrawBackground (see page 6814)	This is DrawBackground, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarHeight (see page 6814)	This is GetBarHeight, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarWidth (see page 6815)	This is GetBarWidth, a member of class TCustomRibbonQuickAccessToolbar.
V	GetScrollBtnClass (see page 6815)	This is GetScrollBtnClass, a member of class TCustomRibbonQuickAccessToolbar.
V	Notification (see page 6815)	Responds to notifications that components are being created or destroyed.
V	RequestAlign (see page 6815)	Instructs the parent of a control to reposition the control, enforcing its Align property.
V	SetParent (see page 6816)	Sets the parent of the control.

	SetupDropDownBtn (see page 6816)	This is SetupDropDownBtn, a member of class TCustomRibbonQuickAccessToolbar.
	ShowCustomizeMenu (see page 6817)	This is ShowCustomizeMenu, a member of class TCustomRibbonQuickAccessToolbar.
	ShowMoreControls (see page 6817)	This is ShowMoreControls, a member of class TCustomRibbonQuickAccessToolbar.
	UpdateQATWidth (see page 6817)	This is UpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.

Public Methods

	Name	Description
	BarHandleKeyTips (see page 6810)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 6810)	Hide any visible **KeyTip** windows.
	BarShowKeyTips (see page 6811)	Show **KeyTip**s for any commands that support them.
	CreateControl (see page 6811)	This is CreateControl, a member of class TCustomRibbonQuickAccessToolbar.
	Customize (see page 6812)	This is Customize, a member of class TCustomRibbonQuickAccessToolbar.
	DoDropCategory (see page 6813)	This is DoDropCategory, a member of class TCustomRibbonQuickAccessToolbar.
	DoUpdateQATWidth (see page 6813)	This is DoUpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.
	DragDrop (see page 6814)	Generates an OnDragDrop event.

76.1.9.3.1 TCustomRibbonQuickAccessToolbar.BarHandleKeyTips Method

Check if any commands on Action Bar are partial or full match for **KeyTip**.

Pascal

```
procedure BarHandleKeyTips(const KeyTip: string; var KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHandleKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

BarHandleKeyTips checks if any commands on the Action Bar are a partial or full match for the **KeyTip**. Matches are returned as a TKeyTipMatchItem in the `List` parameter. If only one item is a match, that command is executed.

See Also

TKeyTipMatchItem (see page 6962)

BarHideKeyTips (see page 7273)

BarShowKeyTips (see page 7274)

76.1.9.3.2 TCustomRibbonQuickAccessToolbar.BarHideKeyTips Method

Hide any visible **KeyTip** windows.

Pascal

```
procedure BarHideKeyTips(KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHideKeyTips(TKeyTipList KeyTips);
```

Description

BarHideKeyTips hides any visible **KeyTip** windows. `ExceptList` may contain a list of TKeyTipMatchItem objects for commands that should not be hidden. These items are current partial matches and require additional handling.

See Also

TKeyTipMatchItem (see page 6962)

BarHandleKeyTips (see page 7273)

BarShowKeyTips (see page 7274)

76.1.9.3.3 TCustomRibbonQuickAccessToolbar.BarShowKeyTips Method

Show **KeyTip**s for any commands that support them.

Pascal

```
procedure BarShowKeyTips;
```

C++

```
__fastcall BarShowKeyTips();
```

Description

BarShowKeyTips shows **KeyTip**s for any commands that support them.

See Also

BarHandleKeyTips (see page 7273)

BarHideKeyTips (see page 7273)

76.1.9.3.4 TCustomRibbonQuickAccessToolbar.CMFontchanged Method

Pascal

```
procedure CMFontchanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontchanged(TMessage Message);
```

Description

This is CMFontchanged, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.5 TCustomRibbonQuickAccessToolbar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.6 TCustomRibbonQuickAccessToolbar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.7 TCustomRibbonQuickAccessToolbar.Customize Method

Pascal

```
procedure Customize;
```

C++

```
__fastcall Customize();
```

Description

This is Customize, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.8 TCustomRibbonQuickAccessToolbar.DoContextPopup Method

OnContextPopup event dispatcher.

Pascal

```
procedure DoContextPopup(MousePos: TPoint; var Handled: Boolean); override;
```

C++

```
virtual __fastcall DoContextPopup(TPoint MousePos, Boolean Handled);
```

Description

OnContextPopup calls any event handler attached to the OnContextPopup event. When creating a component that descends from TControl, override OnContextPopup to provide other responses in addition to the inherited event-handler call when the user invokes the control's context menu.

MousePos indicates the position where the user right-clicked the control. If the user invoked the context menu using the keyboard, MousePos is (-1,-1).

Handled returns a Boolean value indicating whether the control should continue with its default response to the user action.

See Also

OnContextPopup

76.1.9.3.9 TCustomRibbonQuickAccessToolbar.DoDropActions Method

Pascal

```
procedure DoDropActions(Source: TActionDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropActions(TActionDragObject Source, const int X, const int Y);
```

Description

This is DoDropActions, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.10 TCustomRibbonQuickAccessToolbar.DoDropCategory Method

Pascal

```
procedure DoDropCategory(Source: TCategoryDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropCategory(TCategoryDragObject Source, const int X, const int Y);
```

Description

This is DoDropCategory, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.11 TCustomRibbonQuickAccessToolbar.DoDropItem Method

Pascal

```
procedure DoDropItem(Source: TActionItemDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropItem(TActionItemDragObject Source, const int X, const int Y);
```

Description

This is DoDropItem, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.12 TCustomRibbonQuickAccessToolbar.DoMoreCommands Method

Pascal

```
procedure DoMoreCommands; virtual;
```

C++

```
virtual __fastcall DoMoreCommands();
```

Description

This is DoMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.13 TCustomRibbonQuickAccessToolbar.DoScrollBtnClick Method

Pascal

```
procedure DoScrollBtnClick; override;
```

C++

```
virtual __fastcall DoScrollBtnClick();
```

Description

This is DoScrollBtnClick, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.14 TCustomRibbonQuickAccessToolbar.DoUpdateQATWidth Method

Pascal

```
procedure DoUpdateQATWidth;
```

C++

```
__fastcall DoUpdateQATWidth();
```

Description

This is DoUpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.15 TCustomRibbonQuickAccessToolbar.DragDrop Method

Generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

DragDrop is called automatically when the user drags and drops a control onto the tool bar. This method generates an OnDragDrop event, and then updates its list of hidden controls to reflect the presence of the new child control.

Source is the dragged control that is dropped onto the tool bar.

X and Y are the mouse coordinates where Source was dropped.

See Also

DragCanceled

OnDragDrop

HiddenCount

76.1.9.3.16 TCustomRibbonQuickAccessToolbar.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.17 TCustomRibbonQuickAccessToolbar.GetBarHeight Method

Pascal

```
function GetBarHeight: Integer; override;
```

C++

```
virtual __fastcall int GetBarHeight();
```

Description

This is GetBarHeight, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.18 TCustomRibbonQuickAccessToolbar.GetBarWidth Method

Pascal

```
function GetBarWidth: Integer; override;
```

C++

```
virtual __fastcall int GetBarWidth();
```

Description

This is GetBarWidth, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.19 TCustomRibbonQuickAccessToolbar.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.20 TCustomRibbonQuickAccessToolbar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

76.1.9.3.21 TCustomRibbonQuickAccessToolbar.RequestAlign Method

Instructs the parent of a control to reposition the control, enforcing its Align property.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

Controls call RequestAlign internally when changes are made to the size or position of the control, so that they remain properly aligned within the parent control. Do not call RequestAlign in application code.

See Also

Align

Parent

76.1.9.3.22 TCustomRibbonQuickAccessToolbar.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

76.1.9.3.23 TCustomRibbonQuickAccessToolbar.SetupDropDownBtn Method

Pascal

```
procedure SetupDropDownBtn; override;
```

C++

```
virtual __fastcall SetupDropDownBtn();
```

Description

This is SetupDropDownBtn, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.24 TCustomRibbonQuickAccessToolbar.ShowCustomizeMenu Method

Pascal

```
procedure ShowCustomizeMenu(AScrollBtn: TRibbonQATScrollButton); virtual;
```

C++

```
virtual __fastcall ShowCustomizeMenu(TRibbonQATScrollButton AScrollBtn);
```

Description

This is ShowCustomizeMenu, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.25 TCustomRibbonQuickAccessToolbar.ShowMoreControls Method

Pascal

```
procedure ShowMoreControls; virtual;
```

C++

```
virtual __fastcall ShowMoreControls();
```

Description

This is ShowMoreControls, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.3.26 TCustomRibbonQuickAccessToolbar.UpdateQATWidth Method

Pascal

```
procedure UpdateQATWidth; virtual;
```

C++

```
virtual __fastcall UpdateQATWidth();
```

Description

This is UpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.4 TCustomRibbonQuickAccessToolbar Properties

The properties of the TCustomRibbonQuickAccessToolbar class are listed here.

Public Properties

	Name	Description
	Items (see page 6818)	Lists the action clients that correspond to the child controls of this action band.
	OnMoreCommands (see page 6818)	This is OnMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.
R	Ribbon (see page 6818)	This is Ribbon, a member of class TCustomRibbonQuickAccessToolbar.
	ScrollMode (see page 6818)	This is ScrollMode, a member of class TCustomRibbonQuickAccessToolbar.
	ShowBorder (see page 6819)	This is ShowBorder, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.4.1 TCustomRibbonQuickAccessToolbar.Items Property

Lists the action clients that correspond to the child controls of this action band.

Pascal

```
property Items;
```

C++

```
__property Items;
```

Description

Items is used internally to access the components that the action manager lists to represent the child controls of this action band. Each action client object in the Items collection corresponds to one of the controls that the ActionControls property lists. The entire collection of items belongs to the action client specified by this action band's ActionClient property.

See Also

ActionControls

ActionClient

ContextBar

76.1.9.4.2 TCustomRibbonQuickAccessToolbar.OnMoreCommands Property

Pascal

```
property OnMoreCommands: TMoreCommandsEvent;
```

C++

```
__property TMoreCommandsEvent OnMoreCommands;
```

Description

This is OnMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.4.3 TCustomRibbonQuickAccessToolbar.Ribbon Property

Pascal

```
property Ribbon: TCustomRibbon;
```

C++

```
__property TCustomRibbon Ribbon;
```

Description

This is Ribbon, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.4.4 TCustomRibbonQuickAccessToolbar.ScrollMode Property

Pascal

```
property ScrollMode: TQATScrollMode;
```

C++

```
__property TQATScrollMode ScrollMode;
```

Description

This is ScrollMode, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.4.5 TCustomRibbonQuickAccessToolbar.ShowBorder Property

Pascal

```
property ShowBorder: Boolean;
```

C++

```
__property Boolean ShowBorder;
```

Description

This is ShowBorder, a member of class TCustomRibbonQuickAccessToolbar.

76.1.9.5 TCustomRibbonQuickAccessToolbar Nested Types

The nested types of the TCustomRibbonQuickAccessToolbar class are listed here.

Public Nested Types

Name	Description
TMoreCommandsEvent (see page 6819)	This is nested type Ribbon.TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent.

76.1.9.5.1 TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent Nested Type

Pascal

```
TMoreCommandsEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( TMoreCommandsEvent)();
```

Description

This is nested type Ribbon.TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent.

76.1.10 TCustomTabData Class

Encapsulates all functions and status to manage scrolling of tabs and selected group.

Class Hierarchy

Pascal

```
TCustomTabData = class;
```

C++

```
class TCustomTabData;
```

File

Ribbon

Description

TCustomTabData encapsulates all the functions and status information needed to manage scrolling tabs and the selected group on a ribbon.

76.1.10.1 TCustomTabData Members

The following tables list the members exposed by TCustomTabData.

Public Methods

	Name	Description
	Add (see page 6821)	This is Add, a member of class TCustomTabData.
	CalculateTabWidths (see page 6821)	This is CalculateTabWidths, a member of class TCustomTabData.
	Clear (see page 6821)	This is Clear, a member of class TCustomTabData.
V	Create (see page 6820)	This is Create, a member of class TCustomTabData.
V	Destroy (see page 6821)	This is Destroy, a member of class TCustomTabData.

Public Properties

	Name	Description
R	AvailableSpace (see page 6822)	This is AvailableSpace, a member of class TCustomTabData.
	FirstVisibleTabIndex (see page 6822)	This is FirstVisibleTabIndex, a member of class TCustomTabData.
R	FullWidth (see page 6822)	This is FullWidth, a member of class TCustomTabData.
R	ItemCount (see page 6823)	This is ItemCount, a member of class TCustomTabData.
R	LastVisibleTabIndex (see page 6823)	This is LastVisibleTabIndex, a member of class TCustomTabData.
R	LeftTabScrollButtonRect (see page 6823)	This is LeftTabScrollButtonRect, a member of class TCustomTabData.
R	NewWidth (see page 6823)	This is NewWidth, a member of class TCustomTabData.
R	Owner (see page 6824)	This is Owner, a member of class TCustomTabData.
R	RightTabScrollButtonRect (see page 6824)	This is RightTabScrollButtonRect, a member of class TCustomTabData.
R	ScrollButtons (see page 6824)	This is ScrollButtons, a member of class TCustomTabData.
R	SeparatorOpacity (see page 6824)	This is SeparatorOpacity, a member of class TCustomTabData.
R	TabSeparatorState (see page 6824)	This is TabSeparatorState, a member of class TCustomTabData.
R	TabUpdateRect (see page 6825)	This is TabUpdateRect, a member of class TCustomTabData.

76.1.10.2 TCustomTabData.Create Constructor

Pascal

```
constructor Create(AOwner: TCustomRibbon); virtual; reintroduce;
```

C++

```
virtual __fastcall TCustomTabData(TCustomRibbon AOwner);
```

Description

This is Create, a member of class TCustomTabData.

76.1.10.3 TCustomTabData.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTabData();
```

Description

This is Destroy, a member of class TCustomTabData.

76.1.10.4 TCustomTabData Methods

The methods of the TCustomTabData class are listed here.

Public Methods

	Name	Description
	Add (see page 6821)	This is Add, a member of class TCustomTabData.
	CalculateTabWidths (see page 6821)	This is CalculateTabWidths, a member of class TCustomTabData.
	Clear (see page 6821)	This is Clear, a member of class TCustomTabData.

76.1.10.4.1 TCustomTabData.Add Method

Pascal

```
procedure Add(ATab: TRibbonTabItem);
```

C++

```
__fastcall Add(TRibbonTabItem ATab);
```

Description

This is Add, a member of class TCustomTabData.

76.1.10.4.2 TCustomTabData.CalculateTabWidths Method

Pascal

```
procedure CalculateTabWidths(AAvailableSpace: Integer; APages: TRibbonTabs);
```

C++

```
__fastcall CalculateTabWidths(int AAvailableSpace, TRibbonTabs APages);
```

Description

This is CalculateTabWidths, a member of class TCustomTabData.

76.1.10.4.3 TCustomTabData.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TCustomTabData.

76.1.10.5 TCustomTabData Properties

The properties of the TCustomTabData class are listed here.

Public Properties

	Name	Description
	AvailableSpace (see page 6822)	This is AvailableSpace, a member of class TCustomTabData.
	FirstVisibleTabIndex (see page 6822)	This is FirstVisibleTabIndex, a member of class TCustomTabData.
	FullWidth (see page 6822)	This is FullWidth, a member of class TCustomTabData.
	ItemCount (see page 6823)	This is ItemCount, a member of class TCustomTabData.
	LastVisibleTabIndex (see page 6823)	This is LastVisibleTabIndex, a member of class TCustomTabData.
	LeftTabScrollButtonRect (see page 6823)	This is LeftTabScrollButtonRect, a member of class TCustomTabData.
	NewWidth (see page 6823)	This is NewWidth, a member of class TCustomTabData.
	Owner (see page 6824)	This is Owner, a member of class TCustomTabData.
	RightTabScrollButtonRect (see page 6824)	This is RightTabScrollButtonRect, a member of class TCustomTabData.
	ScrollButtons (see page 6824)	This is ScrollButtons, a member of class TCustomTabData.
	SeparatorOpacity (see page 6824)	This is SeparatorOpacity, a member of class TCustomTabData.
	TabSeparatorState (see page 6824)	This is TabSeparatorState, a member of class TCustomTabData.
	TabUpdateRect (see page 6825)	This is TabUpdateRect, a member of class TCustomTabData.

76.1.10.5.1 TCustomTabData.AvailableSpace Property

Pascal

```
property AvailableSpace: Integer;
```

C++

```
__property int AvailableSpace;
```

Description

This is AvailableSpace, a member of class TCustomTabData.

76.1.10.5.2 TCustomTabData.FirstVisibleTabIndex Property

Pascal

```
property FirstVisibleTabIndex: Integer;
```

C++

```
__property int FirstVisibleTabIndex;
```

Description

This is FirstVisibleTabIndex, a member of class TCustomTabData.

76.1.10.5.3 TCustomTabData.FullWidth Property

Pascal

```
property FullWidth [Idx: Integer]: Integer;
```

C++

```
__property int FullWidth[int Idx];
```

Description

This is FullWidth, a member of class TCustomTabData.

76.1.10.5.4 TCustomTabData.ItemCount Property

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

This is ItemCount, a member of class TCustomTabData.

76.1.10.5.5 TCustomTabData.LastVisibleTabIndex Property

Pascal

```
property LastVisibleTabIndex: Integer;
```

C++

```
__property int LastVisibleTabIndex;
```

Description

This is LastVisibleTabIndex, a member of class TCustomTabData.

76.1.10.5.6 TCustomTabData.LeftTabScrollButtonRect Property

Pascal

```
property LeftTabScrollButtonRect: TRect;
```

C++

```
__property TRect LeftTabScrollButtonRect;
```

Description

This is LeftTabScrollButtonRect, a member of class TCustomTabData.

76.1.10.5.7 TCustomTabData.NewWidth Property

Pascal

```
property NewWidth [Idx: Integer]: Integer;
```

C++

```
__property int NewWidth[int Idx];
```

Description

This is NewWidth, a member of class TCustomTabData.

76.1.10.5.8 TCustomTabData.Owner Property

Pascal

```
property Owner: TCustomRibbon;
```

C++

```
__property TCustomRibbon Owner;
```

Description

This is Owner, a member of class TCustomTabData.

76.1.10.5.9 TCustomTabData.RightTabScrollButtonRect Property

Pascal

```
property RightTabScrollButtonRect: TRect;
```

C++

```
__property TRect RightTabScrollButtonRect;
```

Description

This is RightTabScrollButtonRect, a member of class TCustomTabData.

76.1.10.5.10 TCustomTabData.ScrollButtons Property

Pascal

```
property ScrollButtons: TRibbonScrollButtons;
```

C++

```
__property TRibbonScrollButtons ScrollButtons;
```

Description

This is ScrollButtons, a member of class TCustomTabData.

76.1.10.5.11 TCustomTabData.SeparatorOpacity Property

Pascal

```
property SeparatorOpacity: Byte;
```

C++

```
__property Byte SeparatorOpacity;
```

Description

This is SeparatorOpacity, a member of class TCustomTabData.

76.1.10.5.12 TCustomTabData.TabSeparatorState Property

Pascal

```
property TabSeparatorState: TTabSeparatorState;
```

C++

```
__property TTabSeparatorState TabSeparatorState;
```

Description

This is TabSeparatorState, a member of class TCustomTabData.

76.1.10.5.13 TCustomTabData.TabUpdateRect Property

Pascal

```
property TabUpdateRect: TRect;
```

C++

```
__property TRect TabUpdateRect;
```

Description

This is TabUpdateRect, a member of class TCustomTabData.

76.1.11 THelpButton Class

Button to activate help.

Class Hierarchy

Pascal

```
THelpButton = class(TOffice2007Button);
```

C++

```
class THelpButton : public TOffice2007Button;
```

File

Ribbon

Description

THelpButton represents the button used for activating help.

See Also

ShowHelpButton (see page 6764)

76.1.11.1 THelpButton Members

The following tables list the members exposed by THelpButton.

Protected Methods

	Name	Description
	CMMouseenter (see page 6829)	This is CMMouseenter, a member of class TOffice2007Button.
	CMMouseleave (see page 6829)	This is CMMouseleave, a member of class TOffice2007Button.
V	DoClick (see page 6829)	This is DoClick, a member of class TOffice2007Button.
V	DrawBackground (see page 6829)	This is DrawBackground, a member of class TOffice2007Button.
V	DrawImage (see page 6826)	This is DrawImage, a member of class THelpButton.
V	MouseDown (see page 6830)	OnMouseDown event dispatcher.
V	MouseMove (see page 6830)	Respond to mouse moving over control area..

	MouseUp (see page 6831)	OnMouseUp event dispatcher.
	Paint (see page 6831)	Render control surface.
	SetParent (see page 6832)	Sets the parent of the control.

Protected Properties

	Name	Description
	MouseInControl (see page 6832)	This is MouseInControl, a member of class TOffice2007Button.
	RibbonStyle (see page 6833)	This is RibbonStyle, a member of class TOffice2007Button.

Public Events

	Name	Description
	OnClick (see page 6833)	Occurs when the user clicks the control.

Public Methods

	Name	Description
	Create (see page 6827)	Creates an instance of TGraphicControl.
	Destroy (see page 6828)	Destroys an instance of TGraphicControl.

76.1.11.2 THelpButton Methods

The methods of the THelpButton class are listed here.

Protected Methods

	Name	Description
	DrawImage (see page 6826)	This is DrawImage, a member of class THelpButton.

76.1.11.2.1 THelpButton.DrawImage Method

Pascal

```
procedure DrawImage; override;
```

C++

```
virtual __fastcall DrawImage();
```

Description

This is DrawImage, a member of class THelpButton.

76.1.12 TOffice2007Button Class

Class Hierarchy

Pascal

```
TOffice2007Button = class(TGraphicControl);
```

C++

```
class TOffice2007Button : public TGraphicControl;
```

File

Ribbon

Description

This is class Ribbon.TOffice2007Button.

76.1.12.1 TOffice2007Button Members

The following tables list the members exposed by TOffice2007Button.

Protected Methods

	Name	Description
	CMMouseenter (see page 6829)	This is CMMouseenter, a member of class TOffice2007Button.
	CMMouseleave (see page 6829)	This is CMMouseleave, a member of class TOffice2007Button.
V	DoClick (see page 6829)	This is DoClick, a member of class TOffice2007Button.
V	DrawBackground (see page 6829)	This is DrawBackground, a member of class TOffice2007Button.
V	DrawImage (see page 6830)	This is DrawImage, a member of class TOffice2007Button.
V	MouseDown (see page 6830)	OnMouseDown event dispatcher.
V	MouseMove (see page 6830)	Respond to mouse moving over control area..
V	MouseUp (see page 6831)	OnMouseUp event dispatcher.
V	Paint (see page 6831)	Render control surface.
V	SetParent (see page 6832)	Sets the parent of the control.

Protected Properties

	Name	Description
	MouseInControl (see page 6832)	This is MouseInControl, a member of class TOffice2007Button.
R	RibbonStyle (see page 6833)	This is RibbonStyle, a member of class TOffice2007Button.

Public Events

	Name	Description
	OnClick (see page 6833)	Occurs when the user clicks the control.

Public Methods

	Name	Description
V	Create (see page 6827)	Creates an instance of TGraphicControl.
V	Destroy (see page 6828)	Destroys an instance of TGraphicControl.

76.1.12.2 TOffice2007Button.Create Constructor

Creates an instance of TGraphicControl.

Pascal

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

C++

```
virtual __fastcall TOffice2007Button(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TGraphicControl. However, you should never attempt to instantiate a TGraphicControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component, typically a form, that is responsible for freeing the graphic control. It becomes the value of the Owner property. If you do not explicitly provide an Owner to the constructor (that is, if AOwner is nil (Delphi) or NULL (C++)), your application is responsible for explicitly freeing the graphic control.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

76.1.12.3 TOffice2007Button.Destroy Destructor

Destroys an instance of TGraphicControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOffice2007Button();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

76.1.12.4 TOffice2007Button Methods

The methods of the TOffice2007Button class are listed here.

Protected Methods

	Name	Description
	CMMouseenter (see page 6829)	This is CMMouseenter, a member of class TOffice2007Button.
	CMMouseleave (see page 6829)	This is CMMouseleave, a member of class TOffice2007Button.
V	DoClick (see page 6829)	This is DoClick, a member of class TOffice2007Button.
V	DrawBackground (see page 6829)	This is DrawBackground, a member of class TOffice2007Button.
V	DrawImage (see page 6830)	This is DrawImage, a member of class TOffice2007Button.
V	MouseDown (see page 6830)	OnMouseDown event dispatcher.
V	MouseMove (see page 6830)	Respond to mouse moving over control area..

	MouseUp (see page 6831)	OnMouseUp event dispatcher.
	Paint (see page 6831)	Render control surface.
	SetParent (see page 6832)	Sets the parent of the control.

76.1.12.4.1 TOffice2007Button.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TOffice2007Button.

76.1.12.4.2 TOffice2007Button.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TOffice2007Button.

76.1.12.4.3 TOffice2007Button.DoClick Method

Pascal

```
procedure DoClick; virtual;
```

C++

```
virtual __fastcall DoClick();
```

Description

This is DoClick, a member of class TOffice2007Button.

76.1.12.4.4 TOffice2007Button.DrawBackground Method

Pascal

```
procedure DrawBackground; virtual;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TOffice2007Button.

76.1.12.4.5 TOffice2007Button.DrawImage Method

Pascal

```
procedure DrawImage; virtual;
```

C++

```
virtual __fastcall DrawImage();
```

Description

This is DrawImage, a member of class TOffice2007Button.

76.1.12.4.6 TOffice2007Button.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

76.1.12.4.7 TOffice2007Button.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler

when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

76.1.12.4.8 TOffice2007Button.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

76.1.12.4.9 TOffice2007Button.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

76.1.12.4.10 TOffice2007Button.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

76.1.12.5 TOffice2007Button Properties

The properties of the TOffice2007Button class are listed here.

Protected Properties

	Name	Description
	MouseInControl (see page 6832)	This is MouseInControl, a member of class TOffice2007Button.
R	RibbonStyle (see page 6833)	This is RibbonStyle, a member of class TOffice2007Button.

76.1.12.5.1 TOffice2007Button.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TOffice2007Button.

76.1.12.5.2 TOffice2007Button.RibbonStyle Property

Pascal

```
property RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__property TRibbonStyleActionBars RibbonStyle;
```

Description

This is RibbonStyle, a member of class TOffice2007Button.

76.1.12.6 TOffice2007Button Events

The events of the TOffice2007Button class are listed here.

Public Events

	Name	Description
	OnClick (see page 6833)	Occurs when the user clicks the control.

76.1.12.6.1 TOffice2007Button.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

76.1.13 TQuickAccessToolbar Class

Quick Access Toolbar appearance.

Class Hierarchy

Pascal

```
TQuickAccessToolbar = class(TPersistent);
```

C++

```
class TQuickAccessToolbar : public TPersistent;
```

File

Ribbon

Description

TQuickAccessToolbar defines the Quick Access Toolbar appearance. This class contains a reference to the Quick Access Toolbar on the form.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

76.1.13.1 TQuickAccessToolbar Members

The following tables list the members exposed by TQuickAccessToolbar.

Enumerations

	Name	Description
	TQATPosition (see page 6836)	This is record Ribbon.TQuickAccessToolbar.TQATPosition.

Protected Methods

	Name	Description
	DoChange (see page 6838)	This is DoChange, a member of class TQuickAccessToolbar.

Public Events

	Name	Description
	OnChange (see page 6840)	This is OnChange, a member of class TQuickAccessToolbar.

Public Methods

	Name	Description
	Assign (see page 6837)	Copies the contents of another, similar object.
	Create (see page 6836)	This is Create, a member of class TQuickAccessToolbar.
	Show (see page 6838)	This is Show, a member of class TQuickAccessToolbar.
	TogglePosition (see page 6838)	This is TogglePosition, a member of class TQuickAccessToolbar.

Public Properties

	Name	Description
	BarWidth (see page 6839)	This is BarWidth, a member of class TQuickAccessToolbar.

Published Properties

	Name	Description
	ActionBar (see page 6839)	This is ActionBar, a member of class TQuickAccessToolbar.
	Position (see page 6839)	This is Position, a member of class TQuickAccessToolbar.
	ShowMinimize (see page 6839)	This is ShowMinimize, a member of class TQuickAccessToolbar.
	ShowMoreCommands (see page 6840)	This is ShowMoreCommands, a member of class TQuickAccessToolbar.
	ShowPosition (see page 6840)	This is ShowPosition, a member of class TQuickAccessToolbar.

76.1.13.2 TQuickAccessToolbar.Create Constructor

Pascal

```
constructor Create(ARibbon: TCustomRibbon); reintroduce;
```

C++

```
__fastcall TQuickAccessToolbar(TCustomRibbon ARibbon);
```

Description

This is Create, a member of class TQuickAccessToolbar.

76.1.13.3 TQuickAccessToolbar Enumerations

The enumerations of the TQuickAccessToolbar class are listed here.

Enumerations

	Name	Description
	TQATPosition (see page 6836)	This is record Ribbon.TQuickAccessToolbar.TQATPosition.

76.1.13.3.1 Ribbon.TQuickAccessToolbar.TQATPosition Enumeration

Pascal

```
TQATPosition = (
  qpTop,
  qpBottom
);
```

C++

```
enum TQATPosition {
  qpTop,
  qpBottom
};
```

File

Ribbon

Description

This is record Ribbon.TQuickAccessToolbar.TQATPosition.

76.1.13.4 TQuickAccessToolbar Methods

The methods of the TQuickAccessToolbar class are listed here.

Protected Methods

	Name	Description
	DoChange (see page 6838)	This is DoChange, a member of class TQuickAccessToolbar.

Public Methods

	Name	Description
	Assign (see page 6837)	Copies the contents of another, similar object.
	Show (see page 6838)	This is Show, a member of class TQuickAccessToolbar.
	TogglePosition (see page 6838)	This is TogglePosition, a member of class TQuickAccessToolbar.

76.1.13.4.1 TQuickAccessToolbar.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

76.1.13.4.2 TQuickAccessToolbar.DoChange Method

Pascal

```
procedure DoChange; virtual;
```

C++

```
virtual __fastcall DoChange();
```

Description

This is DoChange, a member of class TQuickAccessToolbar.

76.1.13.4.3 TQuickAccessToolbar.Show Method

Pascal

```
procedure Show(const Visible: Boolean);
```

C++

```
__fastcall Show(const Boolean Visible);
```

Description

This is Show, a member of class TQuickAccessToolbar.

76.1.13.4.4 TQuickAccessToolbar.TogglePosition Method

Pascal

```
procedure TogglePosition;
```

C++

```
__fastcall TogglePosition();
```

Description

This is TogglePosition, a member of class TQuickAccessToolbar.

76.1.13.5 TQuickAccessToolbar Properties

The properties of the TQuickAccessToolbar class are listed here.

Public Properties

	Name	Description
	BarWidth (see page 6839)	This is BarWidth, a member of class TQuickAccessToolbar.

Published Properties

	Name	Description
	ActionBar (see page 6839)	This is ActionBar, a member of class TQuickAccessToolbar.
	Position (see page 6839)	This is Position, a member of class TQuickAccessToolbar.
	ShowMinimize (see page 6839)	This is ShowMinimize, a member of class TQuickAccessToolbar.
	ShowMoreCommands (see page 6840)	This is ShowMoreCommands, a member of class TQuickAccessToolbar.
	ShowPosition (see page 6840)	This is ShowPosition, a member of class TQuickAccessToolbar.

76.1.13.5.1 TQuickAccessToolbar.ActionBar Property

Pascal

```
property ActionBar: TCustomRibbonQuickAccessToolbar;
```

C++

```
__property TCustomRibbonQuickAccessToolbar ActionBar;
```

Description

This is ActionBar, a member of class TQuickAccessToolbar.

76.1.13.5.2 TQuickAccessToolbar.BarWidth Property

Pascal

```
property BarWidth: Integer;
```

C++

```
__property int BarWidth;
```

Description

This is BarWidth, a member of class TQuickAccessToolbar.

76.1.13.5.3 TQuickAccessToolbar.Position Property

Pascal

```
property Position: TQATPosition;
```

C++

```
__property TQATPosition Position;
```

Description

This is Position, a member of class TQuickAccessToolbar.

76.1.13.5.4 TQuickAccessToolbar.ShowMinimize Property

Pascal

```
property ShowMinimize: Boolean;
```

C++

```
__property Boolean ShowMinimize;
```

Description

This is ShowMinimize, a member of class TQuickAccessToolbar.

76.1.13.5.5 TQuickAccessToolbar.ShowMoreCommands Property

Pascal

```
property ShowMoreCommands: Boolean;
```

C++

```
__property Boolean ShowMoreCommands;
```

Description

This is ShowMoreCommands, a member of class TQuickAccessToolbar.

76.1.13.5.6 TQuickAccessToolbar.ShowPosition Property

Pascal

```
property ShowPosition: Boolean;
```

C++

```
__property Boolean ShowPosition;
```

Description

This is ShowPosition, a member of class TQuickAccessToolbar.

76.1.13.6 TQuickAccessToolbar Events

The events of the TQuickAccessToolbar class are listed here.

Public Events

	Name	Description
	OnChange (see page 6840)	This is OnChange, a member of class TQuickAccessToolbar.

76.1.13.6.1 TQuickAccessToolbar.OnChange Event

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

This is OnChange, a member of class TQuickAccessToolbar.

76.1.14 TQuickAccessToolbarGroupItem Class

Display Quick Access Toolbar commands to display entire Ribbon (see page 6713) group.

Class Hierarchy

Pascal

```
TQuickAccessToolbarGroupItem = class;
```

C++

```
class TQuickAccessToolbarGroupItem;
```

File

Ribbon

Description

TQuickAccessToolbarGroupItem is used to display Quick Access Toolbar commands that display an entire Ribbon (see page 6713) group when clicked.

76.1.14.1 TQuickAccessToolbarGroupItem Members

The following tables list the members exposed by TQuickAccessToolbarGroupItem.

Public Methods

	Name	Description
	Create (see page 6841)	This is Create, a member of class TQuickAccessToolbarGroupItem.
V	Destroy (see page 6842)	This is Destroy, a member of class TQuickAccessToolbarGroupItem.

Public Properties

	Name	Description
R	Action (see page 6842)	This is Action, a member of class TQuickAccessToolbarGroupItem.
R	Caption (see page 6842)	This is Caption, a member of class TQuickAccessToolbarGroupItem.
R	Group (see page 6842)	This is Group, a member of class TQuickAccessToolbarGroupItem.

76.1.14.2 TQuickAccessToolbarGroupItem.Create Constructor

Pascal

```
constructor Create(ActionList: TActionList; AGroup: TCustomRibbonGroup); reintroduce;
```

C++

```
__fastcall TQuickAccessToolbarGroupItem(TActionList ActionList, TCustomRibbonGroup AGroup);
```

Description

This is Create, a member of class TQuickAccessToolbarGroupItem.

76.1.14.3 TQuickAccessToolbarGroupItem.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TQuickAccessToolbarGroupItem();
```

Description

This is Destroy, a member of class TQuickAccessToolbarGroupItem.

76.1.14.4 TQuickAccessToolbarGroupItem Properties

The properties of the TQuickAccessToolbarGroupItem class are listed here.

Public Properties

	Name	Description
	Action (see page 6842)	This is Action, a member of class TQuickAccessToolbarGroupItem.
	Caption (see page 6842)	This is Caption, a member of class TQuickAccessToolbarGroupItem.
	Group (see page 6842)	This is Group, a member of class TQuickAccessToolbarGroupItem.

76.1.14.4.1 TQuickAccessToolbarGroupItem.Action Property

Pascal

```
property Action: TAction;
```

C++

```
__property TAction Action;
```

Description

This is Action, a member of class TQuickAccessToolbarGroupItem.

76.1.14.4.2 TQuickAccessToolbarGroupItem.Caption Property

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

This is Caption, a member of class TQuickAccessToolbarGroupItem.

76.1.14.4.3 TQuickAccessToolbarGroupItem.Group Property

Pascal

```
property Group: TCustomRibbonGroup;
```

C++

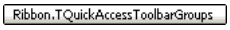
```
__property TCustomRibbonGroup Group;
```

Description

This is Group, a member of class TQuickAccessToolbarGroupItem.

76.1.15 TQuickAccessToolbarGroups Class

Class Hierarchy



Pascal

```
TQuickAccessToolbarGroups = class;
```

C++

```
class TQuickAccessToolbarGroups;
```

File

Ribbon

Description

This is class Ribbon.TQuickAccessToolbarGroups.

76.1.15.1 TQuickAccessToolbarGroups Members

The following tables list the members exposed by TQuickAccessToolbarGroups.

Public Methods

	Name	Description
	Add (see page 6844)	This is Add, a member of class TQuickAccessToolbarGroups.
	Clean (see page 6844)	This is Clean, a member of class TQuickAccessToolbarGroups.
V	Create (see page 6843)	This is Create, a member of class TQuickAccessToolbarGroups.
V	Destroy (see page 6844)	This is Destroy, a member of class TQuickAccessToolbarGroups.
	Exists (see page 6844)	This is Exists, a member of class TQuickAccessToolbarGroups.
	Group (see page 6845)	This is Group, a member of class TQuickAccessToolbarGroups.
	Remove (see page 6845)	This is Remove, a member of class TQuickAccessToolbarGroups.

76.1.15.2 TQuickAccessToolbarGroups.Create Constructor

Pascal

```
constructor Create(Images: TCustomImageList); virtual;
```

C++

```
virtual __fastcall TQuickAccessToolbarGroups(TCustomImageList Images);
```

Description

This is Create, a member of class TQuickAccessToolbarGroups.

76.1.15.3 TQuickAccessToolbarGroups.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TQuickAccessToolbarGroups();
```

Description

This is Destroy, a member of class TQuickAccessToolbarGroups.

76.1.15.4 TQuickAccessToolbarGroups Methods

The methods of the TQuickAccessToolbarGroups class are listed here.

Public Methods

	Name	Description
	Add (see page 6844)	This is Add, a member of class TQuickAccessToolbarGroups.
	Clean (see page 6844)	This is Clean, a member of class TQuickAccessToolbarGroups.
	Exists (see page 6844)	This is Exists, a member of class TQuickAccessToolbarGroups.
	Group (see page 6845)	This is Group, a member of class TQuickAccessToolbarGroups.
	Remove (see page 6845)	This is Remove, a member of class TQuickAccessToolbarGroups.

76.1.15.4.1 TQuickAccessToolbarGroups.Add Method

Pascal

```
function Add(AGroup: TCustomRibbonGroup): TQuickAccessToolbarGroupItem;
```

C++

```
__fastcall TQuickAccessToolbarGroupItem Add(TCustomRibbonGroup AGroup);
```

Description

This is Add, a member of class TQuickAccessToolbarGroups.

76.1.15.4.2 TQuickAccessToolbarGroups.Clean Method

Pascal

```
procedure Clean;
```

C++

```
__fastcall Clean();
```

Description

This is Clean, a member of class TQuickAccessToolbarGroups.

76.1.15.4.3 TQuickAccessToolbarGroups.Exists Method

Pascal

```
function Exists(const ACaption: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const UnicodeString ACaption);
```

Description

This is Exists, a member of class TQuickAccessToolbarGroups.

76.1.15.4.4 TQuickAccessToolbarGroups.Group Method

Pascal

```
function Group(const ACaption: string): TCustomRibbonGroup;
```

C++

```
__fastcall TCustomRibbonGroup Group(const UnicodeString ACaption);
```

Description

This is Group, a member of class TQuickAccessToolbarGroups.

76.1.15.4.5 TQuickAccessToolbarGroups.Remove Method

Pascal

```
procedure Remove(const ACaption: string);
```

C++

```
__fastcall Remove(const UnicodeString ACaption);
```

Description

This is Remove, a member of class TQuickAccessToolbarGroups.

76.1.16 TRibbon Class

Implement ribbon control.

Class Hierarchy

Pascal

```
TRibbon = class(TCustomRibbon);
```

C++

```
class TRibbon : public TCustomRibbon;
```

File

Ribbon

Description

TRibbon is a TCustomRibbon (see page 6724) that implements the ribbon control. Use TRibbon to create a user interface that looks and performs like the Microsoft Fluent User Interface.

Instead of the tradional user interface where the application has a top level menu, there is no top level menu when using TRibbon. Instead there is an Application (see page 2612) Menu (see page 6762), represented by a large round button in the

top left corner. Commands are displayed in groups in tabs. These groups are more like toolbars and can display a variety of user interface elements.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TCustomRibbon (see page 6724)

76.1.16.1 TRibbon Members

The following tables list the members exposed by TRibbon.

Protected Events

	Name	Description
	OnLeftTabScrollButtonClick (see page 6766)	This is OnLeftTabScrollButtonClick, a member of class TCustomRibbon.
	OnRightTabScrollButtonClick (see page 6767)	This is OnRightTabScrollButtonClick, a member of class TCustomRibbon.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 6736)	This is CMBidimodechanged, a member of class TCustomRibbon.
	CMDesignHitTest (see page 6737)	This is CMDesignHitTest, a member of class TCustomRibbon.
	CMFontchanged (see page 6737)	This is CMFontchanged, a member of class TCustomRibbon.
	CMMouseenter (see page 6737)	This is CMMouseenter, a member of class TCustomRibbon.
	CMMouseleave (see page 6737)	This is CMMouseleave, a member of class TCustomRibbon.
	CMTextchanged (see page 6738)	This is CMTextchanged, a member of class TCustomRibbon.
	CMThemeChanged (see page 6738)	This is CMThemeChanged, a member of class TCustomRibbon.
	ContextMenuPopupHandler (see page 6738)	This is ContextMenuPopupHandler, a member of class TCustomRibbon.
	CreateContextMenu (see page 6738)	This is CreateContextMenu, a member of class TCustomRibbon.
	CustomFrameWndProc (see page 6739)	This is CustomFrameWndProc, a member of class TCustomRibbon.
	DblClick (see page 6739)	OnDblClick event dispatcher.
	DefineProperties (see page 6739)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DisplayKeyTips (see page 6740)	This is DisplayKeyTips, a member of class TCustomRibbon.
	DoAddToQuickAccessToolbar (see page 6740)	This is DoAddToQuickAccessToolbar, a member of class TCustomRibbon.
	DoChangeQuickAccessToolbarPosition (see page 6740)	This is DoChangeQuickAccessToolbarPosition, a member of class TCustomRibbon.
	DoContextPopup (see page 6741)	OnContextPopup event dispatcher.
	DoCustomizeQuickAccessToolbar (see page 6741)	This is DoCustomizeQuickAccessToolbar, a member of class TCustomRibbon.
	DoGroupStyleChanged (see page 6741)	This is DoGroupStyleChanged, a member of class TCustomRibbon.
	DoHelpButtonClick (see page 6741)	This is DoHelpButtonClick, a member of class TCustomRibbon.
	DoMinimizeRibbon (see page 6742)	This is DoMinimizeRibbon, a member of class TCustomRibbon.
	DoMouseWheelDown (see page 6742)	OnMouseWheelDown event dispatcher.
	DoMouseWheelUp (see page 6743)	OnMouseWheelUp event dispatcher.
	DoRecentItemClick (see page 6743)	This is DoRecentItemClick, a member of class TCustomRibbon.
	DoScrollTabsLeft (see page 6743)	This is DoScrollTabsLeft, a member of class TCustomRibbon.
	DoScrollTabsRight (see page 6743)	This is DoScrollTabsRight, a member of class TCustomRibbon.
	DoTabChange (see page 6744)	This is DoTabChange, a member of class TCustomRibbon.
	DrawCaption (see page 6744)	This is DrawCaption, a member of class TCustomRibbon.
	DrawQuickAccessToolbar (see page 6744)	This is DrawQuickAccessToolbar, a member of class TCustomRibbon.
	DrawSystemButtons (see page 6744)	This is DrawSystemButtons, a member of class TCustomRibbon.
	DrawTab (see page 6745)	This is DrawTab, a member of class TCustomRibbon.
	GetContextMenuOptions (see page 6745)	This is GetContextMenuOptions, a member of class TCustomRibbon.
	GetHelpButtonRect (see page 6745)	This is GetHelpButtonRect, a member of class TCustomRibbon.

	GetRibbonHeight (see page 6746)	This is GetRibbonHeight, a member of class TCustomRibbon.
V	GetTabAt (see page 6746)	This is GetTabAt, a member of class TCustomRibbon.
V	GetTabRect (see page 6746)	This is GetTabRect, a member of class TCustomRibbon.
V	GetTabWidth (see page 6747)	This is GetTabWidth, a member of class TCustomRibbon.
	HasApplicationMenu (see page 6747)	This is HasApplicationMenu, a member of class TCustomRibbon.
	HasQuickAccessToolbar (see page 6747)	This is HasQuickAccessToolbar, a member of class TCustomRibbon.
	Idle (see page 6747)	This is Idle, a member of class TCustomRibbon.
V	KeyDown (see page 6749)	Respond to key press events.
V	Loaded (see page 6749)	Initializes the control after it is loaded from a stream.
V	MouseDown (see page 6750)	OnMouseDown event dispatcher.
	MouseInSystemMenu (see page 6750)	This is MouseInSystemMenu, a member of class TCustomRibbon.
V	MouseMove (see page 6750)	Respond to mouse moving over control area..
V	MouseUp (see page 6751)	OnMouseUp event dispatcher.
V	Notification (see page 6752)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 6752)	Renders the image of a custom control.
	RebuildRibbonMetrics (see page 6753)	This is RebuildRibbonMetrics, a member of class TCustomRibbon.
V	Resize (see page 6754)	Respond to control resize.
V	SetParent (see page 6754)	Sets the parent of the control.
	UMCheckResize (see page 6756)	This is UMCheckResize, a member of class TCustomRibbon.
	UMDisplayKeyTips (see page 6756)	This is UMDisplayKeyTips, a member of class TCustomRibbon.
	UpdateQATPosition (see page 6757)	This is UpdateQATPosition, a member of class TCustomRibbon.
	UpdateSystemMenuInfo (see page 6757)	This is UpdateSystemMenuInfo, a member of class TCustomRibbon.
	ValidTab (see page 6757)	This is ValidTab, a member of class TCustomRibbon.
	WMEraseBkgnd (see page 6757)	This is WMEraseBkgnd, a member of class TCustomRibbon.
	WMGetDlgCode (see page 6758)	This is WMGetDlgCode, a member of class TCustomRibbon.
	WMKillFocus (see page 6758)	This is WMKillFocus, a member of class TCustomRibbon.
	WMSettingChange (see page 6758)	This is WMSettingChange, a member of class TCustomRibbon.
	WMWindowPosChanged (see page 6758)	This is WMWindowPosChanged, a member of class TCustomRibbon.

Protected Properties

	Name	Description
	AllowKeyboardNavigation (see page 6760)	This is AllowKeyboardNavigation, a member of class TCustomRibbon.
	HideRibbon (see page 6761)	This is HideRibbon, a member of class TCustomRibbon.
	HotTabIndex (see page 6762)	This is HotTabIndex, a member of class TCustomRibbon.
	KeyTipsVisible (see page 6762)	This is KeyTipsVisible, a member of class TCustomRibbon.
R	Menu (see page 6762)	This is Menu, a member of class TCustomRibbon.
R	ScaleFactor (see page 6763)	This is ScaleFactor, a member of class TCustomRibbon.

Public Constants

	Name	Description
	cRibbonFirstTabOffSet (see page 6730)	This is cRibbonFirstTabOffSet, a member of class TCustomRibbon.
	cRibbonGroupCaptionHeight (see page 6730)	This is cRibbonGroupCaptionHeight, a member of class TCustomRibbon.
	cRibbonGroupHeight (see page 6730)	This is cRibbonGroupHeight, a member of class TCustomRibbon.
	cRibbonHeight (see page 6730)	This is cRibbonHeight, a member of class TCustomRibbon.
	cRibbonHideHeight (see page 6730)	This is cRibbonHideHeight, a member of class TCustomRibbon.
	cRibbonHideWidth (see page 6731)	This is cRibbonHideWidth, a member of class TCustomRibbon.
	cRibbonMinimizedHeight (see page 6731)	This is cRibbonMinimizedHeight, a member of class TCustomRibbon.
	cRibbonMinimumCaptionWidth (see page 6731)	This is cRibbonMinimumCaptionWidth, a member of class TCustomRibbon.
	cRibbonPageHeight (see page 6731)	This is cRibbonPageHeight, a member of class TCustomRibbon.
	cRibbonQATHeight (see page 6732)	This is cRibbonQATHeight, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarLeft (see page 6732)	This is cRibbonQuickAccessToolbarLeft, a member of class TCustomRibbon.
	cRibbonQuickAccessToolbarTop (see page 6732)	This is cRibbonQuickAccessToolbarTop, a member of class TCustomRibbon.
	cRibbonTabHeight (see page 6732)	This is cRibbonTabHeight, a member of class TCustomRibbon.
	cRibbonTabScrollButtonWidth (see page 6732)	This is cRibbonTabScrollButtonWidth, a member of class TCustomRibbon.
	cRibbonTabSpacing (see page 6733)	This is cRibbonTabSpacing, a member of class TCustomRibbon.

	cRibbonUnthemedCaptionHeight (see page 6733)	This is cRibbonUnthemedCaptionHeight, a member of class TCustomRibbon.
	UM_CHECKSIZE (see page 6733)	This is UM_CHECKSIZE, a member of class TCustomRibbon.
	UM_DISPLAYKEYTIPS (see page 6733)	This is UM_DISPLAYKEYTIPS, a member of class TCustomRibbon.

Public Methods

	Name	Description
	AddRecentItem (see page 6735)	Add new entry into Recent Items list.
	AddTab (see page 6736)	This is AddTab, a member of class TCustomRibbon.
	BeginUpdate (see page 6736)	This is BeginUpdate, a member of class TCustomRibbon.
	CheckRibbonSize (see page 6736)	This is CheckRibbonSize, a member of class TCustomRibbon.
	ClearRecentItems (see page 6736)	This is ClearRecentItems, a member of class TCustomRibbon.
V	Create (see page 6728)	Creates an instance of TCustomControl.
	CustomFrameSupported (see page 6738)	This is CustomFrameSupported, a member of class TCustomRibbon.
	DeleteTab (see page 6740)	Delete tab at specified Index.
V	Destroy (see page 6728)	Destroys an instance of TCustomControl.
	DoMouseWheelEx (see page 6742)	This is DoMouseWheelEx, a member of class TCustomRibbon.
	EndUpdate (see page 6745)	This is EndUpdate, a member of class TCustomRibbon.
V	GetIcon (see page 6745)	Return Icon used by Application (see page 2612) Menu (see page 6762) Button.
	GetRibbonMetric (see page 6746)	This is GetRibbonMetric, a member of class TCustomRibbon.
	InvalidateHeaderArea (see page 6747)	Invalidate header section of Ribbon (see page 6713).
	InvalidateScrollButtons (see page 6748)	This is InvalidateScrollButtons, a member of class TCustomRibbon.
	InvalidateSystemMenuArea (see page 6748)	This is InvalidateSystemMenuArea, a member of class TCustomRibbon.
	InvalidateTabArea (see page 6748)	Invalidate area that displays tabs.
	MoveGroup (see page 6751)	Move group to new page.
	RemoveAllTabs (see page 6753)	Remove all Tabs (see page 6764) from Ribbon (see page 6713).
	RemoveRecentItem (see page 6753)	This is RemoveRecentItem, a member of class TCustomRibbon.
	RemoveTab (see page 6753)	Remove tab with specified caption.
V	SetBounds (see page 6754)	Sets the windowed control's boundary properties all at once.
	TabHandledKeyTips (see page 6755)	Validate **KeyTip** input when **KeyTip**s State is **ktsTabs**.
	TabHideKeyTips (see page 6755)	Hide K**eyTip**s.
	TabShowKeyTips (see page 6756)	Display **KeyTip**s for **ktsTabs** State.

Public Nested Types

Name	Description
TTabChangeEvent (see page 6765)	This is nested type Ribbon.TCustomRibbon.TTabChangeEvent.

Public Properties

	Name	Description
	Active (see page 6759)	Indicates if form that ribbon is placed on is active form.
	ActivePage (see page 6760)	Page currently displayed by ribbon control.
	ColorMap (see page 6761)	This is ColorMap, a member of class TCustomRibbon.
	Minimized (see page 6763)	Show Ribbon (see page 6713) in minimized state where tabs can be selected but are not visible.

Published Events

	Name	Description
	OnHelpButtonClick (see page 6863)	This is OnHelpButtonClick, a member of class TRibbon.
	OnRecentItemClick (see page 6863)	Recent item clicked.
	OnTabChange (see page 6864)	This is OnTabChange, a member of class TRibbon.

Published Properties

	Name	Description
	ActionManager (see page 6849)	This is ActionManager, a member of class TRibbon.
	Align (see page 6850)	Determines how the control aligns within its container (parent control).

	Anchors (see page 6853)	Specifies how the control is anchored to its parent.
	ApplicationMenu (see page 6854)	Change appearance of ribbon's associated TRibbonApplicationMenuBar control.
	BiDiMode (see page 6854)	Specifies the bi-directional mode for the control.
	Caption (see page 6854)	Specifies a text string that identifies the control to the user.
	DocumentName (see page 6857)	Text in Ribbon caption.
	Enabled (see page 6857)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 6858)	Controls the attributes of text written on or in the control.
	Height (see page 6859)	Specifies the vertical size of the control in pixels.
	HideTabs (see page 6860)	Shows or hides Tab portion of Ribbon.
	ParentBiDiMode (see page 6860)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 6861)	Determines where a control looks for its font information.
	QuickAccessToolbar (see page 6861)	Properties associated with TQuickAccessToolbar.
	ScreenTips (see page 6861)	This is ScreenTips, a member of class TRibbon.
	ShowHelpButton (see page 6862)	Determines whether ribbon displays Help Button.
	Style (see page 6862)	This is Style, a member of class TRibbon.
	TabIndex (see page 6862)	Change the active page of ribbon.
	Tabs (see page 6862)	Collection of TRibbonTabs visible on ribbon.
	UseCustomFrame (see page 6863)	Indicates whether to use a custom frame for ribbon controls.

76.1.16.2 TRibbon Properties

The properties of the TRibbon class are listed here.

Published Properties

	Name	Description
	ActionManager (see page 6849)	This is ActionManager, a member of class TRibbon.
	Align (see page 6850)	Determines how the control aligns within its container (parent control).
	Anchors (see page 6853)	Specifies how the control is anchored to its parent.
	ApplicationMenu (see page 6854)	Change appearance of ribbon's associated TRibbonApplicationMenuBar control.
	BiDiMode (see page 6854)	Specifies the bi-directional mode for the control.
	Caption (see page 6854)	Specifies a text string that identifies the control to the user.
	DocumentName (see page 6857)	Text in Ribbon caption.
	Enabled (see page 6857)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 6858)	Controls the attributes of text written on or in the control.
	Height (see page 6859)	Specifies the vertical size of the control in pixels.
	HideTabs (see page 6860)	Shows or hides Tab portion of Ribbon.
	ParentBiDiMode (see page 6860)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 6861)	Determines where a control looks for its font information.
	QuickAccessToolbar (see page 6861)	Properties associated with TQuickAccessToolbar.
	ScreenTips (see page 6861)	This is ScreenTips, a member of class TRibbon.
	ShowHelpButton (see page 6862)	Determines whether ribbon displays Help Button.
	Style (see page 6862)	This is Style, a member of class TRibbon.
	TabIndex (see page 6862)	Change the active page of ribbon.
	Tabs (see page 6862)	Collection of TRibbonTabs visible on ribbon.
	UseCustomFrame (see page 6863)	Indicates whether to use a custom frame for ribbon controls.

76.1.16.2.1 TRibbon.ActionManager Property

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

This is ActionManager, a member of class TRibbon.

76.1.16.2.2 TRibbon.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
```

```
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
```

```
  clause of the form's implementation section.
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
```

```
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

76.1.16.2.3 TRibbon.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

76.1.16.2.4 TRibbon.ApplicationMenu Property

Change appearance of ribbon's associated TRibbonApplicationMenuBar control.

Pascal

```
property ApplicationMenu: TApplicationMenu;
```

C++

```
__property TApplicationMenu ApplicationMenu;
```

Description

ApplicationMenu changes the appearance for the ribbon's associated TRibbonApplicationMenuBar control.

Changing the values of the fields in ApplicationMenu changes the characteristics of the Application Menu for the ribbon. The Application Menu on the ribbon is represented by the round button in the top left corner.

See Also

TApplicationMenu (see page 6716)

TRibbonApplicationMenuBar (see page 7302)

76.1.16.2.5 TRibbon.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

76.1.16.2.6 TRibbon.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
```

```
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
```

```
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

76.1.16.2.7 TRibbon.DocumentName Property

Text in Ribbon caption.

Pascal

```
property DocumentName: string;
```

C++

```
__property UnicodeString DocumentName;
```

Description

DocumentName appears in the Ribbon caption.

See Also

CaptionFont

76.1.16.2.8 TRibbon.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

76.1.16.2.9 TRibbon.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
```

```
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

76.1.16.2.10 TRibbon.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

76.1.16.2.11 TRibbon.HideTabs Property

Shows or hides Tab portion of Ribbon.

Pascal

```
property HideTabs: Boolean;
```

C++

```
__property Boolean HideTabs;
```

Description

HideTabs shows or hides the Tab portion of the Ribbon. When HideTabs is True, the user cannot access any of the Tabs on the Ribbon.

76.1.16.2.12 TRibbon.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

76.1.16.2.13 TRibbon.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

76.1.16.2.14 TRibbon.QuickAccessToolbar Property

Properties associated with TQuickAccessToolbar.

Pascal

```
property QuickAccessToolbar: TQuickAccessToolbar;
```

C++

```
__property TQuickAccessToolbar QuickAccessToolbar;
```

Description

QuickAccessToolbar has the properties associated with the TQuickAccessToolbar on the ribbon. Use these properties to change the appearance of the Quick Access Toolbar as displayed on the ribbon.

See Also

TQuickAccessToolbar (see page 6835)

76.1.16.2.15 TRibbon.ScreenTips Property

Pascal

```
property ScreenTips: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTips;
```

Description

This is ScreenTips, a member of class TRibbon.

76.1.16.2.16 TRibbon.ShowHelpButton Property

Determines whether ribbon displays Help Button.

Pascal

```
property ShowHelpButton: Boolean;
```

C++

```
__property Boolean ShowHelpButton;
```

Description

ShowHelpButton indicates whether or not the ribbon displays a Help Button in the top right hand corner.

See Also

THelpButton (see page 6825)

76.1.16.2.17 TRibbon.Style Property

Pascal

```
property Style: TRibbonStyleActionBars;
```

C++

```
__property TRibbonStyleActionBars Style;
```

Description

This is Style, a member of class TRibbon.

76.1.16.2.18 TRibbon.TabIndex Property

Change the active page of ribbon.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

TabIndex changes the active page of the ribbon. Set the TabIndex property to the desired page index.

See Also

Tabs (see page 6764)

76.1.16.2.19 TRibbon.Tabs Property

Collection of TRibbonTabs visible on ribbon.

Pascal

```
property Tabs: TRibbonTabs;
```

C++

```
__property TRibbonTabs Tabs;
```

Description

Tabs contains a collection of TRibbonTabs visible on the ribbon.

Tabs is a linked list of TRibbonTabs objects. TRibbonTabs is a simple data structure that contains the caption for the tab, the keytip for the tab as well as a reference to the Tab object.

See Also

TRibbonTabs (see page 6925)

TabIndex (see page 6764)

76.1.16.2.20 TRibbon.UseCustomFrame Property

Indicates whether to use a custom frame for ribbon controls.

Pascal

```
property UseCustomFrame: Boolean;
```

C++

```
__property Boolean UseCustomFrame;
```

Description

UseCustomFrame indicates whether or not to use a custom frame, such as TRibbonForm, for ribbon controls.

See Also

TRibbonForm

76.1.16.3 TRibbon Events

The events of the TRibbon class are listed here.

Published Events

	Name	Description
	OnHelpButtonClick (see page 6863)	This is OnHelpButtonClick, a member of class TRibbon.
	OnRecentItemClick (see page 6863)	Recent item clicked.
	OnTabChange (see page 6864)	This is OnTabChange, a member of class TRibbon.

76.1.16.3.1 TRibbon.OnHelpButtonClick Event

Pascal

```
property OnHelpButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHelpButtonClick;
```

Description

This is OnHelpButtonClick, a member of class TRibbon.

76.1.16.3.2 TRibbon.OnRecentItemClick Event

Recent item clicked.

Pascal

```
property OnRecentItemClick: TRecentItemClickEvent;
```

C++

```
__property TRecentItemClickEvent OnRecentItemClick;
```

Description

OnRecentItemClick occurs when a recent item is clicked.

See Also

TRibbonApplicationMenuBar (see page 7302)

RecentItems (see page 7316)

76.1.16.3.3 TRibbon.OnTabChange Event

Pascal

```
property OnTabChange: TTabChangeEvent;
```

C++

```
__property TTabChangeEvent OnTabChange;
```

Description

This is OnTabChange, a member of class TRibbon.

76.1.17 TRibbonCustomizeDlg Class

Class Hierarchy

Pascal

```
TRibbonCustomizeDlg = class(TCustomizeDlg);
```

C++

```
class TRibbonCustomizeDlg : public TCustomizeDlg;
```

File

Ribbon

Description

This is class Ribbon.TRibbonCustomizeDlg.

76.1.17.1 TRibbonCustomizeDlg Members

The following tables list the members exposed by TRibbonCustomizeDlg.

Protected Methods

	Name	Description
	SetupDlg (see page 6865)	This is SetupDlg, a member of class TRibbonCustomizeDlg.

76.1.17.2 TRibbonCustomizeDlg Methods

The methods of the TRibbonCustomizeDlg class are listed here.

Protected Methods

	Name	Description
	SetupDlg (see page 6865)	This is SetupDlg, a member of class TRibbonCustomizeDlg.

76.1.17.2.1 TRibbonCustomizeDlg.SetupDlg Method

Pascal

```
procedure SetupDlg; override;
```

C++

```
virtual __fastcall SetupDlg();
```

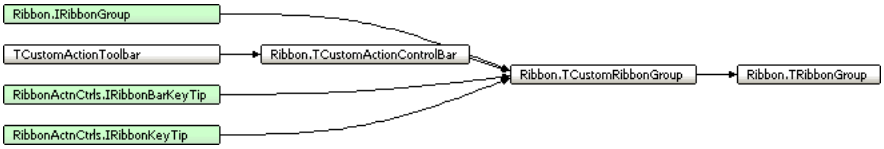
Description

This is SetupDlg, a member of class TRibbonCustomizeDlg.

76.1.18 TRibbonGroup Class

Control used to display the user selectable commands for the ribbon.

Class Hierarchy



Pascal

```
TRibbonGroup = class(TCustomRibbonGroup);
```

C++

```
class TRibbonGroup : public TCustomRibbonGroup;
```

File

Ribbon

Description

TRibbonGroup is a TCustomRibbonGroup (see page 6771) that is used to display the user selectable commands for the ribbon.

When creating a user interface using TRibbon (see page 6845), you add commands to ribbon groups. Ribbon (see page 6713) groups are placed on a ribbon page. You can add new groups to a ribbon by right clicking on the ribbon page and selecting the Add Group component editor item.

The position of the ribbon group is determined by the ribbon page that it was placed on. You cannot freely move the ribbon group around the ribbon page. To change the position of the group within the same page, change the GroupIndex (see page 6869) property.

To move the ribbon group to another ribbon page, use the Move Group command available when right clicking on the ribbon page at design time.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TRibbon (see page 6845)

TCustomRibbonGroup (see page 6771)

GroupIndex (see page 6787)

76.1.18.1 TRibbonGroup Members

The following tables list the members exposed by TRibbonGroup.

Enumerations

	Name	Description
	TGroupAlign (see page 6774)	This is record Ribbon.TCustomRibbonGroup.TGroupAlign.

Protected Methods

	Name	Description
V	AlignControls (see page 6775)	This is AlignControls, a member of class TCustomRibbonGroup.
V	AutoSizingChanged (see page 6776)	This is AutoSizingChanged, a member of class TCustomRibbonGroup.
	CMControlChange (see page 6722)	This is CMControlChange, a member of class TCustomActionControlBar.
	CMFontchanged (see page 6777)	This is CMFontchanged, a member of class TCustomRibbonGroup.
	CMItemDropPoint (see page 6777)	This is CMItemDropPoint, a member of class TCustomRibbonGroup.
	CMMouseenter (see page 6778)	This is CMMouseenter, a member of class TCustomRibbonGroup.
	CMMouseleave (see page 6778)	This is CMMouseleave, a member of class TCustomRibbonGroup.
	CMTextchanged (see page 6778)	This is CMTextchanged, a member of class TCustomRibbonGroup.
V	CreateControls (see page 6723)	This is CreateControls, a member of class TCustomActionControlBar.
V	CreateControls (see page 6779)	This is CreateControls, a member of class TCustomRibbonGroup.
V	DoDropItem (see page 6779)	This is DoDropItem, a member of class TCustomRibbonGroup.
V	DoEndDrag (see page 6779)	This is DoEndDrag, a member of class TCustomRibbonGroup.
V	DrawShowDialogButton (see page 6780)	This is DrawShowDialogButton, a member of class TCustomRibbonGroup.
	GetCaptionHeight (see page 6780)	This is GetCaptionHeight, a member of class TCustomRibbonGroup.
V	GetControlClass (see page 6723)	This is GetControlClass, a member of class TCustomActionControlBar.
	GetGroupHeight (see page 6780)	This is GetGroupHeight, a member of class TCustomRibbonGroup.
V	HideControls (see page 6781)	This is HideControls, a member of class TCustomRibbonGroup.
V	MouseDown (see page 6782)	This is MouseDown, a member of class TCustomRibbonGroup.
V	MouseMove (see page 6782)	This is MouseMove, a member of class TCustomRibbonGroup.
V	MouseUp (see page 6783)	This is MouseUp, a member of class TCustomRibbonGroup.
V	Notification (see page 6783)	This is Notification, a member of class TCustomRibbonGroup.
V	Paint (see page 6783)	This is Paint, a member of class TCustomRibbonGroup.
V	Resize (see page 6783)	This is Resize, a member of class TCustomRibbonGroup.
V	SetAutoSizing (see page 6783)	This is SetAutoSizing, a member of class TCustomRibbonGroup.
V	SetParent (see page 6784)	This is SetParent, a member of class TCustomRibbonGroup.
V	SetupDropDownBtn (see page 6785)	This is SetupDropDownBtn, a member of class TCustomRibbonGroup.
V	ShowControls (see page 6785)	This is ShowControls, a member of class TCustomRibbonGroup.

Protected Properties

	Name	Description
R	CreatingControls (see page 6723)	This is CreatingControls, a member of class TCustomActionControlBar.

Public Methods

	Name	Description
	AlignGroupControls (see page 6934)	Realign controls on group.
V	AlignGroupControls (see page 6775)	Align controls in Ribbon (see page 6713) Group
V	Assign (see page 6775)	This is Assign, a member of class TCustomRibbonGroup.
	BarHandleKeyTips (see page 6776)	Iterate over any visible **KeyTip (see page 6788)**s and perform validation to determine if one of **KeyTip (see page 6788)**s is partial or full match.
	BarHandleKeyTips (see page 7273)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 6776)	Hide any visible KeyTips that commands are displaying.
	BarHideKeyTips (see page 7273)	Hide any visible **KeyTip** windows.
	BarShowKeyTips (see page 6777)	Show K**eyTip**s for commands.
	BarShowKeyTips (see page 7274)	Show **KeyTip**s for any commands that support them.
	Clone (see page 6777)	This is Clone, a member of class TCustomRibbonGroup.
V	Create (see page 6722)	This is Create, a member of class TCustomActionControlBar.
V	Create (see page 6773)	This is Create, a member of class TCustomRibbonGroup.
	CreateControl (see page 6934)	Recreate specific control.
V	CreateControl (see page 6722)	This is CreateControl, a member of class TCustomActionControlBar.
V	CreateControl (see page 6778)	Recreate specific control.
V	Destroy (see page 6773)	This is Destroy, a member of class TCustomRibbonGroup.
V	DockDrop (see page 6779)	This is DockDrop, a member of class TCustomRibbonGroup.
V	DragDrop (see page 6779)	This is DragDrop, a member of class TCustomRibbonGroup.
	GetKeyTip (see page 6780)	Returns current **KeyTip** for command.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	HandleKeyTip (see page 6781)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HideKeyTip (see page 6781)	Hide **KeyTip** for command.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	KeyTipClick (see page 6782)	Handle clicking a **KeyTip**.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
V	SetBounds (see page 6784)	This is SetBounds, a member of class TCustomRibbonGroup.
	SetKeyTip (see page 6784)	Sets **KeyTip** for command.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	ShowKeyTip (see page 6785)	Shows **KeyTip** for command in specified position.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	StyleChanged (see page 6934)	This is StyleChanged, a member of class IRibbonGroup.
V	StyleChanged (see page 6785)	This is StyleChanged, a member of class TCustomRibbonGroup.

Public Nested Types

Name	Description
TRowPosition (see page 6790)	This is nested type Ribbon.TCustomRibbonGroup.TRowPosition.

Public Properties

	Name	Description
R	CanCollapse (see page 6786)	Determine whether group can be displayed collapsed.
R	CollapsedControl (see page 6786)	This is CollapsedControl, a member of class TCustomRibbonGroup.
	GroupVariant (see page 6787)	Current size variant ribbon group is using.
R	GroupWidth (see page 6788)	This is GroupWidth, a member of class TCustomRibbonGroup.
R	HasGallery (see page 6788)	Indicates if group has an in ribbon Gallery.
	Items (see page 6788)	This is Items, a member of class TCustomRibbonGroup.
R	Page (see page 6789)	Reference to Page group is on.
R	RequiredWidth (see page 6789)	Width required to display group.

Published Properties

	Name	Description
	ActionManager (see page 6868)	This is ActionManager, a member of class TRibbonGroup.
	BiDiMode (see page 6868)	This is BiDiMode, a member of class TRibbonGroup.
	Caption (see page 6869)	This is Caption, a member of class TRibbonGroup.
	DialogAction (see page 6869)	Determines whether dialog button displayed in group.
	GroupAlign (see page 6869)	Determines how controls on group are aligned.
	GroupIndex (see page 6869)	Index of group within current tab.
	Height (see page 6870)	This is Height, a member of class TRibbonGroup.
	KeyTip (see page 6870)	Key tip for Dialog button of group.
	ParentBiDiMode (see page 6870)	This is ParentBiDiMode, a member of class TRibbonGroup.
	Rows (see page 6870)	Number of Rows in group has for controls.

76.1.18.2 TRibbonGroup Properties

The properties of the TRibbonGroup class are listed here.

Published Properties

	Name	Description
	ActionManager (see page 6868)	This is ActionManager, a member of class TRibbonGroup.
	BiDiMode (see page 6868)	This is BiDiMode, a member of class TRibbonGroup.
	Caption (see page 6869)	This is Caption, a member of class TRibbonGroup.
	DialogAction (see page 6869)	Determines whether dialog button displayed in group.
	GroupAlign (see page 6869)	Determines how controls on group are aligned.
	GroupIndex (see page 6869)	Index of group within current tab.
	Height (see page 6870)	This is Height, a member of class TRibbonGroup.
	KeyTip (see page 6870)	Key tip for Dialog button of group.
	ParentBiDiMode (see page 6870)	This is ParentBiDiMode, a member of class TRibbonGroup.
	Rows (see page 6870)	Number of Rows in group has for controls.

76.1.18.2.1 TRibbonGroup.ActionManager Property

Pascal

```
property ActionManager;
```

C++

```
__property ActionManager;
```

Description

This is ActionManager, a member of class TRibbonGroup.

76.1.18.2.2 TRibbonGroup.BiDiMode Property

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

This is BiDiMode, a member of class TRibbonGroup.

76.1.18.2.3 TRibbonGroup.Caption Property

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

This is Caption, a member of class TRibbonGroup.

76.1.18.2.4 TRibbonGroup.DialogAction Property

Determines whether dialog button displayed in group.

Pascal

```
property DialogAction: TContainedAction;
```

C++

```
__property TContainedAction DialogAction;
```

Description

DialogAction determines whether the dialog button is displayed in the lower right hand corner of the group.

See Also

KeyTip (see page 6788)

76.1.18.2.5 TRibbonGroup.GroupAlign Property

Determines how controls on group are aligned.

Pascal

```
property GroupAlign: TGroupAlign;
```

C++

```
__property TGroupAlign GroupAlign;
```

Description

GroupAlign contains a TGroupAlign that determines how controls on a group are aligned.

If alignment is **gaHorizontal**, controls are aligned left to right, top to bottom. To make a control start in a new row, set the NewRow property to true.

If alignment is **gaVertical**, controls are aligned top to bottom, left to right. Controls are automatically moved to the next column in the group if three controls are already in the column. You can also force a control to a new column by setting the NewCol property to true.

See Also

TGroupAlign

76.1.18.2.6 TRibbonGroup.GroupIndex Property

Index of group within current tab.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

GroupIndex is the index of the group within the current tab. Set the GroupIndex property to change the position of the group within the current tab.

76.1.18.2.7 TRibbonGroup.Height Property

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

This is Height, a member of class TRibbonGroup.

76.1.18.2.8 TRibbonGroup.KeyTip Property

Key tip for Dialog button of group.

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;__property UnicodeString KeyTip;
```

Description

KeyTip is the key tip for the Dialog button of the group. It is only valid when DialogAction has been set.

See Also

DialogAction (see page 6786)

76.1.18.2.9 TRibbonGroup.ParentBiDiMode Property

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

This is ParentBiDiMode, a member of class TRibbonGroup.

76.1.18.2.10 TRibbonGroup.Rows Property

Number of Rows in group has for controls.

Pascal

```
property Rows: Integer;
```

C++

```
__property int Rows;
```

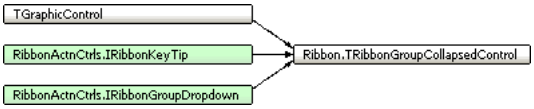
Description

Rows is the number of Rows the group has for displaying controls. A group must have at least 1 row and a maximum of 3.

76.1.19 TRibbonGroupCollapsedControl Class

Control used to represent a collapsed group on a tab.

Class Hierarchy



Pascal

```
TRibbonGroupCollapsedControl = class(TGraphicControl, IRibbonKeyTip, IRibbonGroupDropdown);
```

C++

```
class TRibbonGroupCollapsedControl : public TGraphicControl, public IRibbonKeyTip, public
IRibbonGroupDropdown;
```

File

Ribbon

Description

TRibbonGroupCollapsedControl is a control used to represent a collapsed group on a tab.

This class is used internally by the ribbon group to represent a group in the collapsed state. A group is collapsed when the GroupVariant property of the ribbon group is **rgvCollapsed**. This happens automatically when the ribbon is resized in such a way that the group is no longer completely visible within the current tab.

Note: This class is for internal use only.

See Also

GroupVariant (see page 6787)

76.1.19.1 TRibbonGroupCollapsedControl Members

The following tables list the members exposed by TRibbonGroupCollapsedControl.

Protected Methods

	Name	Description
	AdjustBounds (see page 6874)	This is AdjustBounds, a member of class TRibbonGroupCollapsedControl.
	Click (see page 6874)	Respond to user click.
	CMFontchanged (see page 6875)	This is CMFontchanged, a member of class TRibbonGroupCollapsedControl.
	CMMouseenter (see page 6875)	This is CMMouseenter, a member of class TRibbonGroupCollapsedControl.

	CMMouseleave (see page 6875)	This is CMMouseleave, a member of class TRibbonGroupCollapsedControl.
	CMTextchanged (see page 6875)	This is CMTextchanged, a member of class TRibbonGroupCollapsedControl.
V	Notification (see page 6878)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 6878)	Render control surface.
	SetupGroupImage (see page 6879)	This is SetupGroupImage, a member of class TRibbonGroupCollapsedControl.

Protected Properties

	Name	Description
	MouseInControl (see page 6883)	This is MouseInControl, a member of class TRibbonGroupCollapsedControl.

Public Methods

	Name	Description
	CloseUp (see page 6875)	This is CloseUp, a member of class TRibbonGroupCollapsedControl.
V	Create (see page 6872)	Creates an instance of TGraphicControl.
V	Destroy (see page 6873)	Destroys an instance of TGraphicControl.
	GetKeyTip (see page 6876)	Returns current **KeyTip** for command.
	GetParentControl (see page 6876)	This is GetParentControl, a member of class TRibbonGroupCollapsedControl.
	GetTopPosition (see page 6876)	This is GetTopPosition, a member of class TRibbonGroupCollapsedControl.
	HandleKeyTip (see page 6876)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6877)	Hide **KeyTip** for command.
	KeyTipClick (see page 6877)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6878)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6879)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	ActionManager (see page 6880)	This is ActionManager, a member of class TRibbonGroupCollapsedControl.
	OldWidth (see page 6883)	This is OldWidth, a member of class TRibbonGroupCollapsedControl.

Published Properties

	Name	Description
	Caption (see page 6880)	Specifies a text string that identifies the control to the user.
	Group (see page 6883)	Group associated with collapsed button.
	KeyTip (see page 6883)	Current **KeyTip** for command

76.1.19.2 TRibbonGroupCollapsedControl.Create Constructor

Creates an instance of TGraphicControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TGraphicControl. However, you should never attempt to instantiate a TGraphicControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component, typically a form, that is responsible for freeing the graphic control. It becomes the value of the Owner property. If you do not explicitly provide an Owner to the constructor (that is, if AOwner is nil (Delphi) or NULL (C++)), your application is responsible for explicitly freeing the graphic control.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

76.1.19.3 TRibbonGroupCollapsedControl.Destroy Destructor

Destroys an instance of TGraphicControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonGroupCollapsedControl();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

76.1.19.4 TRibbonGroupCollapsedControl Methods

The methods of the TRibbonGroupCollapsedControl class are listed here.

Protected Methods

	Name	Description
	AdjustBounds (see page 6874)	This is AdjustBounds, a member of class TRibbonGroupCollapsedControl.
V	Click (see page 6874)	Respond to user click.
	CMFontchanged (see page 6875)	This is CMFontchanged, a member of class TRibbonGroupCollapsedControl.
	CMMouseenter (see page 6875)	This is CMMouseenter, a member of class TRibbonGroupCollapsedControl.
	CMMouseleave (see page 6875)	This is CMMouseleave, a member of class TRibbonGroupCollapsedControl.
	CMTextchanged (see page 6875)	This is CMTextchanged, a member of class TRibbonGroupCollapsedControl.
V	Notification (see page 6878)	Responds to notifications that components are being created or destroyed.

	Paint (see page 6878)	Render control surface.
	SetupGroupImage (see page 6879)	This is SetupGroupImage, a member of class TRibbonGroupCollapsedControl.

Public Methods

	Name	Description
	CloseUp (see page 6875)	This is CloseUp, a member of class TRibbonGroupCollapsedControl.
	GetKeyTip (see page 6876)	Returns current **KeyTip** for command.
	GetParentControl (see page 6876)	This is GetParentControl, a member of class TRibbonGroupCollapsedControl.
	GetTopPosition (see page 6876)	This is GetTopPosition, a member of class TRibbonGroupCollapsedControl.
	HandleKeyTip (see page 6876)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6877)	Hide **KeyTip** for command.
	KeyTipClick (see page 6877)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6878)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6879)	Shows **KeyTip** for command in specified position.

76.1.19.4.1 TRibbonGroupCollapsedControl.AdjustBounds Method

Pascal

```
procedure AdjustBounds; dynamic;
```

C++

```
__fastcall AdjustBounds();
```

Description

This is AdjustBounds, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.2 TRibbonGroupCollapsedControl.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

76.1.19.4.3 TRibbonGroupCollapsedControl.CloseUp Method

Pascal

```
procedure CloseUp;
```

C++

```
__fastcall CloseUp();
```

Description

This is CloseUp, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.4 TRibbonGroupCollapsedControl.CMFontchanged Method

Pascal

```
procedure CMFontchanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontchanged(TMessage Message);
```

Description

This is CMFontchanged, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.5 TRibbonGroupCollapsedControl.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.6 TRibbonGroupCollapsedControl.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.7 TRibbonGroupCollapsedControl.CMTextchanged Method

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This is CMTextchanged, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.8 TRibbonGroupCollapsedControl.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.19.4.9 TRibbonGroupCollapsedControl.GetParentControl Method

Pascal

```
function GetParentControl: TWinControl;
```

C++

```
__fastcall TWinControl * GetParentControl();
```

Description

This is GetParentControl, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.10 TRibbonGroupCollapsedControl.GetTopPosition Method

Pascal

```
function GetTopPosition: Integer;
```

C++

```
__fastcall int GetTopPosition();
```

Description

This is GetTopPosition, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.11 TRibbonGroupCollapsedControl.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.19.4.12 TRibbonGroupCollapsedControl.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

76.1.19.4.13 TRibbonGroupCollapsedControl.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

76.1.19.4.14 TRibbonGroupCollapsedControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

76.1.19.4.15 TRibbonGroupCollapsedControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

76.1.19.4.16 TRibbonGroupCollapsedControl.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

76.1.19.4.17 TRibbonGroupCollapsedControl.SetupGroupImage Method

Pascal

```
procedure SetupGroupImage;
```

C++

```
__fastcall SetupGroupImage();
```

Description

This is SetupGroupImage, a member of class TRibbonGroupCollapsedControl.

76.1.19.4.18 TRibbonGroupCollapsedControl.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

76.1.19.5 TRibbonGroupCollapsedControl Properties

The properties of the TRibbonGroupCollapsedControl class are listed here.

Protected Properties

	Name	Description
	MouseInControl (see page 6883)	This is MouseInControl, a member of class TRibbonGroupCollapsedControl.

Public Properties

	Name	Description
	ActionManager (see page 6880)	This is ActionManager, a member of class TRibbonGroupCollapsedControl.
	OldWidth (see page 6883)	This is OldWidth, a member of class TRibbonGroupCollapsedControl.

Published Properties

	Name	Description
	Caption (see page 6880)	Specifies a text string that identifies the control to the user.
	Group (see page 6883)	Group associated with collapsed button.
	KeyTip (see page 6883)	Current **KeyTip** for command

76.1.19.5.1 TRibbonGroupCollapsedControl.ActionManager Property

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

This is ActionManager, a member of class TRibbonGroupCollapsedControl.

76.1.19.5.2 TRibbonGroupCollapsedControl.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
```

```
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
```

```
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

76.1.19.5.3 TRibbonGroupCollapsedControl.Group Property

Group associated with collapsed button.

Pascal

```
property Group: TCustomRibbonGroup;
```

C++

```
__property TCustomRibbonGroup Group;
```

Description

Group is the group associated with the collapsed button.

76.1.19.5.4 TRibbonGroupCollapsedControl.KeyTip Property

Current **KeyTip** for command

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

KeyTip is the current **KeyTip** for the command

See Also

GetKeyTip (see page 7279)

SetKeyTip (see page 7281)

76.1.19.5.5 TRibbonGroupCollapsedControl.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TRibbonGroupCollapsedControl.

76.1.19.5.6 TRibbonGroupCollapsedControl.OldWidth Property

Pascal

```
property OldWidth: Integer;
```

C++

```
__property int OldWidth;
```

Description

This is OldWidth, a member of class TRibbonGroupCollapsedControl.

76.1.20 TRibbonGroupDockForm Class

Display a collapsed group.

Class Hierarchy

Pascal

```
TRibbonGroupDockForm = class(TXToolDockForm);
```

C++

```
class TRibbonGroupDockForm : public TXToolDockForm;
```

File

Ribbon

Description

TRibbonGroupDockForm is used to display a collapsed group.

When the user clicks on a groups collapsed button, the large group variant of the ribbon group is displayed as a child to an instance of this class.

Note: This class is for internal use only.

76.1.20.1 TRibbonGroupDockForm Members

The following tables list the members exposed by TRibbonGroupDockForm.

Protected Methods

	Name	Description
	CreateParams (see page 6885)	Initializes the window-creation parameter record when the form window is created.
	WMActivate (see page 6885)	This is WMActivate, a member of class TRibbonGroupDockForm.

Public Methods

	Name	Description
	Create (see page 6885)	Creates an instance of TToolDockForm.

Public Properties

	Name	Description
	Control (see page 6886)	Reference to invoking control.
	ParentControl (see page 6886)	This is ParentControl, a member of class TRibbonGroupDockForm.

76.1.20.2 TRibbonGroupDockForm.Create Constructor

Creates an instance of TToolDockForm.

Pascal

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

C++

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

Description

AOwner is the dock form's owner, which is responsible for freeing the tool dock form.

The constructor initializes the BorderWidth property to 1.

See Also

BorderWidth

Owner

76.1.20.3 TRibbonGroupDockForm Methods

The methods of the TRibbonGroupDockForm class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 6885)	Initializes the window-creation parameter record when the form window is created.
	WMActivate (see page 6885)	This is WMActivate, a member of class TRibbonGroupDockForm.

76.1.20.3.1 TRibbonGroupDockForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams calls the inherited CreateParams method, and then adds WS_POPUP to the Style property.

76.1.20.3.2 TRibbonGroupDockForm.WMActivate Method

Pascal

```
procedure WMActivate(var Message: TWMActivate); message WM_ACTIVATE;
```

C++

```
__fastcall WMActivate(TWMActivate Message);
```

Description

This is WMActivate, a member of class TRibbonGroupDockForm.

76.1.20.4 TRibbonGroupDockForm Properties

The properties of the TRibbonGroupDockForm class are listed here.

Public Properties

	Name	Description
	Control (see page 6886)	Reference to invoking control.
	ParentControl (see page 6886)	This is ParentControl, a member of class TRibbonGroupDockForm.

76.1.20.4.1 TRibbonGroupDockForm.Control Property

Reference to invoking control.

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

Control is a reference to the control that invoked the floating group.

76.1.20.4.2 TRibbonGroupDockForm.ParentControl Property

Pascal

```
property ParentControl: TWinControl;
```

C++

```
__property TWinControl * ParentControl;
```

Description

This is ParentControl, a member of class TRibbonGroupDockForm.

76.1.21 TRibbonGroupScrollButton Class

Class Hierarchy

Pascal

```
TRibbonGroupScrollButton = class(TCustomControl);
```

C++

```
class TRibbonGroupScrollButton : public TCustomControl;
```

File

Ribbon

Description

This is class Ribbon.TRibbonGroupScrollButton.

76.1.21.1 TRibbonGroupScrollButton Members

The following tables list the members exposed by TRibbonGroupScrollButton.

Protected Methods

	Name	Description
	CMDesignHitTest (see page 6888)	This is CMDesignHitTest, a member of class TRibbonGroupScrollButton.
	CMMouseenter (see page 6888)	This is CMMouseenter, a member of class TRibbonGroupScrollButton.
	CMMouseleave (see page 6888)	This is CMMouseleave, a member of class TRibbonGroupScrollButton.
	Paint (see page 6888)	Renders the image of a custom control.

Protected Properties

	Name	Description
	Direction (see page 6889)	This is Direction, a member of class TRibbonGroupScrollButton.
	MouseInControl (see page 6889)	This is MouseInControl, a member of class TRibbonGroupScrollButton.
	RibbonStyle (see page 6890)	This is RibbonStyle, a member of class TRibbonGroupScrollButton.

Public Methods

	Name	Description
	Create (see page 6887)	Creates an instance of TCustomControl.

76.1.21.2 TRibbonGroupScrollButton.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

76.1.21.3 TRibbonGroupScrollButton Methods

The methods of the TRibbonGroupScrollButton class are listed here.

Protected Methods

	Name	Description
	CMDesignHitTest (see page 6888)	This is CMDesignHitTest, a member of class TRibbonGroupScrollButton.
	CMMouseenter (see page 6888)	This is CMMouseenter, a member of class TRibbonGroupScrollButton.
	CMMouseleave (see page 6888)	This is CMMouseleave, a member of class TRibbonGroupScrollButton.
	Paint (see page 6888)	Renders the image of a custom control.

76.1.21.3.1 TRibbonGroupScrollButton.CMDesignHitTest Method

Pascal

```
procedure CMDesignHitTest(var Message: TCMDesignHitTest); message CM_DESIGNHITTEST;
```

C++

```
__fastcall CMDesignHitTest(TCMDesignHitTest Message);
```

Description

This is CMDesignHitTest, a member of class TRibbonGroupScrollButton.

76.1.21.3.2 TRibbonGroupScrollButton.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TRibbonGroupScrollButton.

76.1.21.3.3 TRibbonGroupScrollButton.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TRibbonGroupScrollButton.

76.1.21.3.4 TRibbonGroupScrollButton.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect (see page 2718)

76.1.21.4 TRibbonGroupScrollButton Properties

The properties of the TRibbonGroupScrollButton class are listed here.

Protected Properties

	Name	Description
	Direction (see page 6889)	This is Direction, a member of class TRibbonGroupScrollButton.
	MouseInControl (see page 6889)	This is MouseInControl, a member of class TRibbonGroupScrollButton.
	RibbonStyle (see page 6890)	This is RibbonStyle, a member of class TRibbonGroupScrollButton.

76.1.21.4.1 TRibbonGroupScrollButton.Direction Property

Pascal

```
property Direction: TScrollDirection;
```

C++

```
__property TScrollDirection Direction;
```

Description

This is Direction, a member of class TRibbonGroupScrollButton.

76.1.21.4.2 TRibbonGroupScrollButton.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TRibbonGroupScrollButton.

76.1.21.4.3 TRibbonGroupScrollButton.RibbonStyle Property

Pascal

```
property RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__property TRibbonStyleActionBars RibbonStyle;
```

Description

This is RibbonStyle, a member of class TRibbonGroupScrollButton.

76.1.22 TRibbonLunaColorMap Class

Color map class for Luna.

Class Hierarchy

Pascal

```
TRibbonLunaColorMap = class(TCustomRibbonColorMap);
```

C++

```
class TRibbonLunaColorMap : public TCustomRibbonColorMap;
```

File

Ribbon

Description

TRibbonLunaColorMap is the color map class for the Luna ribbon style.

See Also

TRibbonObsidianColorMap (see page 6896)

TRibbonSilverColorMap (see page 6914)

76.1.22.1 TRibbonLunaColorMap Members

The following tables list the members exposed by TRibbonLunaColorMap.

Public Methods

	Name	Description
	UpdateColors (see page 6891)	Updates an action component to reflect the current color property.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6892)	This is ActiveTabFontColor, a member of class TRibbonLunaColorMap.

	BevelMinimizedInner (see page 6892)	This is BevelMinimizedInner, a member of class TRibbonLunaColorMap.
	BevelMinimizedOuter (see page 6892)	This is BevelMinimizedOuter, a member of class TRibbonLunaColorMap.
	BtnSelectedFont (see page 6892)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6893)	This is CaptionFontColor, a member of class TRibbonLunaColorMap.
	ControlColor (see page 6893)	This is ControlColor, a member of class TRibbonLunaColorMap.
	ControlFrameColor (see page 6893)	This is ControlFrameColor, a member of class TRibbonLunaColorMap.
	ControlInactiveColor (see page 6893)	This is ControlInactiveColor, a member of class TRibbonLunaColorMap.
	DisabledFontColor (see page 6894)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6894)	This is DocumentFontColor, a member of class TRibbonLunaColorMap.
	FontColor (see page 6894)	Changes the look and feel of your applications.
	GroupFontColor (see page 6894)	This is GroupFontColor, a member of class TRibbonLunaColorMap.
	HighlightColor (see page 6895)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6895)	This is InactiveTabFontColor, a member of class TRibbonLunaColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6895)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonLunaColorMap.
	QuickAccessToolbarMoreColor (see page 6895)	This is QuickAccessToolbarMoreColor, a member of class TRibbonLunaColorMap.

76.1.22.2 TRibbonLunaColorMap Methods

The methods of the TRibbonLunaColorMap class are listed here.

Public Methods

	Name	Description
	UpdateColors (see page 6891)	Updates an action component to reflect the current color property.

76.1.22.2.1 TRibbonLunaColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

76.1.22.3 TRibbonLunaColorMap Properties

The properties of the TRibbonLunaColorMap class are listed here.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6892)	This is ActiveTabFontColor, a member of class TRibbonLunaColorMap.
	BevelMinimizedInner (see page 6892)	This is BevelMinimizedInner, a member of class TRibbonLunaColorMap.
	BevelMinimizedOuter (see page 6892)	This is BevelMinimizedOuter, a member of class TRibbonLunaColorMap.
	BtnSelectedFont (see page 6892)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6893)	This is CaptionFontColor, a member of class TRibbonLunaColorMap.
	ControlColor (see page 6893)	This is ControlColor, a member of class TRibbonLunaColorMap.
	ControlFrameColor (see page 6893)	This is ControlFrameColor, a member of class TRibbonLunaColorMap.

	ControlInactiveColor (see page 6893)	This is ControlInactiveColor, a member of class TRibbonLunaColorMap.
	DisabledFontColor (see page 6894)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6894)	This is DocumentFontColor, a member of class TRibbonLunaColorMap.
	FontColor (see page 6894)	Changes the look and feel of your applications.
	GroupFontColor (see page 6894)	This is GroupFontColor, a member of class TRibbonLunaColorMap.
	HighlightColor (see page 6895)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6895)	This is InactiveTabFontColor, a member of class TRibbonLunaColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6895)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonLunaColorMap.
	QuickAccessToolbarMoreColor (see page 6895)	This is QuickAccessToolbarMoreColor, a member of class TRibbonLunaColorMap.

76.1.22.3.1 TRibbonLunaColorMap.ActiveTabFontColor Property

Pascal

```
property ActiveTabFontColor: TColor;
```

C++

```
__property TColor ActiveTabFontColor;
```

Description

This is ActiveTabFontColor, a member of class TRibbonLunaColorMap.

76.1.22.3.2 TRibbonLunaColorMap.BevelMinimizedInner Property

Pascal

```
property BevelMinimizedInner: TColor;
```

C++

```
__property TColor BevelMinimizedInner;
```

Description

This is BevelMinimizedInner, a member of class TRibbonLunaColorMap.

76.1.22.3.3 TRibbonLunaColorMap.BevelMinimizedOuter Property

Pascal

```
property BevelMinimizedOuter: TColor;
```

C++

```
__property TColor BevelMinimizedOuter;
```

Description

This is BevelMinimizedOuter, a member of class TRibbonLunaColorMap.

76.1.22.3.4 TRibbonLunaColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont;
```

C++

```
__property BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.22.3.5 TRibbonLunaColorMap.CaptionFontColor Property

Pascal

```
property CaptionFontColor: TColor;
```

C++

```
__property TColor CaptionFontColor;
```

Description

This is CaptionFontColor, a member of class TRibbonLunaColorMap.

76.1.22.3.6 TRibbonLunaColorMap.ControlColor Property

Pascal

```
property ControlColor: TColor;
```

C++

```
__property TColor ControlColor;
```

Description

This is ControlColor, a member of class TRibbonLunaColorMap.

76.1.22.3.7 TRibbonLunaColorMap.ControlFrameColor Property

Pascal

```
property ControlFrameColor: TColor;
```

C++

```
__property TColor ControlFrameColor;
```

Description

This is ControlFrameColor, a member of class TRibbonLunaColorMap.

76.1.22.3.8 TRibbonLunaColorMap.ControlInactiveColor Property

Pascal

```
property ControlInactiveColor: TColor;
```

C++

```
__property TColor ControlInactiveColor;
```

Description

This is ControlInactiveColor, a member of class TRibbonLunaColorMap.

76.1.22.3.9 TRibbonLunaColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor;
```

C++

```
__property DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.22.3.10 TRibbonLunaColorMap.DocumentFontColor Property

Pascal

```
property DocumentFontColor: TColor;
```

C++

```
__property TColor DocumentFontColor;
```

Description

This is DocumentFontColor, a member of class TRibbonLunaColorMap.

76.1.22.3.11 TRibbonLunaColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor;
```

C++

```
__property FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.22.3.12 TRibbonLunaColorMap.GroupFontColor Property

Pascal

```
property GroupFontColor: TColor;
```

C++

```
__property TColor GroupFontColor;
```

Description

This is GroupFontColor, a member of class TRibbonLunaColorMap.

76.1.22.3.13 TRibbonLunaColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor;
```

C++

```
__property HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.22.3.14 TRibbonLunaColorMap.InactiveTabFontColor Property

Pascal

```
property InactiveTabFontColor: TColor;
```

C++

```
__property TColor InactiveTabFontColor;
```

Description

This is InactiveTabFontColor, a member of class TRibbonLunaColorMap.

76.1.22.3.15 TRibbonLunaColorMap.QuickAccessToolbarMoreBorderColor Property

Pascal

```
property QuickAccessToolbarMoreBorderColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreBorderColor;
```

Description

This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonLunaColorMap.

76.1.22.3.16 TRibbonLunaColorMap.QuickAccessToolbarMoreColor Property

Pascal

```
property QuickAccessToolbarMoreColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreColor;
```

Description

This is QuickAccessToolbarMoreColor, a member of class TRibbonLunaColorMap.

76.1.23 TRibbonObsidianColorMap Class

Color map class for Obsidian.

Class Hierarchy

Pascal

```
TRibbonObsidianColorMap = class(TCustomRibbonColorMap);
```

C++

```
class TRibbonObsidianColorMap : public TCustomRibbonColorMap;
```

File

Ribbon

Description

TRibbonObsidianColorMap is the color map class for the Obsidian ribbon style.

See Also

TRibbonLunaColorMap (see page 6890)

TRibbonSilverColorMap (see page 6914)

76.1.23.1 TRibbonObsidianColorMap Members

The following tables list the members exposed by TRibbonObsidianColorMap.

Public Methods

	Name	Description
	UpdateColors (see page 6897)	Updates an action component to reflect the current color property.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6898)	This is ActiveTabFontColor, a member of class TRibbonObsidianColorMap.
	BevelMinimizedInner (see page 6898)	This is BevelMinimizedInner, a member of class TRibbonObsidianColorMap.
	BevelMinimizedOuter (see page 6898)	This is BevelMinimizedOuter, a member of class TRibbonObsidianColorMap.
	BtnSelectedFont (see page 6898)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6898)	This is CaptionFontColor, a member of class TRibbonObsidianColorMap.
	ControlColor (see page 6899)	This is ControlColor, a member of class TRibbonObsidianColorMap.
	ControlFrameColor (see page 6899)	This is ControlFrameColor, a member of class TRibbonObsidianColorMap.
	ControlInactiveColor (see page 6899)	This is ControlInactiveColor, a member of class TRibbonObsidianColorMap.
	DisabledFontColor (see page 6899)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6900)	This is DocumentFontColor, a member of class TRibbonObsidianColorMap.
	FontColor (see page 6900)	Changes the look and feel of your applications.
	GroupFontColor (see page 6900)	This is GroupFontColor, a member of class TRibbonObsidianColorMap.
	HighlightColor (see page 6900)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6901)	This is InactiveTabFontColor, a member of class TRibbonObsidianColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6901)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonObsidianColorMap.

	QuickAccessToolbarMoreColor (see page 6901)	This is QuickAccessToolbarMoreColor, a member of class TRibbonObsidianColorMap.

76.1.23.2 TRibbonObsidianColorMap Methods

The methods of the TRibbonObsidianColorMap class are listed here.

Public Methods

	Name	Description
	UpdateColors (see page 6897)	Updates an action component to reflect the current color property.

76.1.23.2.1 TRibbonObsidianColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

76.1.23.3 TRibbonObsidianColorMap Properties

The properties of the TRibbonObsidianColorMap class are listed here.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6898)	This is ActiveTabFontColor, a member of class TRibbonObsidianColorMap.
	BevelMinimizedInner (see page 6898)	This is BevelMinimizedInner, a member of class TRibbonObsidianColorMap.
	BevelMinimizedOuter (see page 6898)	This is BevelMinimizedOuter, a member of class TRibbonObsidianColorMap.
	BtnSelectedFont (see page 6898)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6898)	This is CaptionFontColor, a member of class TRibbonObsidianColorMap.
	ControlColor (see page 6899)	This is ControlColor, a member of class TRibbonObsidianColorMap.
	ControlFrameColor (see page 6899)	This is ControlFrameColor, a member of class TRibbonObsidianColorMap.
	ControlInactiveColor (see page 6899)	This is ControlInactiveColor, a member of class TRibbonObsidianColorMap.
	DisabledFontColor (see page 6899)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6900)	This is DocumentFontColor, a member of class TRibbonObsidianColorMap.
	FontColor (see page 6900)	Changes the look and feel of your applications.
	GroupFontColor (see page 6900)	This is GroupFontColor, a member of class TRibbonObsidianColorMap.
	HighlightColor (see page 6900)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6901)	This is InactiveTabFontColor, a member of class TRibbonObsidianColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6901)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonObsidianColorMap.
	QuickAccessToolbarMoreColor (see page 6901)	This is QuickAccessToolbarMoreColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.1 TRibbonObsidianColorMap.ActiveTabFontColor Property

Pascal

```
property ActiveTabFontColor: TColor;
```

C++

```
__property TColor ActiveTabFontColor;
```

Description

This is ActiveTabFontColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.2 TRibbonObsidianColorMap.BevelMinimizedInner Property

Pascal

```
property BevelMinimizedInner: TColor;
```

C++

```
__property TColor BevelMinimizedInner;
```

Description

This is BevelMinimizedInner, a member of class TRibbonObsidianColorMap.

76.1.23.3.3 TRibbonObsidianColorMap.BevelMinimizedOuter Property

Pascal

```
property BevelMinimizedOuter: TColor;
```

C++

```
__property TColor BevelMinimizedOuter;
```

Description

This is BevelMinimizedOuter, a member of class TRibbonObsidianColorMap.

76.1.23.3.4 TRibbonObsidianColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont;
```

C++

```
__property BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.23.3.5 TRibbonObsidianColorMap.CaptionFontColor Property

Pascal

```
property CaptionFontColor: TColor;
```

C++

```
__property TColor CaptionFontColor;
```

Description

This is CaptionFontColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.6 TRibbonObsidianColorMap.ControlColor Property

Pascal

```
property ControlColor: TColor;
```

C++

```
__property TColor ControlColor;
```

Description

This is ControlColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.7 TRibbonObsidianColorMap.ControlFrameColor Property

Pascal

```
property ControlFrameColor: TColor;
```

C++

```
__property TColor ControlFrameColor;
```

Description

This is ControlFrameColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.8 TRibbonObsidianColorMap.ControlInactiveColor Property

Pascal

```
property ControlInactiveColor: TColor;
```

C++

```
__property TColor ControlInactiveColor;
```

Description

This is ControlInactiveColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.9 TRibbonObsidianColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor;
```

C++

```
__property DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.23.3.10 TRibbonObsidianColorMap.DocumentFontColor Property

Pascal

```
property DocumentFontColor: TColor;
```

C++

```
__property TColor DocumentFontColor;
```

Description

This is DocumentFontColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.11 TRibbonObsidianColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor;
```

C++

```
__property FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.23.3.12 TRibbonObsidianColorMap.GroupFontColor Property

Pascal

```
property GroupFontColor: TColor;
```

C++

```
__property TColor GroupFontColor;
```

Description

This is GroupFontColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.13 TRibbonObsidianColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor;
```

C++

```
__property HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.23.3.14 TRibbonObsidianColorMap.InactiveTabFontColor Property

Pascal

```
property InactiveTabFontColor: TColor;
```

C++

```
__property TColor InactiveTabFontColor;
```

Description

This is InactiveTabFontColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.15 TRibbonObsidianColorMap.QuickAccessToolbarMoreBorderColor Property

Pascal

```
property QuickAccessToolbarMoreBorderColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreBorderColor;
```

Description

This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonObsidianColorMap.

76.1.23.3.16 TRibbonObsidianColorMap.QuickAccessToolbarMoreColor Property

Pascal

```
property QuickAccessToolbarMoreColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreColor;
```

Description

This is QuickAccessToolbarMoreColor, a member of class TRibbonObsidianColorMap.

76.1.24 TRibbonPage Class

Tab page for ribbon control.

Class Hierarchy

Pascal

```
TRibbonPage = class(TCustomRibbonPage);
```

C++

```
class TRibbonPage : public TCustomRibbonPage;
```

File

Ribbon

Description

TRibbonPage is a TCustomRibbonPage (see page 6790) tab page for the ribbon control that holds the groups of controls displayed.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TCustomRibbonPage (see page 6790)

76.1.24.1 TRibbonPage Members

The following tables list the members exposed by TRibbonPage.

Protected Methods

	Name	Description
	CMControlListChanging (see page 6795)	This is CMControlListChanging, a member of class TCustomRibbonPage.
	CMTextchanged (see page 6795)	This is CMTextchanged, a member of class TCustomRibbonPage.
V	CreateScrollButton (see page 6795)	This is CreateScrollButton, a member of class TCustomRibbonPage.
V	DoScrollButtonClick (see page 6795)	This is DoScrollButtonClick, a member of class TCustomRibbonPage.
	GetPageHeight (see page 6797)	This is GetPageHeight, a member of class TCustomRibbonPage.
	GetPageTop (see page 6797)	This is GetPageTop, a member of class TCustomRibbonPage.
V	Loaded (see page 6798)	Initializes the control after it is loaded from a stream.
V	Notification (see page 6798)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 6799)	Renders the image of a custom control.
V	RequestAlign (see page 6799)	Instructs the parent of a control to reposition the control, enforcing its Align property.
V	Resize (see page 6800)	Respond to control resize.
V	SetParent (see page 6801)	Sets the parent of the control.
	UpdateScrollButtons (see page 6802)	This is UpdateScrollButtons, a member of class TCustomRibbonPage.

Protected Properties

	Name	Description
	GroupOffset (see page 6803)	This is GroupOffset, a member of class TCustomRibbonPage.

Public Constants

	Name	Description
	cRibbonGroupLeftOffset (see page 6793)	This is cRibbonGroupLeftOffset, a member of class TCustomRibbonPage.
	cRibbonGroupSpacing (see page 6793)	This is cRibbonGroupSpacing, a member of class TCustomRibbonPage.

Public Methods

	Name	Description
	AddGroup (see page 6794)	This is AddGroup, a member of class TCustomRibbonPage.
	AlignGroups (see page 6794)	This is AlignGroups, a member of class TCustomRibbonPage.
	CollapseGroup (see page 6795)	This is CollapseGroup, a member of class TCustomRibbonPage.
V	Create (see page 6792)	Creates an instance of TCustomControl.
V	Destroy (see page 6792)	Destroys an instance of TCustomControl.
	FindGroup (see page 6796)	This is FindGroup, a member of class TCustomRibbonPage.
	FindGroupAt (see page 6796)	This is FindGroupAt, a member of class TCustomRibbonPage.

	GetGroupPos (see page 6796)	This is GetGroupPos, a member of class TCustomRibbonPage.
	GetNextVisibleGroup (see page 6796)	This is GetNextVisibleGroup, a member of class TCustomRibbonPage.
	HandleKeyTips (see page 6797)	This is HandleKeyTips, a member of class TCustomRibbonPage.
	HideKeyTips (see page 6797)	This is HideKeyTips, a member of class TCustomRibbonPage.
V	Invalidate (see page 6797)	Schedules a control repaint.
	RibbonStyle (see page 6800)	Return style class to use when drawing specific Ribbon (see page 6804) elements.
V	SetBounds (see page 6800)	Sets the windowed control's boundary properties all at once.
	ShowKeyTips (see page 6802)	This is ShowKeyTips, a member of class TCustomRibbonPage.
	UpdateActionManager (see page 6802)	This is UpdateActionManager, a member of class TCustomRibbonPage.

Public Properties

	Name	Description
R	CollapseGroupCount (see page 6803)	This is CollapseGroupCount, a member of class TCustomRibbonPage.
R	GroupCount (see page 6803)	This is GroupCount, a member of class TCustomRibbonPage.
R	Groups (see page 6803)	This is Groups, a member of class TCustomRibbonPage.
	Ribbon (see page 6804)	This is Ribbon, a member of class TCustomRibbonPage.

Published Properties

	Name	Description
	BiDiMode (see page 6903)	Specifies the bi-directional mode for the control.
	Caption (see page 6904)	Specifies a text string that identifies the control to the user.
	Height (see page 6906)	Specifies the vertical size of the control in pixels.
	Index (see page 6908)	This is Index, a member of class TRibbonPage.
	KeyTip (see page 6908)	This is KeyTip, a member of class TRibbonPage.
	Name (see page 6908)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 6910)	Specifies whether the control uses its parent's BiDiMode.

76.1.24.2 TRibbonPage Properties

The properties of the TRibbonPage class are listed here.

Published Properties

	Name	Description
	BiDiMode (see page 6903)	Specifies the bi-directional mode for the control.
	Caption (see page 6904)	Specifies a text string that identifies the control to the user.
	Height (see page 6906)	Specifies the vertical size of the control in pixels.
	Index (see page 6908)	This is Index, a member of class TRibbonPage.
	KeyTip (see page 6908)	This is KeyTip, a member of class TRibbonPage.
	Name (see page 6908)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 6910)	Specifies whether the control uses its parent's BiDiMode.

76.1.24.2.1 TRibbonPage.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale

that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

76.1.24.2.2 TRibbonPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
```

```
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
  This example creates a ToolBar and adds ToolButtons to it.
  Include ComCtlrs in the uses clause.
  }
  procedure AddButtons(ToolBar: TToolBar;
    Const ButtonCaptions: array of String);
  var
    I: Integer;
  begin
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

76.1.24.2.3 TRibbonPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

76.1.24.2.4 TRibbonPage.Index Property

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

This is Index, a member of class TRibbonPage.

76.1.24.2.5 TRibbonPage.KeyTip Property

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

This is KeyTip, a member of class TRibbonPage.

76.1.24.2.6 TRibbonPage.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name;
```

C++

```
__property Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
```

```
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
```

```
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
  end;
end;
```

See Also

ChangeName

SetName

ValidateRename

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

76.1.24.2.7 TRibbonPage.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

76.1.25 TRibbonQuickAccessPopupToolbar Class

Class Hierarchy

Pascal

```
TRibbonQuickAccessPopupToolbar = class(TCustomRibbonQuickAccessPopupToolbar);
```

C++

```
class TRibbonQuickAccessPopupToolbar : public TCustomRibbonQuickAccessPopupToolbar;
```

File

Ribbon

Description

This is class Ribbon.TRibbonQuickAccessPopupToolbar.

76.1.26 TRibbonQuickAccessToolbar Class

Quck Access Toolbar.

Class Hierarchy

Pascal

```
TRibbonQuickAccessToolbar = class(TCustomRibbonQuickAccessToolbar);
```

C++

```
class TRibbonQuickAccessToolbar : public TCustomRibbonQuickAccessToolbar;
```

File

Ribbon

Description

TRibbonQuickAccessToolbar is a TCustomRibbonQuickAccessToolbar (see page 6807) Quck Access Toolbar implementation.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

TCustomRibbonQuickAccessToolbar (see page 6807)

76.1.26.1 TRibbonQuickAccessToolbar Members

The following tables list the members exposed by TRibbonQuickAccessToolbar.

Protected Methods

	Name	Description
	CMFontchanged (see page 6811)	This is CMFontchanged, a member of class TCustomRibbonQuickAccessToolbar.
V	CreateControls (see page 6811)	This is CreateControls, a member of class TCustomRibbonQuickAccessToolbar.
V	DoContextPopup (see page 6812)	OnContextPopup event dispatcher.
V	DoDropActions (see page 6812)	This is DoDropActions, a member of class TCustomRibbonQuickAccessToolbar.
V	DoDropItem (see page 6813)	This is DoDropItem, a member of class TCustomRibbonQuickAccessToolbar.
V	DoMoreCommands (see page 6813)	This is DoMoreCommands, a member of class TCustomRibbonQuickAccessToolbar.
V	DoScrollBtnClick (see page 6813)	This is DoScrollBtnClick, a member of class TCustomRibbonQuickAccessToolbar.
V	DrawBackground (see page 6814)	This is DrawBackground, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarHeight (see page 6814)	This is GetBarHeight, a member of class TCustomRibbonQuickAccessToolbar.
V	GetBarWidth (see page 6815)	This is GetBarWidth, a member of class TCustomRibbonQuickAccessToolbar.
V	GetScrollBtnClass (see page 6815)	This is GetScrollBtnClass, a member of class TCustomRibbonQuickAccessToolbar.
V	Notification (see page 6815)	Responds to notifications that components are being created or destroyed.
V	RequestAlign (see page 6815)	Instructs the parent of a control to reposition the control, enforcing its Align property.
V	SetParent (see page 6816)	Sets the parent of the control.
V	SetupDropDownBtn (see page 6816)	This is SetupDropDownBtn, a member of class TCustomRibbonQuickAccessToolbar.
V	ShowCustomizeMenu (see page 6817)	This is ShowCustomizeMenu, a member of class TCustomRibbonQuickAccessToolbar.
V	ShowMoreControls (see page 6817)	This is ShowMoreControls, a member of class TCustomRibbonQuickAccessToolbar.
V	UpdateQATWidth (see page 6817)	This is UpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.

Public Methods

	Name	Description
	BarHandleKeyTips (see page 6810)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHandleKeyTips (see page 7273)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 6810)	Hide any visible **KeyTip** windows.
	BarHideKeyTips (see page 7273)	Hide any visible **KeyTip** windows.
	BarShowKeyTips (see page 6811)	Show **KeyTip**s for any commands that support them.
	BarShowKeyTips (see page 7274)	Show **KeyTip**s for any commands that support them.
V	Create (see page 6809)	Creates and initializes a TCustomActionToolBar instance.
V	CreateControl (see page 6811)	This is CreateControl, a member of class TCustomRibbonQuickAccessToolbar.
	Customize (see page 6812)	This is Customize, a member of class TCustomRibbonQuickAccessToolbar.
V	DoDropCategory (see page 6813)	This is DoDropCategory, a member of class TCustomRibbonQuickAccessToolbar.
	DoUpdateQATWidth (see page 6935)	This is DoUpdateQATWidth, a member of class IRibbonQuickAccessToolbar.
	DoUpdateQATWidth (see page 6813)	This is DoUpdateQATWidth, a member of class TCustomRibbonQuickAccessToolbar.
V	DragDrop (see page 6814)	Generates an OnDragDrop event.

Public Nested Types

Name	Description
TMoreCommandsEvent (see page 6819)	This is nested type Ribbon.TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent.

Public Properties

	Name	Description
	Items (see page 6818)	Lists the action clients that correspond to the child controls of this action band.
R	Ribbon (see page 6818)	This is Ribbon, a member of class TCustomRibbonQuickAccessToolbar.
	ScrollMode (see page 6818)	This is ScrollMode, a member of class TCustomRibbonQuickAccessToolbar.
	ShowBorder (see page 6819)	This is ShowBorder, a member of class TCustomRibbonQuickAccessToolbar.

Published Properties

	Name	Description
	ActionManager (see page 6913)	Specifies the action manager component that is associated with the action band.
	OnMoreCommands (see page 6913)	This is OnMoreCommands, a member of class TRibbonQuickAccessToolbar.
	VertMargin (see page 6914)	Specifies the width of the vertical margins in pixels.

76.1.26.2 TRibbonQuickAccessToolbar Properties

The properties of the TRibbonQuickAccessToolbar class are listed here.

Published Properties

	Name	Description
	ActionManager (see page 6913)	Specifies the action manager component that is associated with the action band.
	OnMoreCommands (see page 6913)	This is OnMoreCommands, a member of class TRibbonQuickAccessToolbar.
	VertMargin (see page 6914)	Specifies the width of the vertical margins in pixels.

76.1.26.2.1 TRibbonQuickAccessToolbar.ActionManager Property

Specifies the action manager component that is associated with the action band.

Pascal

```
property ActionManager;
```

C++

```
__property ActionManager;
```

Description

Use ActionManager to access the action manager that uses this action band to render UI elements it manages. Typically, this association is created when you drag actions from the action manager and drop them on the action bar. When an action is dropped on an action band and the ActionManager property is not assigned, the ActionManager property is automatically assigned to the action manager that manages the action that was dropped on the action band.

At runtime, you can perform the equivalent operation to dragging an action onto the action band by setting the ActionBar property of one of the action clients that the action manager's ActionBars property lists so that it specifies this component.

See Also

ActionBars

ActionClient

ActionBar

76.1.26.2.2 TRibbonQuickAccessToolbar.OnMoreCommands Property

Pascal

```
property OnMoreCommands: TMoreCommandsEvent;
```

C++

```
__property TMoreCommandsEvent OnMoreCommands;
```

Description

This is OnMoreCommands, a member of class TRibbonQuickAccessToolbar.

76.1.26.2.3 TRibbonQuickAccessToolbar.VertMargin Property

Specifies the width of the vertical margins in pixels.

Pascal

```
property VertMargin;
```

C++

```
__property VertMargin;
```

Description

VertMargin specifies the width of the action band's top and bottom vertical margins in pixels.

See Also

HorzMargin

HorzSeparator

76.1.27 TRibbonSilverColorMap Class

Color map class for Silver.

Class Hierarchy

Pascal

```
TRibbonSilverColorMap = class(TCustomRibbonColorMap);
```

C++

```
class TRibbonSilverColorMap : public TCustomRibbonColorMap;
```

File

Ribbon

Description

TRibbonSilverColorMap is the color map class for the Silver ribbon style.

See Also

TRibbonLunaColorMap (see page 6890)

TRibbonObsidianColorMap (see page 6896)

76.1.27.1 TRibbonSilverColorMap Members

The following tables list the members exposed by TRibbonSilverColorMap.

Public Methods

	Name	Description
	UpdateColors (see page 6915)	Updates an action component to reflect the current color property.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6916)	This is ActiveTabFontColor, a member of class TRibbonSilverColorMap.
	BevelMinimizedInner (see page 6916)	This is BevelMinimizedInner, a member of class TRibbonSilverColorMap.
	BevelMinimizedOuter (see page 6916)	This is BevelMinimizedOuter, a member of class TRibbonSilverColorMap.
	BtnSelectedFont (see page 6917)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6917)	This is CaptionFontColor, a member of class TRibbonSilverColorMap.
	ControlColor (see page 6917)	This is ControlColor, a member of class TRibbonSilverColorMap.
	ControlFrameColor (see page 6917)	This is ControlFrameColor, a member of class TRibbonSilverColorMap.
	ControlInactiveColor (see page 6917)	This is ControlInactiveColor, a member of class TRibbonSilverColorMap.
	DisabledFontColor (see page 6918)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6918)	This is DocumentFontColor, a member of class TRibbonSilverColorMap.
	FontColor (see page 6918)	Changes the look and feel of your applications.
	GroupFontColor (see page 6918)	This is GroupFontColor, a member of class TRibbonSilverColorMap.
	HighlightColor (see page 6919)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6919)	This is InactiveTabFontColor, a member of class TRibbonSilverColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6919)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonSilverColorMap.
	QuickAccessToolbarMoreColor (see page 6919)	This is QuickAccessToolbarMoreColor, a member of class TRibbonSilverColorMap.

76.1.27.2 TRibbonSilverColorMap Methods

The methods of the TRibbonSilverColorMap class are listed here.

Public Methods

	Name	Description
	UpdateColors (see page 6915)	Updates an action component to reflect the current color property.

76.1.27.2.1 TRibbonSilverColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

76.1.27.3 TRibbonSilverColorMap Properties

The properties of the TRibbonSilverColorMap class are listed here.

Published Properties

	Name	Description
	ActiveTabFontColor (see page 6916)	This is ActiveTabFontColor, a member of class TRibbonSilverColorMap.
	BevelMinimizedInner (see page 6916)	This is BevelMinimizedInner, a member of class TRibbonSilverColorMap.
	BevelMinimizedOuter (see page 6916)	This is BevelMinimizedOuter, a member of class TRibbonSilverColorMap.
	BtnSelectedFont (see page 6917)	Changes the look and feel of your applications.
	CaptionFontColor (see page 6917)	This is CaptionFontColor, a member of class TRibbonSilverColorMap.
	ControlColor (see page 6917)	This is ControlColor, a member of class TRibbonSilverColorMap.
	ControlFrameColor (see page 6917)	This is ControlFrameColor, a member of class TRibbonSilverColorMap.
	ControlInactiveColor (see page 6917)	This is ControlInactiveColor, a member of class TRibbonSilverColorMap.
	DisabledFontColor (see page 6918)	Changes the look and feel of your applications.
	DocumentFontColor (see page 6918)	This is DocumentFontColor, a member of class TRibbonSilverColorMap.
	FontColor (see page 6918)	Changes the look and feel of your applications.
	GroupFontColor (see page 6918)	This is GroupFontColor, a member of class TRibbonSilverColorMap.
	HighlightColor (see page 6919)	Changes the look and feel of your applications.
	InactiveTabFontColor (see page 6919)	This is InactiveTabFontColor, a member of class TRibbonSilverColorMap.
	QuickAccessToolbarMoreBorderColor (see page 6919)	This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonSilverColorMap.
	QuickAccessToolbarMoreColor (see page 6919)	This is QuickAccessToolbarMoreColor, a member of class TRibbonSilverColorMap.

76.1.27.3.1 TRibbonSilverColorMap.ActiveTabFontColor Property

Pascal

```
property ActiveTabFontColor: TColor;
```

C++

```
__property TColor ActiveTabFontColor;
```

Description

This is ActiveTabFontColor, a member of class TRibbonSilverColorMap.

76.1.27.3.2 TRibbonSilverColorMap.BevelMinimizedInner Property

Pascal

```
property BevelMinimizedInner: TColor;
```

C++

```
__property TColor BevelMinimizedInner;
```

Description

This is BevelMinimizedInner, a member of class TRibbonSilverColorMap.

76.1.27.3.3 TRibbonSilverColorMap.BevelMinimizedOuter Property

Pascal

```
property BevelMinimizedOuter: TColor;
```

C++

```
__property TColor BevelMinimizedOuter;
```

Description

This is BevelMinimizedOuter, a member of class TRibbonSilverColorMap.

76.1.27.3.4 TRibbonSilverColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont;
```

C++

```
__property BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.27.3.5 TRibbonSilverColorMap.CaptionFontColor Property

Pascal

```
property CaptionFontColor: TColor;
```

C++

```
__property TColor CaptionFontColor;
```

Description

This is CaptionFontColor, a member of class TRibbonSilverColorMap.

76.1.27.3.6 TRibbonSilverColorMap.ControlColor Property

Pascal

```
property ControlColor: TColor;
```

C++

```
__property TColor ControlColor;
```

Description

This is ControlColor, a member of class TRibbonSilverColorMap.

76.1.27.3.7 TRibbonSilverColorMap.ControlFrameColor Property

Pascal

```
property ControlFrameColor: TColor;
```

C++

```
__property TColor ControlFrameColor;
```

Description

This is ControlFrameColor, a member of class TRibbonSilverColorMap.

76.1.27.3.8 TRibbonSilverColorMap.ControlInactiveColor Property

Pascal

```
property ControlInactiveColor: TColor;
```

C++

```
__property TColor ControlInactiveColor;
```

Description

This is ControlInactiveColor, a member of class TRibbonSilverColorMap.

76.1.27.3.9 TRibbonSilverColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor;
```

C++

```
__property DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.27.3.10 TRibbonSilverColorMap.DocumentFontColor Property

Pascal

```
property DocumentFontColor: TColor;
```

C++

```
__property TColor DocumentFontColor;
```

Description

This is DocumentFontColor, a member of class TRibbonSilverColorMap.

76.1.27.3.11 TRibbonSilverColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor;
```

C++

```
__property FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.27.3.12 TRibbonSilverColorMap.GroupFontColor Property

Pascal

```
property GroupFontColor: TColor;
```

C++

```
__property TColor GroupFontColor;
```

Description

This is GroupFontColor, a member of class TRibbonSilverColorMap.

76.1.27.3.13 TRibbonSilverColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor;
```

C++

```
__property HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

76.1.27.3.14 TRibbonSilverColorMap.InactiveTabFontColor Property

Pascal

```
property InactiveTabFontColor: TColor;
```

C++

```
__property TColor InactiveTabFontColor;
```

Description

This is InactiveTabFontColor, a member of class TRibbonSilverColorMap.

76.1.27.3.15 TRibbonSilverColorMap.QuickAccessToolbarMoreBorderColor Property

Pascal

```
property QuickAccessToolbarMoreBorderColor: TColor;
```

C++

```
__property TColor QuickAccessToolbarMoreBorderColor;
```

Description

This is QuickAccessToolbarMoreBorderColor, a member of class TRibbonSilverColorMap.

76.1.27.3.16 TRibbonSilverColorMap.QuickAccessToolbarMoreColor Property

Pascal

```
property QuickAccessToolbarMoreColor: TColor;
```

C++

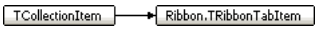
```
__property TColor QuickAccessToolbarMoreColor;
```

Description

This is QuickAccessToolbarMoreColor, a member of class TRibbonSilverColorMap.

76.1.28 TRibbonTabItem Class

Class Hierarchy



Pascal

```
TRibbonTabItem = class(TCollectionItem);
```

C++

```
class TRibbonTabItem : public TCollectionItem;
```

File

Ribbon

Description

This is class Ribbon.TRibbonTabItem.

76.1.28.1 TRibbonTabItem Members

The following tables list the members exposed by TRibbonTabItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 6922)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 6921)	Copies the contents of another, similar object.
	Destroy (see page 6920)	Destroys the TCollectionItem instance and frees its memory.

Public Properties

	Name	Description
	BestTabWidth (see page 6923)	This is BestTabWidth, a member of class TRibbonTabItem.
	KeyTipForm (see page 6923)	This is KeyTipForm, a member of class TRibbonTabItem.
	MinTabwidth (see page 6923)	This is MinTabwidth, a member of class TRibbonTabItem.
	MinTruncatedWidth (see page 6924)	This is MinTruncatedWidth, a member of class TRibbonTabItem.
	Modified (see page 6924)	This is Modified, a member of class TRibbonTabItem.
	X (see page 6924)	This is X, a member of class TRibbonTabItem.
	Y (see page 6925)	This is Y, a member of class TRibbonTabItem.

Published Properties

	Name	Description
	Caption (see page 6923)	This is Caption, a member of class TRibbonTabItem.
	KeyTip (see page 6923)	This is KeyTip, a member of class TRibbonTabItem.
	Page (see page 6924)	This is Page, a member of class TRibbonTabItem.

76.1.28.2 TRibbonTabItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonTabItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free

76.1.28.3 TRibbonTabItem Methods

The methods of the TRibbonTabItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 6922)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 6921)	Copies the contents of another, similar object.

76.1.28.3.1 TRibbonTabItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

76.1.28.3.2 TRibbonTabItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

76.1.28.4 TRibbonTabItem Properties

The properties of the TRibbonTabItem class are listed here.

Public Properties

	Name	Description
R	BestTabWidth (see page 6923)	This is BestTabWidth, a member of class TRibbonTabItem.
	KeyTipForm (see page 6923)	This is KeyTipForm, a member of class TRibbonTabItem.
R	MinTabwidth (see page 6923)	This is MinTabwidth, a member of class TRibbonTabItem.
R	MinTruncatedWidth (see page 6924)	This is MinTruncatedWidth, a member of class TRibbonTabItem.
	Modified (see page 6924)	This is Modified, a member of class TRibbonTabItem.
	X (see page 6924)	This is X, a member of class TRibbonTabItem.
	Y (see page 6925)	This is Y, a member of class TRibbonTabItem.

Published Properties

	Name	Description
	Caption (see page 6923)	This is Caption, a member of class TRibbonTabItem.
	KeyTip (see page 6923)	This is KeyTip, a member of class TRibbonTabItem.
	Page (see page 6924)	This is Page, a member of class TRibbonTabItem.

76.1.28.4.1 TRibbonTabItem.BestTabWidth Property

Pascal

```
property BestTabWidth: Integer;
```

C++

```
__property int BestTabWidth;
```

Description

This is BestTabWidth, a member of class TRibbonTabItem.

76.1.28.4.2 TRibbonTabItem.Caption Property

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

This is Caption, a member of class TRibbonTabItem.

76.1.28.4.3 TRibbonTabItem.KeyTip Property

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

This is KeyTip, a member of class TRibbonTabItem.

76.1.28.4.4 TRibbonTabItem.KeyTipForm Property

Pascal

```
property KeyTipForm: TCustomControl;
```

C++

```
__property TCustomControl KeyTipForm;
```

Description

This is KeyTipForm, a member of class TRibbonTabItem.

76.1.28.4.5 TRibbonTabItem.MinTabwidth Property

Pascal

```
property MinTabwidth: Integer;
```

C++

```
__property int MinTabwidth;
```

Description

This is MinTabwidth, a member of class TRibbonTabItem.

76.1.28.4.6 TRibbonTabItem.MinTruncatedWidth Property

Pascal

```
property MinTruncatedWidth: Integer;
```

C++

```
__property int MinTruncatedWidth;
```

Description

This is MinTruncatedWidth, a member of class TRibbonTabItem.

76.1.28.4.7 TRibbonTabItem.Modified Property

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

This is Modified, a member of class TRibbonTabItem.

76.1.28.4.8 TRibbonTabItem.Page Property

Pascal

```
property Page: TCustomRibbonPage;
```

C++

```
__property TCustomRibbonPage Page;
```

Description

This is Page, a member of class TRibbonTabItem.

76.1.28.4.9 TRibbonTabItem.X Property

Pascal

```
property X: Integer;
```

C++

```
__property int X;
```

Description

This is X, a member of class TRibbonTabItem.

76.1.28.4.10 TRibbonTabItem.Y Property

Pascal

```
property Y: Integer;
```

C++

```
__property int Y;
```

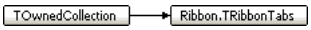
Description

This is Y, a member of class TRibbonTabItem.

76.1.29 TRibbonTabs Class

Collection of information about ribbon pages.

Class Hierarchy



Pascal

```
TRibbonTabs = class(TOwnedCollection);
```

C++

```
class TRibbonTabs : public TOwnedCollection;
```

File

Ribbon

Description

TRibbonTabs is a collection that maintains information about ribbon pages, their captions and keytip values.

The ribbon control can support many user selectable tabs. The TRibbonTabs object stores all tabs and tab specific values associated with the tab. TRibbonTabs also performs the calculations needed to determine if the tabs can fit in the user displayed area and, if not, performs the resizing of the tabs.

76.1.29.1 TRibbonTabs Members

The following tables list the members exposed by TRibbonTabs.

Protected Methods

	Name	Description
	Notify (see page 6927)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 6926)	Creates a new TCollectionItem instance and adds it to the Items array.
	Create (see page 6926)	Creates and initializes a TOwnedCollection instance.
	FindGroup (see page 6927)	This is FindGroup, a member of class TRibbonTabs.
	GetPageFromPageCaption (see page 6927)	This is GetPageFromPageCaption, a member of class TRibbonTabs.
	IndexOfCaption (see page 6927)	This is IndexOfCaption, a member of class TRibbonTabs.
	SelectNext (see page 6928)	Select next page in ribbon.

	SelectPrior (see page 6928)	Select prior page in ribbon.
	TabExists (see page 6929)	This is TabExists, a member of class TRibbonTabs.

Public Properties

	Name	Description
	DesignerAdd (see page 6929)	This is DesignerAdd, a member of class TRibbonTabs.
	Items (see page 6929)	Lists the items in the collection.
R	Ribbon (see page 6930)	This is Ribbon, a member of class TRibbonTabs.

76.1.29.2 TRibbonTabs.Create Constructor

Creates and initializes a TOwnedCollection instance.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TRibbonTabs(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes two parameters: the name of an instance object descended from TPersistent and the name of a TCollectionItem descendant class. The first parameter is the owner of the TOwnedCollection instance. The second parameter determines the class of the items created by the Add method.

See Also

Add

76.1.29.3 TRibbonTabs Methods

The methods of the TRibbonTabs class are listed here.

Protected Methods

	Name	Description
V	Notify (see page 6927)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 6926)	Creates a new TCollectionItem instance and adds it to the Items array.
	FindGroup (see page 6927)	This is FindGroup, a member of class TRibbonTabs.
	GetPageFromPageCaption (see page 6927)	This is GetPageFromPageCaption, a member of class TRibbonTabs.
	IndexOfCaption (see page 6927)	This is IndexOfCaption, a member of class TRibbonTabs.
	SelectNext (see page 6928)	Select next page in ribbon.
	SelectPrior (see page 6928)	Select prior page in ribbon.
	TabExists (see page 6929)	This is TabExists, a member of class TRibbonTabs.

76.1.29.3.1 TRibbonTabs.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TRibbonTabItem;
```

C++

```
__fastcall TRibbonTabItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items

Insert

Create

Delete

76.1.29.3.2 TRibbonTabs.FindGroup Method

Pascal

```
function FindGroup(const GroupName: string): TControl;
```

C++

```
__fastcall TControl * FindGroup(const UnicodeString GroupName);
```

Description

This is FindGroup, a member of class TRibbonTabs.

76.1.29.3.3 TRibbonTabs.GetPageFromPageCaption Method

Pascal

```
function GetPageFromPageCaption(const Caption: string): TCustomRibbonPage;
```

C++

```
__fastcall TCustomRibbonPage GetPageFromPageCaption(const UnicodeString Caption);
```

Description

This is GetPageFromPageCaption, a member of class TRibbonTabs.

76.1.29.3.4 TRibbonTabs.IndexOfCaption Method

Pascal

```
function IndexOfCaption(const Caption: string): Integer;
```

C++

```
__fastcall int IndexOfCaption(const UnicodeString Caption);
```

Description

This is IndexOfCaption, a member of class TRibbonTabs.

76.1.29.3.5 TRibbonTabs.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added

Deleting

76.1.29.3.6 TRibbonTabs.SelectNext Method

Select next page in ribbon.

Pascal

```
function SelectNext: Boolean;
```

C++

```
__fastcall Boolean SelectNext();
```

Description

Use SelectNext to select the next page in the ribbon.

See Also

SelectPrior (see page 6928)

76.1.29.3.7 TRibbonTabs.SelectPrior Method

Select prior page in ribbon.

Pascal

```
function SelectPrior: Boolean;
```

C++

```
__fastcall Boolean SelectPrior();
```

Description

Use SelectPrior to select the prior page in the ribbon.

See Also

SelectNext (see page 6928)

76.1.29.3.8 TRibbonTabs.TabExists Method

Pascal

```
function TabExists(const Caption: string): Boolean;
```

C++

```
__fastcall Boolean TabExists(const UnicodeString Caption);
```

Description

This is TabExists, a member of class TRibbonTabs.

76.1.29.4 TRibbonTabs Properties

The properties of the TRibbonTabs class are listed here.

Public Properties

	Name	Description
	DesignerAdd (see page 6929)	This is DesignerAdd, a member of class TRibbonTabs.
	Items (see page 6929)	Lists the items in the collection.
R	Ribbon (see page 6930)	This is Ribbon, a member of class TRibbonTabs.

76.1.29.4.1 TRibbonTabs.DesignerAdd Property

Pascal

```
property DesignerAdd: Boolean;
```

C++

```
__property Boolean DesignerAdd;
```

Description

This is DesignerAdd, a member of class TRibbonTabs.

76.1.29.4.2 TRibbonTabs.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TRibbonTabItem;
```

C++

```
__property TRibbonTabItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count

ItemClass

Index

Add

Insert

76.1.29.4.3 TRibbonTabs.Ribbon Property

Pascal

```
property Ribbon: TCustomRibbon;
```

C++

```
__property TCustomRibbon Ribbon;
```

Description

This is Ribbon, a member of class TRibbonTabs.

76.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IRibbon (see page 6931)	Interface for ribbon control.
	IRibbonGroup (see page 6933)	Interface for any ribbon-specific group.
	IRibbonQuickAccessToolbar (see page 6934)	This is class Ribbon.IRibbonQuickAccessToolbar.

76.2.1 IRibbon Interface

Interface for ribbon control.

Class Hierarchy

Ribbon.IRibbon

Pascal

```
IRibbon = interface;
```

C++

```
__interface IRibbon;
```

File

Ribbon

Description

Implement the IRibbon interface to acknowledge that the control supports the ribbon control.

See Also

Creating an Application that Uses Ribbon Controls

Adding Commands to the Ribbon

IRibbonForm

IRibbonGroup (see page 6933)

76.2.1.1 IRibbon Members

The following tables list the members exposed by IRibbon.

Public Methods

	Name	Description
	GetRibbon (see page 6932)	This is GetRibbon, a member of class IRibbon.
	SetRibbon (see page 6932)	This is SetRibbon, a member of class IRibbon.

Public Properties

	Name	Description
	Ribbon (see page 6932)	This is Ribbon, a member of class IRibbon.

76.2.1.2 IRibbon Methods

The methods of the IRibbon class are listed here.

Public Methods

	Name	Description
	GetRibbon (see page 6932)	This is GetRibbon, a member of class IRibbon.
	SetRibbon (see page 6932)	This is SetRibbon, a member of class IRibbon.

76.2.1.2.1 IRibbon.GetRibbon Method

Pascal

```
function GetRibbon: TCustomRibbon;
```

C++

```
__fastcall TCustomRibbon GetRibbon();
```

Description

This is GetRibbon, a member of class IRibbon.

76.2.1.2.2 IRibbon.SetRibbon Method

Pascal

```
procedure SetRibbon(const Value: TCustomRibbon);
```

C++

```
__fastcall SetRibbon(const TCustomRibbon Value);
```

Description

This is SetRibbon, a member of class IRibbon.

76.2.1.3 IRibbon Properties

The properties of the IRibbon class are listed here.

Public Properties

	Name	Description
	Ribbon (see page 6932)	This is Ribbon, a member of class IRibbon.

76.2.1.3.1 IRibbon.Ribbon Property

Pascal

```
property Ribbon: TCustomRibbon;
```

C++

```
__property TCustomRibbon Ribbon;
```

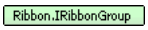
Description

This is Ribbon, a member of class IRibbon.

76.2.2 IRibbonGroup Interface

Interface for any ribbon-specific group.

Class Hierarchy



Pascal

```
IRibbonGroup = interface;
```

C++

```
__interface IRibbonGroup;
```

File

Ribbon

Description

IRibbonGroup is an interface implemented by any ribbon-specific group.

See Also

IRibbon (see page 6931)

IRibbonForm

76.2.2.1 IRibbonGroup Members

The following tables list the members exposed by IRibbonGroup.

Public Methods

	Name	Description
	AlignGroupControls (see page 6934)	Realign controls on group.
	CreateControl (see page 6934)	Recreate specific control.
	StyleChanged (see page 6934)	This is StyleChanged, a member of class IRibbonGroup.

76.2.2.2 IRibbonGroup Methods

The methods of the IRibbonGroup class are listed here.

Public Methods

	Name	Description
	AlignGroupControls (see page 6934)	Realign controls on group.
	CreateControl (see page 6934)	Recreate specific control.
	StyleChanged (see page 6934)	This is StyleChanged, a member of class IRibbonGroup.

76.2.2.2.1 IRibbonGroup.AlignGroupControls Method

Realign controls on group.

Pascal

```
procedure AlignGroupControls;
```

C++

```
__fastcall AlignGroupControls();
```

Description

AlignGroupControls realigns controls on a group.

See Also

CreateControl (see page 6934)

76.2.2.2.2 IRibbonGroup.CreateControl Method

Recreate specific control.

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl;
```

C++

```
__fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

CreateControl recreates a specific control.

See Also

AlignGroupControls (see page 6934)

76.2.2.2.3 IRibbonGroup.StyleChanged Method

Pascal

```
procedure StyleChanged(AStyle: TRibbonStyleActionBars);
```

C++

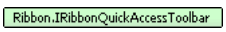
```
__fastcall StyleChanged(TRibbonStyleActionBars AStyle);
```

Description

This is StyleChanged, a member of class IRibbonGroup.

76.2.3 IRibbonQuickAccessToolbar Interface

Class Hierarchy



Pascal

```
IRibbonQuickAccessToolbar = interface;
```

C++

```
__interface IRibbonQuickAccessToolbar;
```

File

Ribbon

Description

This is class Ribbon.IRibbonQuickAccessToolbar.

76.2.3.1 IRibbonQuickAccessToolbar Members

The following tables list the members exposed by IRibbonQuickAccessToolbar.

Public Methods

	Name	Description
	DoUpdateQATWidth (see page 6935)	This is DoUpdateQATWidth, a member of class IRibbonQuickAccessToolbar.

76.2.3.2 IRibbonQuickAccessToolbar Methods

The methods of the IRibbonQuickAccessToolbar class are listed here.

Public Methods

	Name	Description
	DoUpdateQATWidth (see page 6935)	This is DoUpdateQATWidth, a member of class IRibbonQuickAccessToolbar.

76.2.3.2.1 IRibbonQuickAccessToolbar.DoUpdateQATWidth Method

Pascal

```
procedure DoUpdateQATWidth;
```

C++

```
__fastcall DoUpdateQATWidth();
```

Description

This is DoUpdateQATWidth, a member of class IRibbonQuickAccessToolbar.

76.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCommandType (see page 6936)	This is record Ribbon.TCommandType.
	TIconSize (see page 6936)	This is record Ribbon.TIconSize.
	TRibbonGroupVariant (see page 6937)	This is record Ribbon.TRibbonGroupVariant.
	TRibbonMetric (see page 6937)	This is record Ribbon.TRibbonMetric.
	TRibbonScrollButton (see page 6938)	This is record Ribbon.TRibbonScrollButton.
	TTabActiveScrollButton (see page 6938)	This is record Ribbon.TTabActiveScrollButton.
	TTabSeparatorState (see page 6939)	Position of Quick Access Toolbar.

76.3.1 Ribbon.TCommandType Enumeration

Pascal

```
TCommandType = (
  ctRecent,
  ctCommands
);
```

C++

```
enum TCommandType {
  ctRecent,
  ctCommands
};
```

File

Ribbon

Description

This is record Ribbon.TCommandType.

76.3.2 Ribbon.TIconSize Enumeration

Pascal

```
TIconSize = (
  isSmall,
  isMedium,
  isLarge
);
```

C++

```
enum TIconSize {
  isSmall,
  isMedium,
```

```
    isLarge
  };
```

File

Ribbon

Description

This is record Ribbon.TIconSize.

76.3.3 Ribbon.TRibbonGroupVariant Enumeration

Pascal

```
  TRibbonGroupVariant = (
    rgvCollapsed,
    rgvSmall,
    rgvMedium,
    rgvLarge
  );
```

C++

```
  enum TRibbonGroupVariant {
    rgvCollapsed,
    rgvSmall,
    rgvMedium,
    rgvLarge
  };
```

File

Ribbon

Description

This is record Ribbon.TRibbonGroupVariant.

76.3.4 Ribbon.TRibbonMetric Enumeration

Pascal

```
  TRibbonMetric = (
    rmCaption,
    rmHeight,
    rmFullHeight,
    rmMinHeight,
    rmTabTop,
    rmPageHeight,
    rmGroupHeight,
    rmGroupCaptionHeight,
    rmQATWidth,
    rmSystemButtonHeight,
    rmSystemButtonWidth,
    rmTabHeight,
    rmFontSize
  );
```

C++

```
  enum TRibbonMetric {
```

```
    rmCaption,
    rmHeight,
    rmFullHeight,
    rmMinHeight,
    rmTabTop,
    rmPageHeight,
    rmGroupHeight,
    rmGroupCaptionHeight,
    rmQATWidth,
    rmSystemButtonHeight,
    rmSystemButtonWidth,
    rmTabHeight,
    rmFontSize
  };
```

File

Ribbon

Description

This is record Ribbon.TRibbonMetric.

76.3.5 Ribbon.TRibbonScrollButton Enumeration

Pascal

```
  TRibbonScrollButton = (
    tsbTabRight,
    tsbTabLeft,
    tsbGroupLeft,
    tsbGroupRight
  );
```

C++

```
  enum TRibbonScrollButton {
    tsbTabRight,
    tsbTabLeft,
    tsbGroupLeft,
    tsbGroupRight
  };
```

File

Ribbon

Description

This is record Ribbon.TRibbonScrollButton.

76.3.6 Ribbon.TTabActiveScrollButton Enumeration

Pascal

```
  TTabActiveScrollButton = (
    asbNone,
    asbRight,
    asbLeft
  );
```

C++

```
enum TTabActiveScrollButton {
  asbNone,
  asbRight,
  asbLeft
};
```

File

Ribbon

Description

This is record Ribbon.TTabActiveScrollButton.

76.3.7 Ribbon.TTabSeparatorState Enumeration

Position of Quick Access Toolbar.

Pascal

```
TTabSeparatorState = (
  tssHidden,
  tssLight,
  tssMedium,
  tssDark
);
```

C++

```
enum TTabSeparatorState {
  tssHidden,
  tssLight,
  tssMedium,
  tssDark
};
```

File

Ribbon

Description

TTabSeparatorState is the state of the separators that are displayed between tabs when the ribbon is being resized smaller.

Value	Meaning
tssHidden	Hidden
tssLight	Light
tssMedium	Medium
tssDark	Dark

See Also

TApplicationMenu (see page 6716)

76.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCustomRibbonClass (see page 6940)	This is type Ribbon.TCustomRibbonClass.
TRecentItemClickEvent (see page 6940)	This is type Ribbon.TRecentItemClickEvent.
TRibbonScrollButtons (see page 6941)	This is type Ribbon.TRibbonScrollButtons.

76.4.1 Ribbon.TCustomRibbonClass Type

Pascal

```
TCustomRibbonClass = class of TCustomRibbon;
```

C++

```
TCustomRibbon TCustomRibbonClass;
```

File

Ribbon

Description

This is type Ribbon.TCustomRibbonClass.

76.4.2 Ribbon.TRecentItemClickEvent Type

Pascal

```
TRecentItemClickEvent = procedure (Sender: TObject; FileName: string; Index: Integer) of
object;
```

C++

```
(Sender: TObject; FileName: string; Index: Integer) ( TRecentItemClickEvent)();
```

File

Ribbon

Description

This is type Ribbon.TRecentItemClickEvent.

76.4.3 Ribbon.TRibbonScrollButtons Type

Pascal

```
TRibbonScrollButtons = set of TRibbonScrollButton;
```

C++

```
TRibbonScrollButton TRibbonScrollButtons;
```

File

Ribbon

Description

This is type Ribbon.TRibbonScrollButtons.

77 RibbonActnCtrls Namespace

This is namespace RibbonActnCtrls.

Topics

Name	Description
Classes (see page 6946)	The following table lists classes in this documentation.
Interfaces (see page 7272)	The following table lists interfaces in this documentation.
Functions (see page 7284)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7286)	The following table lists structs, records, enums in this documentation.
Types (see page 7291)	The following table lists types in this documentation.
Constants (see page 7292)	The following table lists constants in this documentation.

Classes

	Name	Description
	TCustomRibbonComboBox (see page 6947)	This is class RibbonActnCtrls.TCustomRibbonComboBox.
	TKeyTipList (see page 6956)	**KeyTip** list.
	TKeyTipListEnumerator (see page 6960)	This is class RibbonActnCtrls.TKeyTipListEnumerator.
	TKeyTipMatchItem (see page 6962)	Details about Full and Partial **KeyTip** matches.
	TKeyTipValidator (see page 6964)	Used to validate **KeyTip**.
	TRibbonAddRemoveItem (see page 6965)	This is class RibbonActnCtrls.TRibbonAddRemoveItem.
	TRibbonApplicationButton (see page 6967)	This is class RibbonActnCtrls.TRibbonApplicationButton.
	TRibbonApplicationMenuButton (see page 6968)	Application (see page 2612) Button when using Ribbon (see page 6713) interface.
	TRibbonApplicationMenuItem (see page 6976)	This is class RibbonActnCtrls.TRibbonApplicationMenuItem.
	TRibbonBaseButtonControl (see page 6980)	Common ancestor for many buttons in groups and Quick Access Toolbar.
	TRibbonCheckBoxControl (see page 6990)	This is class RibbonActnCtrls.TRibbonCheckBoxControl.
	TRibbonComboBox (see page 6992)	This is class RibbonActnCtrls.TRibbonComboBox.
	TRibbonComboBoxItem (see page 7045)	This is class RibbonActnCtrls.TRibbonComboBoxItem.
	TRibbonComboControl (see page 7047)	This is class RibbonActnCtrls.TRibbonComboControl.
	TRibbonCustomControl (see page 7061)	This is class RibbonActnCtrls.TRibbonCustomControl.
	TRibbonCustomizeItem (see page 7073)	This is class RibbonActnCtrls.TRibbonCustomizeItem.
	TRibbonCustomStatusControl (see page 7076)	This is class RibbonActnCtrls.TRibbonCustomStatusControl.
	TRibbonDropDownButton (see page 7078)	This is class RibbonActnCtrls.TRibbonDropDownButton.
	TRibbonGallery (see page 7083)	This is class RibbonActnCtrls.TRibbonGallery.
	TRibbonGalleryControl (see page 7083)	This is class RibbonActnCtrls.TRibbonGalleryControl.
	TRibbonGalleryControlMenuItem (see page 7085)	This is class RibbonActnCtrls.TRibbonGalleryControlMenuItem.

	TRibbonGalleryItem (see page 7091)	This is class RibbonActnCtrls.TRibbonGalleryItem.
	TRibbonGalleryMenuItem (see page 7093)	This is class RibbonActnCtrls.TRibbonGalleryMenuItem.
	TRibbonGallerySplitButton (see page 7096)	This is class RibbonActnCtrls.TRibbonGallerySplitButton.
	TRibbonGripItem (see page 7098)	This is class RibbonActnCtrls.TRibbonGripItem.
	TRibbonGroupDropDown (see page 7101)	Display entire group as dropdown menu in Quick Access Toolbar.
	TRibbonLargeButtonControl (see page 7105)	This is class RibbonActnCtrls.TRibbonLargeButtonControl.
	TRibbonLargeSplitButton (see page 7108)	This is class RibbonActnCtrls.TRibbonLargeSplitButton.
	TRibbonMainMenuItem (see page 7115)	This is class RibbonActnCtrls.TRibbonMainMenuItem.
	TRibbonMainMenuSeparator (see page 7117)	This is class RibbonActnCtrls.TRibbonMainMenuSeparator.
	TRibbonMenuItem (see page 7118)	This is class RibbonActnCtrls.TRibbonMenuItem.
	TRibbonMenuSeparatorItem (see page 7123)	This is class RibbonActnCtrls.TRibbonMenuSeparatorItem.
	TRibbonMenuSmallButtonControl (see page 7125)	This is class RibbonActnCtrls.TRibbonMenuSmallButtonControl.
	TRibbonOptionButtonControl (see page 7128)	This is class RibbonActnCtrls.TRibbonOptionButtonControl.
	TRibbonQATScrollButton (see page 7130)	This is class RibbonActnCtrls.TRibbonQATScrollButton.
	TRibbonRadioButtonControl (see page 7134)	This is class RibbonActnCtrls.TRibbonRadioButtonControl.
	TRibbonRecentButtonControl (see page 7137)	This is class RibbonActnCtrls.TRibbonRecentButtonControl.
	TRibbonRichContentItem (see page 7139)	This is class RibbonActnCtrls.TRibbonRichContentItem.
	TRibbonRichContentMainMenuItem (see page 7143)	This is class RibbonActnCtrls.TRibbonRichContentMainMenuItem.
	TRibbonSeparator (see page 7147)	This is class RibbonActnCtrls.TRibbonSeparator.
	TRibbonSmallButtonControl (see page 7150)	This is class RibbonActnCtrls.TRibbonSmallButtonControl.
	TRibbonSmallSplitButton (see page 7152)	This is class RibbonActnCtrls.TRibbonSmallSplitButton.
	TRibbonSpinButton (see page 7159)	This is class RibbonActnCtrls.TRibbonSpinButton.
	TRibbonSpinEdit (see page 7198)	This is class RibbonActnCtrls.TRibbonSpinEdit.
	TRibbonSplitMenuItem (see page 7257)	This is class RibbonActnCtrls.TRibbonSplitMenuItem.
	TRibbonStyleMenuButton (see page 7262)	Top level menus when using Ribbon (see page 6713) Styles in applications not using Ribbon (see page 6713) interface.
	TRibbonTextControl (see page 7263)	This is class RibbonActnCtrls.TRibbonTextControl.
	TRibbonToolScrollButton (see page 7265)	This is class RibbonActnCtrls.TRibbonToolScrollButton.
	TTimerSpeedButton (see page 7267)	This is class RibbonActnCtrls.TTimerSpeedButton.

Constants

Name	Description
UM_FADECONTROL (see page 7292)	This is constant RibbonActnCtrls.UM_FADECONTROL.

Functions

	Name	Description
	DrawLine (see page 7284)	This is function RibbonActnCtrls.DrawLine.
	IsLeftMouseButtonPressed (see page 7284)	This is function RibbonActnCtrls.IsLeftMouseButtonPressed.
	ShowKeyTip (see page 7284)	Show **KeyTip** for control.
	ShowKeyTip (see page 7284)	Show **KeyTip** for control.

Interfaces

	Name	Description
	IRibbonBarKeyTip (see page 7272)	Action Bars that support KeyTips should implement this interface.
	IRibbonContainedControl (see page 7274)	Interface implemented by controls used as containers for displaying other controls.
	IRibbonControl (see page 7276)	Interface implemented by controls visible in Ribbon (see page 6713) Groups.
	IRibbonGroupDropdown (see page 7277)	This is class RibbonActnCtrls.IRibbonGroupDropdown.
	IRibbonKeyTip (see page 7278)	Commands with a **KeyTip (see page 7282)** should implement this interface.
	IRibbonSplitButton (see page 7282)	Interface implemented by controls that have split button.

Structs, Records, Enums

	Name	Description
	TKeyTipHandled (see page 7286)	Validate if command's **KeyTip** matches current user input.
	TKeyTipMatchType (see page 7287)	Type of match command's **KeyTip** is for current **KeyTip** value.

	TKeyTipPosition (see page 7287)	Position of KeyTips.
	TKeyTipState (see page 7288)	Current state of KeyTips.
	TQATScrollMode (see page 7289)	This is record RibbonActnCtrls.TQATScrollMode.
	TSpinButtonStyle (see page 7290)	This is record RibbonActnCtrls.TSpinButtonStyle.

Types

Name	Description
TNumGlyphs (see page 7291)	This is type RibbonActnCtrls.TNumGlyphs.
TTimeBtnState (see page 7291)	This is type RibbonActnCtrls.TTimeBtnState.

77.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomRibbonComboBox (see page 6947)	This is class RibbonActnCtrls.TCustomRibbonComboBox.
	TKeyTipList (see page 6956)	**KeyTip** list.
	TKeyTipListEnumerator (see page 6960)	This is class RibbonActnCtrls.TKeyTipListEnumerator.
	TKeyTipMatchItem (see page 6962)	Details about Full and Partial **KeyTip** matches.
	TKeyTipValidator (see page 6964)	Used to validate **KeyTip**.
	TRibbonAddRemoveItem (see page 6965)	This is class RibbonActnCtrls.TRibbonAddRemoveItem.
	TRibbonApplicationButton (see page 6967)	This is class RibbonActnCtrls.TRibbonApplicationButton.
	TRibbonApplicationMenuButton (see page 6968)	Application (see page 2612) Button when using Ribbon (see page 6713) interface.
	TRibbonApplicationMenuItem (see page 6976)	This is class RibbonActnCtrls.TRibbonApplicationMenuItem.
	TRibbonBaseButtonControl (see page 6980)	Common ancestor for many buttons in groups and Quick Access Toolbar.
	TRibbonCheckBoxControl (see page 6990)	This is class RibbonActnCtrls.TRibbonCheckBoxControl.
	TRibbonComboBox (see page 6992)	This is class RibbonActnCtrls.TRibbonComboBox.
	TRibbonComboBoxItem (see page 7045)	This is class RibbonActnCtrls.TRibbonComboBoxItem.
	TRibbonComboControl (see page 7047)	This is class RibbonActnCtrls.TRibbonComboControl.
	TRibbonCustomControl (see page 7061)	This is class RibbonActnCtrls.TRibbonCustomControl.
	TRibbonCustomizeItem (see page 7073)	This is class RibbonActnCtrls.TRibbonCustomizeItem.
	TRibbonCustomStatusControl (see page 7076)	This is class RibbonActnCtrls.TRibbonCustomStatusControl.
	TRibbonDropDownButton (see page 7078)	This is class RibbonActnCtrls.TRibbonDropDownButton.
	TRibbonGallery (see page 7083)	This is class RibbonActnCtrls.TRibbonGallery.
	TRibbonGalleryControl (see page 7083)	This is class RibbonActnCtrls.TRibbonGalleryControl.
	TRibbonGalleryControlMenuItem (see page 7085)	This is class RibbonActnCtrls.TRibbonGalleryControlMenuItem.
	TRibbonGalleryItem (see page 7091)	This is class RibbonActnCtrls.TRibbonGalleryItem.
	TRibbonGalleryMenuItem (see page 7093)	This is class RibbonActnCtrls.TRibbonGalleryMenuItem.
	TRibbonGallerySplitButton (see page 7096)	This is class RibbonActnCtrls.TRibbonGallerySplitButton.
	TRibbonGripItem (see page 7098)	This is class RibbonActnCtrls.TRibbonGripItem.
	TRibbonGroupDropDown (see page 7101)	Display entire group as dropdown menu in Quick Access Toolbar.
	TRibbonLargeButtonControl (see page 7105)	This is class RibbonActnCtrls.TRibbonLargeButtonControl.
	TRibbonLargeSplitButton (see page 7108)	This is class RibbonActnCtrls.TRibbonLargeSplitButton.
	TRibbonMainMenuItem (see page 7115)	This is class RibbonActnCtrls.TRibbonMainMenuItem.
	TRibbonMainMenuSeparator (see page 7117)	This is class RibbonActnCtrls.TRibbonMainMenuSeparator.
	TRibbonMenuItem (see page 7118)	This is class RibbonActnCtrls.TRibbonMenuItem.
	TRibbonMenuSeparatorItem (see page 7123)	This is class RibbonActnCtrls.TRibbonMenuSeparatorItem.
	TRibbonMenuSmallButtonControl (see page 7125)	This is class RibbonActnCtrls.TRibbonMenuSmallButtonControl.
	TRibbonOptionButtonControl (see page 7128)	This is class RibbonActnCtrls.TRibbonOptionButtonControl.
	TRibbonQATScrollButton (see page 7130)	This is class RibbonActnCtrls.TRibbonQATScrollButton.
	TRibbonRadioButtonControl (see page 7134)	This is class RibbonActnCtrls.TRibbonRadioButtonControl.
	TRibbonRecentButtonControl (see page 7137)	This is class RibbonActnCtrls.TRibbonRecentButtonControl.
	TRibbonRichContentItem (see page 7139)	This is class RibbonActnCtrls.TRibbonRichContentItem.
	TRibbonRichContentMainMenuItem (see page 7143)	This is class RibbonActnCtrls.TRibbonRichContentMainMenuItem.
	TRibbonSeparator (see page 7147)	This is class RibbonActnCtrls.TRibbonSeparator.
	TRibbonSmallButtonControl (see page 7150)	This is class RibbonActnCtrls.TRibbonSmallButtonControl.
	TRibbonSmallSplitButton (see page 7152)	This is class RibbonActnCtrls.TRibbonSmallSplitButton.
	TRibbonSpinButton (see page 7159)	This is class RibbonActnCtrls.TRibbonSpinButton.
	TRibbonSpinEdit (see page 7198)	This is class RibbonActnCtrls.TRibbonSpinEdit.

	TRibbonSplitMenuItem (see page 7257)	This is class RibbonActnCtrls.TRibbonSplitMenuItem.
	TRibbonStyleMenuButton (see page 7262)	Top level menus when using Ribbon (see page 6713) Styles in applications not using Ribbon (see page 6713) interface.
	TRibbonTextControl (see page 7263)	This is class RibbonActnCtrls.TRibbonTextControl.
	TRibbonToolScrollButton (see page 7265)	This is class RibbonActnCtrls.TRibbonToolScrollButton.
	TTimerSpeedButton (see page 7267)	This is class RibbonActnCtrls.TTimerSpeedButton.

77.1.1 TCustomRibbonComboBox Class

Class Hierarchy

Pascal

```
TCustomRibbonComboBox = class(TCustomEdit);
```

C++

```
class TCustomRibbonComboBox : public TCustomEdit;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TCustomRibbonComboBox.

77.1.1.1 TCustomRibbonComboBox Members

The following tables list the members exposed by TCustomRibbonComboBox.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 6949)	Begin Automatic Drag.
	CMMouseenter (see page 6949)	This is CMMouseenter, a member of class TCustomRibbonComboBox.
	CMMouseleave (see page 6950)	This is CMMouseleave, a member of class TCustomRibbonComboBox.
	DragOver (see page 6950)	OnDragOver event dispatcher.
	KeyDown (see page 6951)	Respond to key press events.
	KeyPress (see page 6951)	Respond to keyboard input.
	MouseMove (see page 6952)	Respond to mouse moving over control area..
	Notification (see page 6952)	Responds to notifications that components are being created or destroyed.
	SetParent (see page 6953)	Sets the parent of the control.
	WMKillFocus (see page 6954)	This is WMKillFocus, a member of class TCustomRibbonComboBox.
	WMSetFocus (see page 6954)	This is WMSetFocus, a member of class TCustomRibbonComboBox.

Public Methods

	Name	Description
	Create (see page 6948)	Creates an instance of TCustomEdit.
	Destroy (see page 6948)	Destroys an instance of TWinControl.
	DragDrop (see page 6950)	OnDragDrop event dispatcher.
	SetBounds (see page 6953)	Sets the windowed control's boundary properties all at once.

Public Properties

	Name	Description
	Color (see page 6955)	Specifies the background color of the control.
	Items (see page 6956)	This is Items, a member of class TCustomRibbonComboBox.
	MouseInControl (see page 6956)	This is MouseInControl, a member of class TCustomRibbonComboBox.

77.1.1.2 TCustomRibbonComboBox.Create Constructor

Creates an instance of TCustomEdit.

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomEdit.

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner

77.1.1.3 TCustomRibbonComboBox.Destroy Destructor

Destroys an instance of TWinControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRibbonComboBox();
```

Description

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

See Also

System::TObject::Free

Destroy

77.1.1.4 TCustomRibbonComboBox Methods

The methods of the TCustomRibbonComboBox class are listed here.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 6949)	Begin Automatic Drag.
	CMMouseenter (see page 6949)	This is CMMouseenter, a member of class TCustomRibbonComboBox.
	CMMouseleave (see page 6950)	This is CMMouseleave, a member of class TCustomRibbonComboBox.
	DragOver (see page 6950)	OnDragOver event dispatcher.
	KeyDown (see page 6951)	Respond to key press events.
	KeyPress (see page 6951)	Respond to keyboard input.
	MouseMove (see page 6952)	Respond to mouse moving over control area..
	Notification (see page 6952)	Responds to notifications that components are being created or destroyed.
	SetParent (see page 6953)	Sets the parent of the control.
	WMKillFocus (see page 6954)	This is WMKillFocus, a member of class TCustomRibbonComboBox.
	WMSetFocus (see page 6954)	This is WMSetFocus, a member of class TCustomRibbonComboBox.

Public Methods

	Name	Description
	DragDrop (see page 6950)	OnDragDrop event dispatcher.
	SetBounds (see page 6953)	Sets the windowed control's boundary properties all at once.

77.1.1.4.1 TCustomRibbonComboBox.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl, BeginAutoDrag calls BeginDrag, passing in the global Mouse variable's DragImmediate and DragThreshold properties.

See Also

DragKind

DragMode

EndDrag

Mouse

DragImmediate

DragThreshold

Mouse

77.1.1.4.2 TCustomRibbonComboBox.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TCustomRibbonComboBox.

77.1.1.4.3 TCustomRibbonComboBox.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TCustomRibbonComboBox.

77.1.1.4.4 TCustomRibbonComboBox.DragDrop Method

OnDragDrop event dispatcher.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

When creating a component that descends from TControl, override DragDrop to add additional code that executes before the OnDragDrop event handler is called.

The Source parameter is the object that was dropped onto the control. The X and Y parameters are the mouse coordinates where the object was dropped.

See Also

DragCanceled

OnDragDrop

77.1.1.4.5 TCustomRibbonComboBox.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to

false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop

OnDragOver

77.1.1.4.6 TCustomRibbonComboBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

77.1.1.4.7 TCustomRibbonComboBox.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown

KeyUp

OnKeyPress

77.1.1.4.8 TCustomRibbonComboBox.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

77.1.1.4.9 TCustomRibbonComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

77.1.1.4.10 TCustomRibbonComboBox.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

77.1.1.4.11 TCustomRibbonComboBox.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.1.4.12 TCustomRibbonComboBox.WMKillFocus Method

Pascal

```
procedure WMKillFocus(var Message: TWMKillFocus); message WM_KILLFOCUS;
```

C++

```
__fastcall WMKillFocus(TWMKillFocus Message);
```

Description

This is WMKillFocus, a member of class TCustomRibbonComboBox.

77.1.1.4.13 TCustomRibbonComboBox.WMSetFocus Method

Pascal

```
procedure WMSetFocus(var Message: TWMSetFocus); message WM_SETFOCUS;
```

C++

```
__fastcall WMSetFocus(TWMSetFocus Message);
```

Description

This is WMSetFocus, a member of class TCustomRibbonComboBox.

77.1.1.5 TCustomRibbonComboBox Properties

The properties of the TCustomRibbonComboBox class are listed here.

Public Properties

	Name	Description
	Color (see page 6955)	Specifies the background color of the control.
	Items (see page 6956)	This is Items, a member of class TCustomRibbonComboBox.
	MouseInControl (see page 6956)	This is MouseInControl, a member of class TCustomRibbonComboBox.

77.1.1.5.1 TCustomRibbonComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

77.1.1.5.2 TCustomRibbonComboBox.Items Property

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

This is Items, a member of class TCustomRibbonComboBox.

77.1.1.5.3 TCustomRibbonComboBox.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TCustomRibbonComboBox.

77.1.2 TKeyTipList Class

KeyTip list.

Class Hierarchy

RibbonActnCtrls.TKeyTipList

Pascal

```
TKeyTipList = class;
```

C++

```
class TKeyTipList;
```

File

RibbonActnCtrls

Description

TKeyTipList is a **KeyTip** list.

77.1.2.1 TKeyTipList Members

The following tables list the members exposed by TKeyTipList.

Public Methods

	Name	Description
	Add (see page 6958)	Add new match to list of KeyTips.
	Clear (see page 6958)	Clear list of KeyTips.

	Create (see page 6957)	Create TKeyTipList (see page 6956) instance.
V	Destroy (see page 6957)	Destroy TKeyTipList (see page 6956) instance.

Public Properties

	Name	Description
R	Count (see page 6958)	Number of valid **KeyTip**s.
	Current (see page 6959)	Current **KeyTip** value so far.
R	Full (see page 6959)	Indicates **KeyTip** has full match in list and no other matches
R	Items (see page 6959)	Indexable list of TKeyTipList (see page 6956).
R	Partial (see page 6960)	**KeyTip** has partial match in list.

77.1.2.2 TKeyTipList.Create Constructor

Create TKeyTipList (see page 6956) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall TKeyTipList();
```

Description

This method creates a TKeyTipList (see page 6956) instance.

See Also

Destroy (see page 6957)

77.1.2.3 TKeyTipList.Destroy Destructor

Destroy TKeyTipList (see page 6956) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TKeyTipList();
```

Description

This method destroys a TKeyTipList (see page 6956) instance.

See Also

Create (see page 6957)

77.1.2.4 TKeyTipList Methods

The methods of the TKeyTipList class are listed here.

Public Methods

	Name	Description
	Add (see page 6958)	Add new match to list of KeyTips.
	Clear (see page 6958)	Clear list of KeyTips.

77.1.2.4.1 TKeyTipList.Add Method

Add new match to list of KeyTips.

Pascal

```
procedure Add(MatchType: TKeyTipMatchType; Handled: TKeyTipHandled; Control: TControl; Index:
Integer = -1);
```

C++

```
__fastcall Add(TKeyTipMatchType MatchType, TKeyTipHandled Handled, TControl * Control, int
Index = -1);
```

Description

This method adds a new match to the list of KeyTips.

See Also

Clear (see page 6958)

77.1.2.4.2 TKeyTipList.Clear Method

Clear list of KeyTips.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This method clears the list of KeyTips.

See Also

Add (see page 6958)

77.1.2.5 TKeyTipList Properties

The properties of the TKeyTipList class are listed here.

Public Properties

	Name	Description
R	Count (see page 6958)	Number of valid **KeyTip**s.
	Current (see page 6959)	Current **KeyTip** value so far.
R	Full (see page 6959)	Indicates **KeyTip** has full match in list and no other matches
R	Items (see page 6959)	Indexable list of TKeyTipList (see page 6956).
R	Partial (see page 6960)	**KeyTip** has partial match in list.

77.1.2.5.1 TKeyTipList.Count Property

Number of valid **KeyTip**s.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count contains the number of valid **KeyTip**s.

See Also

Current (see page 6959)

77.1.2.5.2 TKeyTipList.Current Property

Current **KeyTip** value so far.

Pascal

```
property Current: string;
```

C++

```
__property UnicodeString Current;
```

Description

Current is the current **KeyTip** value the user has typed in so far.

See Also

Count (see page 6958)

77.1.2.5.3 TKeyTipList.Full Property

Indicates **KeyTip** has full match in list and no other matches

Pascal

```
property Full: Boolean;
```

C++

```
__property Boolean Full;
```

Description

Full is True if the current **KeyTip** has a full match in the list and there are no other matches, and it is False otherwise.

See Also

Partial (see page 6960)

77.1.2.5.4 TKeyTipList.Items Property

Indexable list of TKeyTipList (see page 6956).

Pascal

```
property Items [Index: Integer]: TKeyTipMatchItem;
```

C++

```
__property TKeyTipMatchItem Items[int Index];
```

Description

Items is an indexable list of TKeyTipList (see page 6956).

77.1.2.5.5 TKeyTipList.Partial Property

KeyTip has partial match in list.

Pascal

```
property Partial: Boolean;
```

C++

```
__property Boolean Partial;
```

Description

Partial is True if the current **KeyTip** has a partial match in the list and false otherwise.

See Also

Full (see page 6959)

77.1.3 TKeyTipListEnumerator Class

Class Hierarchy

Pascal

```
TKeyTipListEnumerator = class;
```

C++

```
class TKeyTipListEnumerator;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TKeyTipListEnumerator.

77.1.3.1 TKeyTipListEnumerator Members

The following tables list the members exposed by TKeyTipListEnumerator.

Public Methods

	Name	Description
	Create (see page 6961)	This is Create, a member of class TKeyTipListEnumerator.
	GetCurrent (see page 6961)	This is GetCurrent, a member of class TKeyTipListEnumerator.
	MoveNext (see page 6961)	This is MoveNext, a member of class TKeyTipListEnumerator.

Public Properties

	Name	Description
	Current (see page 6962)	This is Current, a member of class TKeyTipListEnumerator.

77.1.3.2 TKeyTipListEnumerator.Create Constructor

Pascal

```
constructor Create(AList: TKeyTipList);
```

C++

```
__fastcall TKeyTipListEnumerator(TKeyTipList AList);
```

Description

This is Create, a member of class TKeyTipListEnumerator.

77.1.3.3 TKeyTipListEnumerator Methods

The methods of the TKeyTipListEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 6961)	This is GetCurrent, a member of class TKeyTipListEnumerator.
	MoveNext (see page 6961)	This is MoveNext, a member of class TKeyTipListEnumerator.

77.1.3.3.1 TKeyTipListEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TKeyTipMatchItem;
```

C++

```
__fastcall TKeyTipMatchItem GetCurrent();
```

Description

This is GetCurrent, a member of class TKeyTipListEnumerator.

77.1.3.3.2 TKeyTipListEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TKeyTipListEnumerator.

77.1.3.4 TKeyTipListEnumerator Properties

The properties of the TKeyTipListEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 6962)	This is Current, a member of class TKeyTipListEnumerator.

77.1.3.4.1 TKeyTipListEnumerator.Current Property

Pascal

```
property Current: TKeyTipMatchItem;
```

C++

```
__property TKeyTipMatchItem Current;
```

Description

This is Current, a member of class TKeyTipListEnumerator.

77.1.4 TKeyTipMatchItem Class

Details about Full and Partial **KeyTip** matches.

Class Hierarchy

Pascal

```
TKeyTipMatchItem = class;
```

C++

```
class TKeyTipMatchItem;
```

File

RibbonActnCtrls

Description

TKeyTipMatchItem stores details about Full and Partial **KeyTip** matches. It contains a reference to the control that needs to be executed. If the MatchType (see page 6964) is **mtTabs**, then the Index (see page 6964) property is used to determine which Tab to change to.

See Also

Index (see page 6964)

MatchType (see page 6964)

TKeyTipMatchType (see page 7287)

77.1.4.1 TKeyTipMatchItem Members

The following tables list the members exposed by TKeyTipMatchItem.

Public Methods

	Name	Description
	Create (see page 6963)	This is Create, a member of class TKeyTipMatchItem.

Public Properties

	Name	Description
	Control (see page 6963)	This is Control, a member of class TKeyTipMatchItem.

	Handled (see page 6963)	This is Handled, a member of class TKeyTipMatchItem.
	Index (see page 6964)	Determines which Tab to change to.
	MatchType (see page 6964)	**KeyTip** match type.

77.1.4.2 TKeyTipMatchItem.Create Constructor

Pascal

```
constructor Create(AType: TKeyTipMatchType; AHandled: TKeyTipHandled; AControl: TControl;
AIndex: Integer = -1);
```

C++

```
__fastcall TKeyTipMatchItem(TKeyTipMatchType AType, TKeyTipHandled AHandled, TControl *
AControl, int AIndex = -1);
```

Description

This is Create, a member of class TKeyTipMatchItem.

77.1.4.3 TKeyTipMatchItem Properties

The properties of the TKeyTipMatchItem class are listed here.

Public Properties

	Name	Description
	Control (see page 6963)	This is Control, a member of class TKeyTipMatchItem.
	Handled (see page 6963)	This is Handled, a member of class TKeyTipMatchItem.
	Index (see page 6964)	Determines which Tab to change to.
	MatchType (see page 6964)	**KeyTip** match type.

77.1.4.3.1 TKeyTipMatchItem.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TKeyTipMatchItem.

77.1.4.3.2 TKeyTipMatchItem.Handled Property

Pascal

```
property Handled: TKeyTipHandled;
```

C++

```
__property TKeyTipHandled Handled;
```

Description

This is Handled, a member of class TKeyTipMatchItem.

77.1.4.3.3 TKeyTipMatchItem.Index Property

Determines which Tab to change to.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

If the MatchType (see page 6964) is **mtTabs**, then the Index property is used to determine which Tab to change to.

See Also

MatchType (see page 6964)

TKeyTipMatchType (see page 7287)

77.1.4.3.4 TKeyTipMatchItem.MatchType Property

KeyTip match type.

Pascal

```
property MatchType: TKeyTipMatchType;
```

C++

```
__property TKeyTipMatchType MatchType;
```

Description

MatchType is the **KeyTip** match type.

See Also

Index (see page 6964)

TKeyTipMatchType (see page 7287)

77.1.5 TKeyTipValidator Class

Used to validate **KeyTip**.

Class Hierarchy

Pascal

```
TKeyTipValidator = class;
```

C++

```
class TKeyTipValidator;
```

File

RibbonActnCtrls

Description

TKeyTipValidator is a static class used to validate a **KeyTip**.

77.1.5.1 TKeyTipValidator Members

The following tables list the members exposed by TKeyTipValidator.

Public Methods

	Name	Description
	Validate (see page 6965)	This is Validate, a member of class TKeyTipValidator.

77.1.5.2 TKeyTipValidator Methods

The methods of the TKeyTipValidator class are listed here.

Public Methods

	Name	Description
	Validate (see page 6965)	This is Validate, a member of class TKeyTipValidator.

77.1.5.2.1 TKeyTipValidator.Validate Method

Pascal

```
class procedure Validate(const SrcKeyTip: string; const CmdKeyTip: string; Control: TControl;
var Handled: TKeyTipHandled);
```

C++

```
__fastcall Validate(const UnicodeString SrcKeyTip, const UnicodeString CmdKeyTip, TControl *
Control, TKeyTipHandled Handled);
```

Description

This is Validate, a member of class TKeyTipValidator.

77.1.6 TRibbonAddRemoveItem Class

Class Hierarchy

Pascal

```
TRibbonAddRemoveItem = class(TXPStyleAddRemoveItem);
```

C++

```
class TRibbonAddRemoveItem : public TXPStyleAddRemoveItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonAddRemoveItem.

77.1.6.1 TRibbonAddRemoveItem Members

The following tables list the members exposed by TRibbonAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 6966)	Draws a background image on the control.
	DrawGlyph (see page 6966)	Draws a symbol on the control.

77.1.6.2 TRibbonAddRemoveItem Methods

The methods of the TRibbonAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 6966)	Draws a background image on the control.
	DrawGlyph (see page 6966)	Draws a symbol on the control.

77.1.6.2.1 TRibbonAddRemoveItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.6.2.2 TRibbonAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.7 TRibbonApplicationButton Class

Class Hierarchy

Pascal

```
TRibbonApplicationButton = class(TCustomMenuButton);
```

C++

```
class TRibbonApplicationButton : public TCustomMenuButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonApplicationButton.

77.1.7.1 TRibbonApplicationButton Members

The following tables list the members exposed by TRibbonApplicationButton.

Public Methods

	Name	Description
	CalcBounds (see page 6967)	Calculates the size of the control.

77.1.7.2 TRibbonApplicationButton Methods

The methods of the TRibbonApplicationButton class are listed here.

Public Methods

	Name	Description
	CalcBounds (see page 6967)	Calculates the size of the control.

77.1.7.2.1 TRibbonApplicationButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.8 TRibbonApplicationMenuButton Class

Application (see page 2612) Button when using Ribbon (see page 6713) interface.

Class Hierarchy

Pascal

```
TRibbonApplicationMenuButton = class(TCustomMenuButton, IRibbonKeyTip);
```

C++

```
class TRibbonApplicationMenuButton : public TCustomMenuButton, public IRibbonKeyTip;
```

File

RibbonActnCtrls

Description

TRibbonApplicationMenuButton is used for the Application (see page 2612) Button when using the Ribbon (see page 6713) interface.

77.1.8.1 TRibbonApplicationMenuButton Members

The following tables list the members exposed by TRibbonApplicationMenuButton.

Protected Methods

	Name	Description
	CMMouseEnter (see page 6971)	This is CMMouseEnter, a member of class TRibbonApplicationMenuButton.
	CMMouseLeave (see page 6971)	This is CMMouseLeave, a member of class TRibbonApplicationMenuButton.
	DragOver (see page 6971)	OnDragOver event dispatcher.
	Paint (see page 6973)	Render control surface.
	RibbonStyle (see page 6973)	This is RibbonStyle, a member of class TRibbonApplicationMenuButton.
	SetParent (see page 6974)	Sets the parent of the control.
	UMFadeControl (see page 6975)	This is UMFadeControl, a member of class TRibbonApplicationMenuButton.
	WMLButtonDblClk (see page 6975)	This is WMLButtonDblClk, a member of class TRibbonApplicationMenuButton.

Protected Properties

	Name	Description
	MouseInControl (see page 6976)	This is MouseInControl, a member of class TRibbonApplicationMenuButton.

Public Methods

	Name	Description
	CalcBounds (see page 6970)	Calculates the size of the control.
	Create (see page 6969)	Creates an instance of TCustomMenuButton.
	Destroy (see page 6969)	Destroys an instance of TCustomMenuButton.
	GetKeyTip (see page 6971)	Returns current **KeyTip** for command.

	HandleKeyTip (see page 6972)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6972)	Hide **KeyTip** for command.
	KeyTipClick (see page 6973)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6974)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6975)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 6976)	Current **KeyTip** for command

77.1.8.2 TRibbonApplicationMenuButton.Create Constructor

Creates an instance of TCustomMenuButton.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuButton. However, you should never attempt to instantiate a TCustomMenuButton. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the menu button instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.8.3 TRibbonApplicationMenuButton.Destroy Destructor

Destroys an instance of TCustomMenuButton.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonApplicationMenuButton();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

77.1.8.4 TRibbonApplicationMenuButton Methods

The methods of the TRibbonApplicationMenuButton class are listed here.

Protected Methods

	Name	Description
	CMMouseEnter (see page 6971)	This is CMMouseEnter, a member of class TRibbonApplicationMenuButton.
	CMMouseLeave (see page 6971)	This is CMMouseLeave, a member of class TRibbonApplicationMenuButton.
	DragOver (see page 6971)	OnDragOver event dispatcher.
	Paint (see page 6973)	Render control surface.
	RibbonStyle (see page 6973)	This is RibbonStyle, a member of class TRibbonApplicationMenuButton.
	SetParent (see page 6974)	Sets the parent of the control.
	UMFadeControl (see page 6975)	This is UMFadeControl, a member of class TRibbonApplicationMenuButton.
	WMLButtonDblClk (see page 6975)	This is WMLButtonDblClk, a member of class TRibbonApplicationMenuButton.

Public Methods

	Name	Description
	CalcBounds (see page 6970)	Calculates the size of the control.
	GetKeyTip (see page 6971)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 6972)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6972)	Hide **KeyTip** for command.
	KeyTipClick (see page 6973)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6974)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6975)	Shows **KeyTip** for command in specified position.

77.1.8.4.1 TRibbonApplicationMenuButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.8.4.2 TRibbonApplicationMenuButton.CMMouseEnter Method

Pascal

```
procedure CMMouseEnter(var Message: TMessage); override;
```

C++

```
virtual __fastcall CMMouseEnter(TMessage Message);
```

Description

This is CMMouseEnter, a member of class TRibbonApplicationMenuButton.

77.1.8.4.3 TRibbonApplicationMenuButton.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); override;
```

C++

```
virtual __fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TRibbonApplicationMenuButton.

77.1.8.4.4 TRibbonApplicationMenuButton.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop

OnDragOver

77.1.8.4.5 TRibbonApplicationMenuButton.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.8.4.6 TRibbonApplicationMenuButton.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.8.4.7 TRibbonApplicationMenuButton.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.8.4.8 TRibbonApplicationMenuButton.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.8.4.9 TRibbonApplicationMenuButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

77.1.8.4.10 TRibbonApplicationMenuButton.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonApplicationMenuButton.

77.1.8.4.11 TRibbonApplicationMenuButton.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

77.1.8.4.12 TRibbonApplicationMenuButton.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.8.4.13 TRibbonApplicationMenuButton.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

77.1.8.4.14 TRibbonApplicationMenuButton.UMFadeControl Method

Pascal

```
procedure UMFadeControl(var Message: TMessage); message UM_FADECONTROL;
```

C++

```
__fastcall UMFadeControl(TMessage Message);
```

Description

This is UMFadeControl, a member of class TRibbonApplicationMenuButton.

77.1.8.4.15 TRibbonApplicationMenuButton.WMLButtonDblClk Method

Pascal

```
procedure WMLButtonDblClk(var Message: TWMLButtonDblClk); message WM_LBUTTONDBLCLK;
```

C++

```
__fastcall WMLButtonDblClk(TWMLButtonDblClk Message);
```

Description

This is WMLButtonDblClk, a member of class TRibbonApplicationMenuButton.

77.1.8.5 TRibbonApplicationMenuButton Properties

The properties of the TRibbonApplicationMenuButton class are listed here.

Protected Properties

	Name	Description
	MouseInControl (see page 6976)	This is MouseInControl, a member of class TRibbonApplicationMenuButton.

Public Properties

	Name	Description
	KeyTip (see page 6976)	Current **KeyTip** for command

77.1.8.5.1 TRibbonApplicationMenuButton.KeyTip Property

Current **KeyTip** for command

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

KeyTip is the current **KeyTip** for the command

See Also

GetKeyTip (see page 7279)

SetKeyTip (see page 7281)

77.1.8.5.2 TRibbonApplicationMenuButton.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TRibbonApplicationMenuButton.

77.1.9 TRibbonApplicationMenuItem Class

Class Hierarchy

Pascal

```
TRibbonApplicationMenuItem = class(TRibbonMenuItem);
```

C++

```
class TRibbonApplicationMenuItem : public TRibbonMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonApplicationMenuItem.

77.1.9.1 TRibbonApplicationMenuItem Members

The following tables list the members exposed by TRibbonApplicationMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7121)	Draws a background image on the control.
	DrawGlyph (see page 6978)	Draws a symbol on the control.
	DrawSelectedBackground (see page 6979)	This is DrawSelectedBackground, a member of class TRibbonApplicationMenuItem.
	DrawSeparator (see page 6979)	This is DrawSeparator, a member of class TRibbonApplicationMenuItem.
	DrawShadowedText (see page 6979)	This is DrawShadowedText, a member of class TRibbonApplicationMenuItem.
	DrawSubMenuGlyph (see page 6979)	This is DrawSubMenuGlyph, a member of class TRibbonApplicationMenuItem.
	DrawText (see page 6980)	Draws text on the control.
	GetImageSize (see page 6980)	This is GetImageSize, a member of class TRibbonApplicationMenuItem.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 6978)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 6977)	Creates an instance of TCustomMenuItem.

77.1.9.2 TRibbonApplicationMenuItem.Create Constructor

Creates an instance of TCustomMenuItem.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuItem. However, you should never attempt to instantiate a TCustomMenuItem. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

77.1.9.3 TRibbonApplicationMenuItem Methods

The methods of the TRibbonApplicationMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawGlyph (see page 6978)	Draws a symbol on the control.
	DrawSelectedBackground (see page 6979)	This is DrawSelectedBackground, a member of class TRibbonApplicationMenuItem.
	DrawSeparator (see page 6979)	This is DrawSeparator, a member of class TRibbonApplicationMenuItem.
	DrawShadowedText (see page 6979)	This is DrawShadowedText, a member of class TRibbonApplicationMenuItem.
	DrawSubMenuGlyph (see page 6979)	This is DrawSubMenuGlyph, a member of class TRibbonApplicationMenuItem.
	DrawText (see page 6980)	Draws text on the control.
	GetImageSize (see page 6980)	This is GetImageSize, a member of class TRibbonApplicationMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 6978)	Calculates the size of the menu item.

77.1.9.3.1 TRibbonApplicationMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.9.3.2 TRibbonApplicationMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.9.3.3 TRibbonApplicationMenuItem.DrawSelectedBackground Method

Pascal

```
procedure DrawSelectedBackground(ARect: TRect); override;
```

C++

```
virtual __fastcall DrawSelectedBackground(TRect ARect);
```

Description

This is DrawSelectedBackground, a member of class TRibbonApplicationMenuItem.

77.1.9.3.4 TRibbonApplicationMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TRibbonApplicationMenuItem.

77.1.9.3.5 TRibbonApplicationMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonApplicationMenuItem.

77.1.9.3.6 TRibbonApplicationMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; override;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TRibbonApplicationMenuItem.

77.1.9.3.7 TRibbonApplicationMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.9.3.8 TRibbonApplicationMenuItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

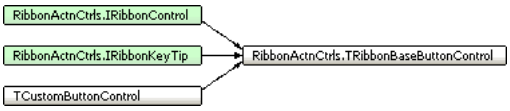
Description

This is GetImageSize, a member of class TRibbonApplicationMenuItem.

77.1.10 TRibbonBaseButtonControl Class

Common ancestor for many buttons in groups and Quick Access Toolbar.

Class Hierarchy



Pascal

```
TRibbonBaseButtonControl = class(TCustomButtonControl, IRibbonControl, IRibbonKeyTip);
```

C++

```
class TRibbonBaseButtonControl : public TCustomButtonControl, public IRibbonControl, public
IRibbonKeyTip;
```

File

RibbonActnCtrls

Description

TRibbonBaseButtonControl is the common ancestor for many buttons that appear in the groups and on the Quick Access Toolbar.

77.1.10.1 TRibbonBaseButtonControl Members

The following tables list the members exposed by TRibbonBaseButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.10.2 TRibbonBaseButtonControl.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.10.3 TRibbonBaseButtonControl.Destroy Destructor

Destroys an instance of TCustomActionControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonBaseButtonControl();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

77.1.10.4 TRibbonBaseButtonControl Methods

The methods of the TRibbonBaseButtonControl class are listed here.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

77.1.10.4.1 TRibbonBaseButtonControl.AllowHotKeys Method

Pascal

```
function AllowHotKeys: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AllowHotKeys();
```

Description

This is AllowHotKeys, a member of class TRibbonBaseButtonControl.

77.1.10.4.2 TRibbonBaseButtonControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.10.4.3 TRibbonBaseButtonControl.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.

77.1.10.4.4 TRibbonBaseButtonControl.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.10.4.5 TRibbonBaseButtonControl.DragDrop Method

Performs actions and generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

This override of the method inherited from TControl executes instructions and generates an OnDragDrop event. The method is

called when a user drags and drops an item (Source) onto the custom action control at coordinates (X,Y).

See Also

DragDrop

77.1.10.4.6 TRibbonBaseButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.10.4.7 TRibbonBaseButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.10.4.8 TRibbonBaseButtonControl.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TRibbonBaseButtonControl.

77.1.10.4.9 TRibbonBaseButtonControl.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonBaseButtonControl.

77.1.10.4.10 TRibbonBaseButtonControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.10.4.11 TRibbonBaseButtonControl.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TRibbonBaseButtonControl.

77.1.10.4.12 TRibbonBaseButtonControl.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.10.4.13 TRibbonBaseButtonControl.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.10.4.14 TRibbonBaseButtonControl.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.10.4.15 TRibbonBaseButtonControl.IsButton Method

Pascal

```
function IsButton(out ButtonProperties: TButtonProperties): Boolean;
```

C++

```
__fastcall Boolean IsButton(TButtonProperties ButtonProperties);
```

Description

This is IsButton, a member of class TRibbonBaseButtonControl.

77.1.10.4.16 TRibbonBaseButtonControl.IsRibbonAvailable Method

Pascal

```
function IsRibbonAvailable: Boolean;
```

C++

```
__fastcall Boolean IsRibbonAvailable();
```

Description

This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.

77.1.10.4.17 TRibbonBaseButtonControl.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick; virtual;
```

C++

```
virtual __fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.10.4.18 TRibbonBaseButtonControl.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonBaseButtonControl.

77.1.10.4.19 TRibbonBaseButtonControl.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

77.1.10.4.20 TRibbonBaseButtonControl.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

77.1.10.5 TRibbonBaseButtonControl Properties

The properties of the TRibbonBaseButtonControl class are listed here.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

77.1.10.5.1 TRibbonBaseButtonControl.DrawDown Property

Pascal

```
property DrawDown: Boolean;
```

C++

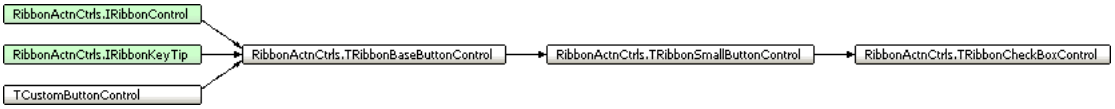
```
__property Boolean DrawDown;
```

Description

This is DrawDown, a member of class TRibbonBaseButtonControl.

77.1.11 TRibbonCheckBoxControl Class

Class Hierarchy



Pascal

```
TRibbonCheckBoxControl = class(TRibbonSmallButtonControl);
```

C++

```
class TRibbonCheckBoxControl : public TRibbonSmallButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonCheckBoxControl.

77.1.11.1 TRibbonCheckBoxControl Members

The following tables list the members exposed by TRibbonCheckBoxControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 6992)	Draws a background image on the control.
	DrawGlyph (see page 6992)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6991)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.11.2 TRibbonCheckBoxControl Methods

The methods of the TRibbonCheckBoxControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 6992)	Draws a background image on the control.
	DrawGlyph (see page 6992)	Draws a symbol on the control.

Public Methods

	Name	Description
	CalcBounds (see page 6991)	Calculates the size of the control.

77.1.11.2.1 TRibbonCheckBoxControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.11.2.2 TRibbonCheckBoxControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.11.2.3 TRibbonCheckBoxControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.12 TRibbonComboBox Class

Class Hierarchy

Pascal

```
TRibbonComboBox = class(TCustomRibbonComboBox);
```

C++

```
class TRibbonComboBox : public TCustomRibbonComboBox;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonComboBox.

77.1.12.1 TRibbonComboBox Members

The following tables list the members exposed by TRibbonComboBox.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 6949)	Begin Automatic Drag.
	CMMouseenter (see page 6949)	This is CMMouseenter, a member of class TCustomRibbonComboBox.
	CMMouseleave (see page 6950)	This is CMMouseleave, a member of class TCustomRibbonComboBox.
	DragOver (see page 6950)	OnDragOver event dispatcher.
	KeyDown (see page 6951)	Respond to key press events.
	KeyPress (see page 6951)	Respond to keyboard input.
	MouseMove (see page 6952)	Respond to mouse moving over control area..
	Notification (see page 6952)	Responds to notifications that components are being created or destroyed.
	SetParent (see page 6953)	Sets the parent of the control.
	WMKillFocus (see page 6954)	This is WMKillFocus, a member of class TCustomRibbonComboBox.
	WMSetFocus (see page 6954)	This is WMSetFocus, a member of class TCustomRibbonComboBox.

Public Methods

	Name	Description
	Create (see page 6948)	Creates an instance of TCustomEdit.
	Destroy (see page 6948)	Destroys an instance of TWinControl.
	DragDrop (see page 6950)	OnDragDrop event dispatcher.
	SetBounds (see page 6953)	Sets the windowed control's boundary properties all at once.

Public Properties

	Name	Description
	Color (see page 6955)	Specifies the background color of the control.
	MouseInControl (see page 6956)	This is MouseInControl, a member of class TCustomRibbonComboBox.

Published Events

	Name	Description
	OnChange (see page 7012)	Occurs when the text for the edit control may have changed.
	OnClick (see page 7013)	Occurs when the user clicks the control.
	OnContextPopup (see page 7015)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 7015)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 7016)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7019)	Occurs when the user drags an object over a control.
	OnEndDock (see page 7022)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 7022)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7023)	Occurs when a control receives the input focus.
	OnExit (see page 7026)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 7029)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 7032)	Occurs when a key is pressed.
	OnKeyUp (see page 7033)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 7035)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 7036)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 7038)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 7038)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 7039)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 7041)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 7043)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 7044)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Alignment (see page 6995)	Determines how the text is aligned within the text edit control.
	AutoSelect (see page 6995)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	CharCase (see page 6996)	Determines the case of the text within the edit control.
	Font (see page 6996)	Controls the attributes of text written on or in the control.
	HideSelection (see page 6997)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6998)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6998)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Items (see page 6999)	This is Items, a member of class TRibbonComboBox.
	MaxLength (see page 6999)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 7000)	Allows only numbers to be typed into the text edit.
	ParentFont (see page 7000)	Determines where a control looks for its font information.
	PasswordChar (see page 7000)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 7001)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 7006)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 7006)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7008)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7008)	Determines if the user can tab to a control.
	Text (see page 7009)	Contains a text string associated with the control.
	TextHint (see page 7010)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 7010)	Determines whether the component appears onscreen.

77.1.12.2 TRibbonComboBox Properties

The properties of the TRibbonComboBox class are listed here.

Published Properties

	Name	Description
	Alignment (see page 6995)	Determines how the text is aligned within the text edit control.

	AutoSelect (see page 6995)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	CharCase (see page 6996)	Determines the case of the text within the edit control.
	Font (see page 6996)	Controls the attributes of text written on or in the control.
	HideSelection (see page 6997)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6998)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6998)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Items (see page 6999)	This is Items, a member of class TRibbonComboBox.
	MaxLength (see page 6999)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 7000)	Allows only numbers to be typed into the text edit.
	ParentFont (see page 7000)	Determines where a control looks for its font information.
	PasswordChar (see page 7000)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 7001)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 7006)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 7006)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7008)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7008)	Determines if the user can tab to a control.
	Text (see page 7009)	Contains a text string associated with the control.
	TextHint (see page 7010)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 7010)	Determines whether the component appears onscreen.

77.1.12.2.1 TRibbonComboBox.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment;
```

C++

```
__property Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment

Text

Displaying and Editing Fields in an Edit Box

77.1.12.2.2 TRibbonComboBox.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect;
```

C++

```
__property AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll

SelLength

SelStart

SelText

SetSelTextBuf

77.1.12.2.3 TRibbonComboBox.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase;
```

C++

```
__property CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

77.1.12.2.4 TRibbonComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

77.1.12.2.5 TRibbonComboBox.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection;
```

C++

```
__property HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have

focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength

SelStart

SelText

SetSelTextBuf

77.1.12.2.6 TRibbonComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

77.1.12.2.7 TRibbonComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme (see page 2462)

Imes (see page 2472)

77.1.12.2.8 TRibbonComboBox.Items Property

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

This is Items, a member of class TRibbonComboBox.

77.1.12.2.9 TRibbonComboBox.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength;
```

C++

```
__property MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

77.1.12.2.10 TRibbonComboBox.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly;
```

C++

```
__property NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

77.1.12.2.11 TRibbonComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

77.1.12.2.12 TRibbonComboBox.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar;
```

C++

```
__property PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

77.1.12.2.13 TRibbonComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
```

```
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
```

```
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

77.1.12.2.14 TRibbonComboBox.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard

GetSelTextBuf

77.1.12.2.15 TRibbonComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

77.1.12.2.16 TRibbonComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

77.1.12.2.17 TRibbonComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

77.1.12.2.18 TRibbonComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                       //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines

77.1.12.2.19 TRibbonComboBox.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint;
```

C++

```
__property TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

77.1.12.2.20 TRibbonComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

77.1.12.3 TRibbonComboBox Events

The events of the TRibbonComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 7012)	Occurs when the text for the edit control may have changed.
	OnClick (see page 7013)	Occurs when the user clicks the control.
	OnContextPopup (see page 7015)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 7015)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 7016)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7019)	Occurs when the user drags an object over a control.
	OnEndDock (see page 7022)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 7022)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7023)	Occurs when a control receives the input focus.
	OnExit (see page 7026)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 7029)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 7032)	Occurs when a key is pressed.
	OnKeyUp (see page 7033)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 7035)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 7036)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 7038)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 7038)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 7039)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 7041)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 7043)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 7044)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

77.1.12.3.1 TRibbonComboBox.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change

Text

77.1.12.3.2 TRibbonComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

77.1.12.3.3 TRibbonComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 4935)

PopupMenu

AutoPopup (see page 4942)

77.1.12.3.4 TRibbonComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

77.1.12.3.5 TRibbonComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

77.1.12.3.6 TRibbonComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

77.1.12.3.7 TRibbonComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

77.1.12.3.8 TRibbonComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
```

```
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

77.1.12.3.9 TRibbonComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

77.1.12.3.10 TRibbonComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
```

```
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

77.1.12.3.11 TRibbonComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

77.1.12.3.12 TRibbonComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

77.1.12.3.13 TRibbonComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

77.1.12.3.14 TRibbonComboBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control.

However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

77.1.12.3.15 TRibbonComboBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
```

```
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
```

```
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

77.1.12.3.16 TRibbonComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

77.1.12.3.17 TRibbonComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

77.1.12.3.18 TRibbonComboBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

77.1.12.3.19 TRibbonComboBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

77.1.12.3.20 TRibbonComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

77.1.12.3.21 TRibbonComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

77.1.13 TRibbonComboBoxItem Class

Class Hierarchy

Pascal

```
TRibbonComboBoxItem = class(TRibbonMenuItem);
```

C++

```
class TRibbonComboBoxItem : public TRibbonMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonComboBoxItem.

77.1.13.1 TRibbonComboBoxItem Members

The following tables list the members exposed by TRibbonComboBoxItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7046)	Draws a background image on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawText (see page 7046)	Draws text on the control.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7046)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 7119)	Creates an instance of TCustomMenuItem.

77.1.13.2 TRibbonComboBoxItem Methods

The methods of the TRibbonComboBoxItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7046)	Draws a background image on the control.
	DrawText (see page 7046)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7046)	Calculates the size of the menu item.

77.1.13.2.1 TRibbonComboBoxItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.13.2.2 TRibbonComboBoxItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.13.2.3 TRibbonComboBoxItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

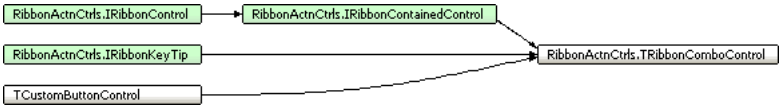
See Also

DrawBackground

DrawGlyph

77.1.14 TRibbonComboControl Class

Class Hierarchy



Pascal

```
TRibbonComboControl = class(TCustomButtonControl, IRibbonContainedControl, IRibbonKeyTip);
```

C++

```
class TRibbonComboControl : public TCustomButtonControl, public IRibbonContainedControl,
public IRibbonKeyTip;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonComboControl.

77.1.14.1 TRibbonComboControl Members

The following tables list the members exposed by TRibbonComboControl.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 7051)	This is CMBidimodechanged, a member of class TRibbonComboControl.
	CMMouseenter (see page 7051)	This is CMMouseenter, a member of class TRibbonComboControl.
	CMMouseleave (see page 7051)	This is CMMouseleave, a member of class TRibbonComboControl.
	CMVisiblechanged (see page 7052)	This is CMVisiblechanged, a member of class TRibbonComboControl.
	ComboClick (see page 7052)	This is ComboClick, a member of class TRibbonComboControl.
V	DrawBackground (see page 7052)	Draws a background image on the control.
	DrawFrame (see page 7052)	This is DrawFrame, a member of class TRibbonComboControl.
V	DropDownClick (see page 7053)	This is DropDownClick, a member of class TRibbonComboControl.
	GetCaptionWidth (see page 7053)	This is GetCaptionWidth, a member of class TRibbonComboControl.
	GetGlyphWidth (see page 7053)	This is GetGlyphWidth, a member of class TRibbonComboControl.
	GetLabelWidth (see page 7054)	This is GetLabelWidth, a member of class TRibbonComboControl.
V	MouseDown (see page 7055)	OnMouseDown event dispatcher.
V	Notification (see page 7056)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 7056)	Render control surface.
	RibbonStyle (see page 7057)	This is RibbonStyle, a member of class TRibbonComboControl.

	SetActionClient (see page 7057)	This is SetActionClient, a member of class TRibbonComboControl.
	SetDragMode (see page 7058)	Sets the drag mode for the control.
	SetParent (see page 7058)	Sets the parent of the control.
	ShowComboItems (see page 7059)	This is ShowComboItems, a member of class TRibbonComboControl.

Public Methods

	Name	Description
	CalcBounds (see page 7050)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7050)	This is CanAddToQuickAccessToolbar, a member of class TRibbonComboControl.
	Click (see page 7051)	Executes the control's function.
	Create (see page 7048)	Creates an instance of TCustomButtonControl.
	Destroy (see page 7049)	Destroys an instance of TCustomActionControl.
	GetContainedControl (see page 7053)	This is GetContainedControl, a member of class TRibbonComboControl.
	GetKeyTip (see page 7053)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7054)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7055)	Hide **KeyTip** for command.
	KeyTipClick (see page 7055)	Handle clicking a **KeyTip**.
	SetBounds (see page 7057)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7057)	This is SetContainedControl, a member of class TRibbonComboControl.
	SetKeyTip (see page 7058)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7059)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	ComboBox (see page 7060)	This is ComboBox, a member of class TRibbonComboControl.
	ContainedControl (see page 7060)	This is ContainedControl, a member of class TRibbonComboControl.
	KeyTip (see page 7282)	Current **KeyTip** for command
	MouseInControl (see page 7060)	This is MouseInControl, a member of class TRibbonComboControl.

77.1.14.2 TRibbonComboControl.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.14.3 TRibbonComboControl.Destroy Destructor

Destroys an instance of TCustomActionControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonComboControl();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

77.1.14.4 TRibbonComboControl Methods

The methods of the TRibbonComboControl class are listed here.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 7051)	This is CMBidimodechanged, a member of class TRibbonComboControl.
	CMMouseenter (see page 7051)	This is CMMouseenter, a member of class TRibbonComboControl.
	CMMouseleave (see page 7051)	This is CMMouseleave, a member of class TRibbonComboControl.
	CMVisiblechanged (see page 7052)	This is CMVisiblechanged, a member of class TRibbonComboControl.
	ComboClick (see page 7052)	This is ComboClick, a member of class TRibbonComboControl.
V	DrawBackground (see page 7052)	Draws a background image on the control.
	DrawFrame (see page 7052)	This is DrawFrame, a member of class TRibbonComboControl.
V	DropDownClick (see page 7053)	This is DropDownClick, a member of class TRibbonComboControl.
	GetCaptionWidth (see page 7053)	This is GetCaptionWidth, a member of class TRibbonComboControl.
	GetGlyphWidth (see page 7053)	This is GetGlyphWidth, a member of class TRibbonComboControl.
	GetLabelWidth (see page 7054)	This is GetLabelWidth, a member of class TRibbonComboControl.
V	MouseDown (see page 7055)	OnMouseDown event dispatcher.
V	Notification (see page 7056)	Responds to notifications that components are being created or destroyed.

	Paint (see page 7056)	Render control surface.
	RibbonStyle (see page 7057)	This is RibbonStyle, a member of class TRibbonComboControl.
	SetActionClient (see page 7057)	This is SetActionClient, a member of class TRibbonComboControl.
	SetDragMode (see page 7058)	Sets the drag mode for the control.
	SetParent (see page 7058)	Sets the parent of the control.
	ShowComboItems (see page 7059)	This is ShowComboItems, a member of class TRibbonComboControl.

Public Methods

	Name	Description
	CalcBounds (see page 7050)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7050)	This is CanAddToQuickAccessToolbar, a member of class TRibbonComboControl.
	Click (see page 7051)	Executes the control's function.
	GetContainedControl (see page 7053)	This is GetContainedControl, a member of class TRibbonComboControl.
	GetKeyTip (see page 7053)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7054)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7055)	Hide **KeyTip** for command.
	KeyTipClick (see page 7055)	Handle clicking a **KeyTip**.
	SetBounds (see page 7057)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7057)	This is SetContainedControl, a member of class TRibbonComboControl.
	SetKeyTip (see page 7058)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7059)	Shows **KeyTip** for command in specified position.

77.1.14.4.1 TRibbonComboControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.14.4.2 TRibbonComboControl.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class TRibbonComboControl.

77.1.14.4.3 TRibbonComboControl.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.14.4.4 TRibbonComboControl.CMBidimodechanged Method

Pascal

```
procedure CMBidimodechanged(var Message: TMessage); message CM_BIDIMODECHANGED;
```

C++

```
__fastcall CMBidimodechanged(TMessage Message);
```

Description

This is CMBidimodechanged, a member of class TRibbonComboControl.

77.1.14.4.5 TRibbonComboControl.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TRibbonComboControl.

77.1.14.4.6 TRibbonComboControl.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TRibbonComboControl.

77.1.14.4.7 TRibbonComboControl.CMVisiblechanged Method

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

This is CMVisiblechanged, a member of class TRibbonComboControl.

77.1.14.4.8 TRibbonComboControl.ComboClick Method

Pascal

```
procedure ComboClick(Sender: TObject);
```

C++

```
__fastcall ComboClick(TObject * Sender);
```

Description

This is ComboClick, a member of class TRibbonComboControl.

77.1.14.4.9 TRibbonComboControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.14.4.10 TRibbonComboControl.DrawFrame Method

Pascal

```
procedure DrawFrame; reintroduce;
```

C++

```
__fastcall DrawFrame();
```

Description

This is DrawFrame, a member of class TRibbonComboControl.

77.1.14.4.11 TRibbonComboControl.DropDownClick Method

Pascal

```
procedure DropDownClick(const SelectFirst: Boolean); virtual;
```

C++

```
virtual __fastcall DropDownClick(const Boolean SelectFirst);
```

Description

This is DropDownClick, a member of class TRibbonComboControl.

77.1.14.4.12 TRibbonComboControl.GetCaptionWidth Method

Pascal

```
function GetCaptionWidth: Integer;
```

C++

```
__fastcall int GetCaptionWidth();
```

Description

This is GetCaptionWidth, a member of class TRibbonComboControl.

77.1.14.4.13 TRibbonComboControl.GetContainedControl Method

Pascal

```
function GetContainedControl: TControl;
```

C++

```
__fastcall TControl * GetContainedControl();
```

Description

This is GetContainedControl, a member of class TRibbonComboControl.

77.1.14.4.14 TRibbonComboControl.GetGlyphWidth Method

Pascal

```
function GetGlyphWidth: Integer;
```

C++

```
__fastcall int GetGlyphWidth();
```

Description

This is GetGlyphWidth, a member of class TRibbonComboControl.

77.1.14.4.15 TRibbonComboControl.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.14.4.16 TRibbonComboControl.GetLabelWidth Method

Pascal

```
function GetLabelWidth: Integer;
```

C++

```
__fastcall int GetLabelWidth();
```

Description

This is GetLabelWidth, a member of class TRibbonComboControl.

77.1.14.4.17 TRibbonComboControl.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.14.4.18 TRibbonComboControl.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.14.4.19 TRibbonComboControl.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.14.4.20 TRibbonComboControl.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of

the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

77.1.14.4.21 TRibbonComboControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

77.1.14.4.22 TRibbonComboControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant

control classes.

See Also

Canvas

77.1.14.4.23 TRibbonComboControl.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonComboControl.

77.1.14.4.24 TRibbonComboControl.SetActionClient Method

Pascal

```
procedure SetActionClient(Value: TActionClientItem); override;
```

C++

```
virtual __fastcall SetActionClient(TActionClientItem Value);
```

Description

This is SetActionClient, a member of class TRibbonComboControl.

77.1.14.4.25 TRibbonComboControl.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

This override of the method inherited from TControl simultaneously sets all boundary properties at once, and thus avoiding the possibility of a control redraw between the time one value sets and another. In addition, some layout calculations are also performed.

See Also

SetBounds

77.1.14.4.26 TRibbonComboControl.SetContainedControl Method

Pascal

```
procedure SetContainedControl(const Value: TControl);
```

C++

```
__fastcall SetContainedControl(const TControl * Value);
```

Description

This is SetContainedControl, a member of class TRibbonComboControl.

77.1.14.4.27 TRibbonComboControl.SetDragMode Method

Sets the drag mode for the control.

Pascal

```
procedure SetDragMode(Value: TDragMode); override;
```

C++

```
virtual __fastcall SetDragMode(TDragMode Value);
```

Description

SetDragMode is the protected write implementation for the DragMode property. Override SetDragMode to execute additional code when the value of the DragMode property changes.

See Also

DragMode

77.1.14.4.28 TRibbonComboControl.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

77.1.14.4.29 TRibbonComboControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.14.4.30 TRibbonComboControl.ShowComboItems Method

Pascal

```
procedure ShowComboItems(const SelectFirst: Boolean); virtual;
```

C++

```
virtual __fastcall ShowComboItems(const Boolean SelectFirst);
```

Description

This is ShowComboItems, a member of class TRibbonComboControl.

77.1.14.4.31 TRibbonComboControl.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

77.1.14.5 TRibbonComboControl Properties

The properties of the TRibbonComboControl class are listed here.

Public Properties

	Name	Description
	ComboBox (see page 7060)	This is ComboBox, a member of class TRibbonComboControl.
	ContainedControl (see page 7060)	This is ContainedControl, a member of class TRibbonComboControl.
	MouseInControl (see page 7060)	This is MouseInControl, a member of class TRibbonComboControl.

77.1.14.5.1 TRibbonComboControl.ComboBox Property

Pascal

```
property ComboBox: TCustomRibbonComboBox;
```

C++

```
__property TCustomRibbonComboBox ComboBox;
```

Description

This is ComboBox, a member of class TRibbonComboControl.

77.1.14.5.2 TRibbonComboControl.ContainedControl Property

Pascal

```
property ContainedControl: TControl;
```

C++

```
__property TControl * ContainedControl;
```

Description

This is ContainedControl, a member of class TRibbonComboControl.

77.1.14.5.3 TRibbonComboControl.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

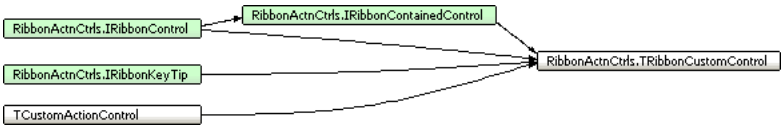
```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TRibbonComboControl.

77.1.15 TRibbonCustomControl Class

Class Hierarchy



Pascal

```
TRibbonCustomControl = class(TCustomActionControl, IRibbonControl, IRibbonContainedControl,
IRibbonKeyTip);
```

C++

```
class TRibbonCustomControl : public TCustomActionControl, public IRibbonControl, public
IRibbonContainedControl, public IRibbonKeyTip;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonCustomControl.

77.1.15.1 TRibbonCustomControl Members

The following tables list the members exposed by TRibbonCustomControl.

Protected Methods

	Name	Description
	CMVisiblechanged (see page 7064)	This is CMVisiblechanged, a member of class TRibbonCustomControl.
V	DrawBackground (see page 7065)	Draws a background image on the control.
	GetCaptionWidth (see page 7065)	This is GetCaptionWidth, a member of class TRibbonCustomControl.
	GetGlyphWidth (see page 7065)	This is GetGlyphWidth, a member of class TRibbonCustomControl.
	GetLabelWidth (see page 7066)	This is GetLabelWidth, a member of class TRibbonCustomControl.
V	Notification (see page 7067)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 7068)	Render control surface.
V	SetActionClient (see page 7068)	This is SetActionClient, a member of class TRibbonCustomControl.
V	SetDragMode (see page 7071)	Sets the drag mode for the control.
V	SetParent (see page 7072)	Sets the parent of the control.

Public Methods

	Name	Description
V	CalcBounds (see page 7063)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7064)	This is CanAddToQuickAccessToolbar, a member of class TRibbonCustomControl.
V	Click (see page 7064)	Executes the control's function in response to a mouse click.
V	Create (see page 7062)	Creates an instance of TCustomActionControl.
V	Destroy (see page 7062)	Destroys an instance of TCustomActionControl.
	GetContainedControl (see page 7065)	This is GetContainedControl, a member of class TRibbonCustomControl.
	GetKeyTip (see page 7066)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7066)	Indicates if command's **KeyTip** matches currently input KeyTip.

	HideKeyTip (see page 7067)	Hide **KeyTip** for command.
	KeyTipClick (see page 7067)	Handle clicking a **KeyTip**.
	SetBounds (see page 7068)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7071)	This is SetContainedControl, a member of class TRibbonCustomControl.
	SetKeyTip (see page 7071)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7072)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	ContainedControl (see page 7073)	This is ContainedControl, a member of class TRibbonCustomControl.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.15.2 TRibbonCustomControl.Create Constructor

Creates an instance of TCustomActionControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomActionControl. However, you should never attempt to instantiate a TCustomActionControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the TCustomActionControl instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.15.3 TRibbonCustomControl.Destroy Destructor

Destroys an instance of TCustomActionControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonCustomControl();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create

System::TObject::Free

77.1.15.4 TRibbonCustomControl Methods

The methods of the TRibbonCustomControl class are listed here.

Protected Methods

	Name	Description
	CMVisiblechanged (see page 7064)	This is CMVisiblechanged, a member of class TRibbonCustomControl.
V	DrawBackground (see page 7065)	Draws a background image on the control.
	GetCaptionWidth (see page 7065)	This is GetCaptionWidth, a member of class TRibbonCustomControl.
	GetGlyphWidth (see page 7065)	This is GetGlyphWidth, a member of class TRibbonCustomControl.
	GetLabelWidth (see page 7066)	This is GetLabelWidth, a member of class TRibbonCustomControl.
V	Notification (see page 7067)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 7068)	Render control surface.
V	SetActionClient (see page 7068)	This is SetActionClient, a member of class TRibbonCustomControl.
V	SetDragMode (see page 7071)	Sets the drag mode for the control.
V	SetParent (see page 7072)	Sets the parent of the control.

Public Methods

	Name	Description
V	CalcBounds (see page 7063)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7064)	This is CanAddToQuickAccessToolbar, a member of class TRibbonCustomControl.
V	Click (see page 7064)	Executes the control's function in response to a mouse click.
	GetContainedControl (see page 7065)	This is GetContainedControl, a member of class TRibbonCustomControl.
	GetKeyTip (see page 7066)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7066)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7067)	Hide **KeyTip** for command.
	KeyTipClick (see page 7067)	Handle clicking a **KeyTip**.
V	SetBounds (see page 7068)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7071)	This is SetContainedControl, a member of class TRibbonCustomControl.
	SetKeyTip (see page 7071)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7072)	Shows **KeyTip** for command in specified position.

77.1.15.4.1 TRibbonCustomControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.15.4.2 TRibbonCustomControl.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class TRibbonCustomControl.

77.1.15.4.3 TRibbonCustomControl.Click Method

Executes the control's function in response to a mouse click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TControl executes the control's function in response to a user's mouse click. Usage data, which is used to determine whether a control is used frequently enough to be shown by default, is updated as well.

See Also

Click

UsageCount

Keyed

77.1.15.4.4 TRibbonCustomControl.CMVisiblechanged Method

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

This is CMVisiblechanged, a member of class TRibbonCustomControl.

77.1.15.4.5 TRibbonCustomControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.15.4.6 TRibbonCustomControl.GetCaptionWidth Method

Pascal

```
function GetCaptionWidth: Integer;
```

C++

```
__fastcall int GetCaptionWidth();
```

Description

This is GetCaptionWidth, a member of class TRibbonCustomControl.

77.1.15.4.7 TRibbonCustomControl.GetContainedControl Method

Pascal

```
function GetContainedControl: TControl;
```

C++

```
__fastcall TControl * GetContainedControl();
```

Description

This is GetContainedControl, a member of class TRibbonCustomControl.

77.1.15.4.8 TRibbonCustomControl.GetGlyphWidth Method

Pascal

```
function GetGlyphWidth: Integer;
```

C++

```
__fastcall int GetGlyphWidth();
```

Description

This is GetGlyphWidth, a member of class TRibbonCustomControl.

77.1.15.4.9 TRibbonCustomControl.GetKeyTip Method

Returns current **KeyTip** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.15.4.10 TRibbonCustomControl.GetLabelWidth Method

Pascal

```
function GetLabelWidth: Integer;
```

C++

```
__fastcall int GetLabelWidth();
```

Description

This is GetLabelWidth, a member of class TRibbonCustomControl.

77.1.15.4.11 TRibbonCustomControl.HandleKeyTip Method

Indicates if command's **KeyTip** matches currently input KeyTip.

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip matches the currently input KeyTip by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.15.4.12 TRibbonCustomControl.HideKeyTip Method

Hide **KeyTip** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip**, the **KeyTip** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.1.15.4.13 TRibbonCustomControl.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.15.4.14 TRibbonCustomControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

77.1.15.4.15 TRibbonCustomControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

77.1.15.4.16 TRibbonCustomControl.SetActionClient Method

Pascal

```
procedure SetActionClient(Value: TActionClientItem); override;
```

C++

```
virtual __fastcall SetActionClient(TActionClientItem Value);
```

Description

This is SetActionClient, a member of class TRibbonCustomControl.

77.1.15.4.17 TRibbonCustomControl.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the component's boundary properties at one time. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately, but SetBounds changes all four properties at once ensuring that the control will not repaint between changes.

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

Calling SetBounds does not necessarily result in the Left, Top, Width, and Height properties changing to the specified values. Before the properties are changed, the AutoSize or Constraints property may limit the changes, and an OnCanResize (or OnConstrainedResize) event handler may change the new values. After the control's Left, Top, Width, and Height properties are changed, SetBounds generates an OnResize event.

Note: Component writers can change the Left, Top, Width, and Height properties while bypassing all resize events and constraint or autosize logic by using the UpdateBoundsRect method instead.

C++ Examples:

```
//
//This example demostrates the usage of Margins and AlignWithMargins
//properties. Note that Margins will only work when AlignWithMargins
//is set to true and Align is not alNone.
//

#define TopPanelMargin 10
#define BottomPanelMargin 20

TPanel *TopPanel, *BottomPanel;

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Create a new panel and align it to top */
    TopPanel = new TPanel(this);
    TopPanel->Parent = this;
    TopPanel->Align = alTop;
    TopPanel->Height = (ClientHeight / 2);

    /* Create a new panel and align it to the remaining client size */
    BottomPanel = new TPanel(this);
    BottomPanel->Parent = this;
    BottomPanel->Align = alClient;

    /* Set Top panel margins */
    /* Note: Margins will not have an effect if Align is alNone */
    TopPanel->Margins->SetBounds(TopPanelMargin, TopPanelMargin,
                                 TopPanelMargin, TopPanelMargin);
    TopPanel->AlignWithMargins = true;
    TopPanel->Caption = "TopPanel";

    /* Set Top Bottom margins */
    /* Note: Margins will not have an effect if Align is alNone */
    BottomPanel->Margins->SetBounds(BottomPanelMargin, BottomPanelMargin,
                                    BottomPanelMargin, BottomPanelMargin);
    BottomPanel->AlignWithMargins = true;
```

```
    BottomPanel->Caption = "BottomPanel";

    if ((TopPanel->Top != TopPanelMargin) ||
       (TopPanel->Left != TopPanelMargin) ||
       (TopPanel->Width != (ClientWidth - (TopPanelMargin * 2))) ||
       (BottomPanel->Top != (TopPanel->Height + (TopPanelMargin * 2) +
                                                BottomPanelMargin)) ||
       (BottomPanel->Left != BottomPanelMargin) ||
       (BottomPanel->Width != (ClientWidth - (BottomPanelMargin * 2))))
       {
          MessageDlg(AnsiString(
            "This should not happen! Position and size are ")+
            "calculated relative to the margins",
            mtError, TMsgDlgButtons() << mbOK, 0);
       }
}
```

Delphi Examples:

```
{
This example demostrates the usage of Margins and AlignWithMargins
properties. Note that Margins will only work when AlignWithMargins
is set to true and Align is not alNone.
}
const
  TopPanelMargin = 10;
  BottomPanelMargin = 30;

var
  TopPanel, BottomPanel : TPanel;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Create a new panel and align it to top }
  TopPanel := TPanel.Create(Self);
  TopPanel.Parent := Self;
  TopPanel.Align := alTop;
  TopPanel.Height := (ClientHeight div 2);

  { Create a new panel and align it to the remaining client size }
  BottomPanel := TPanel.Create(Self);
  BottomPanel.Parent := Self;
  BottomPanel.Align := alClient;

  { Set Top panel margins }
  { Note: Margins will not have an effect if Align is alNone }
  TopPanel.Margins.SetBounds(TopPanelMargin, TopPanelMargin,
                             TopPanelMargin, TopPanelMargin);
  TopPanel.AlignWithMargins := true;
  TopPanel.Caption:= 'TopPanel';

  { Set Bottom panel margins }
  { Note: Margins will not have an effect if Align is alNone }
  BottomPanel.Margins.SetBounds(BottomPanelMargin, BottomPanelMargin,
                                BottomPanelMargin, BottomPanelMargin);
  BottomPanel.AlignWithMargins := true;
  BottomPanel.Caption:= 'BottomPanel';

  if (TopPanel.Top <> TopPanelMargin) or
     (TopPanel.Left <> TopPanelMargin) or
     (TopPanel.Width <> (ClientWidth - (TopPanelMargin * 2))) or
     (BottomPanel.Top <> (TopPanel.Height + (TopPanelMargin * 2) +
                                              BottomPanelMargin)) or
     (BottomPanel.Left <> BottomPanelMargin) or
     (BottomPanel.Width <> (ClientWidth - (BottomPanelMargin * 2)))
     then
       MessageDlg('This should not happen! Position and size are ' +
```

```
            'calculated relative to the margins', mtError, [mbOK], 0);
  end;
```

See Also

Height

Left

Top

Width

BoundsRect

AutoSize

Constraints

OnCanResize

OnConstrainedResize

OnResize

UpdateBoundsRect

77.1.15.4.18 TRibbonCustomControl.SetContainedControl Method

Pascal

```
procedure SetContainedControl(const Value: TControl);
```

C++

```
__fastcall SetContainedControl(const TControl * Value);
```

Description

This is SetContainedControl, a member of class TRibbonCustomControl.

77.1.15.4.19 TRibbonCustomControl.SetDragMode Method

Sets the drag mode for the control.

Pascal

```
procedure SetDragMode(Value: TDragMode); override;
```

C++

```
virtual __fastcall SetDragMode(TDragMode Value);
```

Description

SetDragMode is the protected write implementation for the DragMode property. Override SetDragMode to execute additional code when the value of the DragMode property changes.

See Also

DragMode

77.1.15.4.20 TRibbonCustomControl.SetKeyTip Method

Sets **KeyTip** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

77.1.15.4.21 TRibbonCustomControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.15.4.22 TRibbonCustomControl.ShowKeyTip Method

Shows **KeyTip** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip** for the command in the position specified. Only the Application Button uses a **KeyTip** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

77.1.15.5 TRibbonCustomControl Properties

The properties of the TRibbonCustomControl class are listed here.

Public Properties

	Name	Description
	ContainedControl (see page 7073)	This is ContainedControl, a member of class TRibbonCustomControl.

77.1.15.5.1 TRibbonCustomControl.ContainedControl Property

Pascal

```
property ContainedControl: TControl;
```

C++

```
__property TControl * ContainedControl;
```

Description

This is ContainedControl, a member of class TRibbonCustomControl.

77.1.16 TRibbonCustomizeItem Class

Class Hierarchy

Pascal

```
TRibbonCustomizeItem = class(TCustomAddRemoveItem);
```

C++

```
class TRibbonCustomizeItem : public TCustomAddRemoveItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonCustomizeItem.

77.1.16.1 TRibbonCustomizeItem Members

The following tables list the members exposed by TRibbonCustomizeItem.

Protected Methods

	Name	Description
	DrawBackground (see page 7075)	Draws a background image on the control.
	DrawGlyph (see page 7075)	Draws a symbol on the control.
	RibbonStyle (see page 7075)	This is RibbonStyle, a member of class TRibbonCustomizeItem.

Public Methods

	Name	Description
	CalcBounds (see page 7074)	Calculates the size of the add/remove menu item.

77.1.16.2 TRibbonCustomizeItem Methods

The methods of the TRibbonCustomizeItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7075)	Draws a background image on the control.
	DrawGlyph (see page 7075)	Draws a symbol on the control.
	RibbonStyle (see page 7075)	This is RibbonStyle, a member of class TRibbonCustomizeItem.

Public Methods

	Name	Description
	CalcBounds (see page 7074)	Calculates the size of the add/remove menu item.

77.1.16.2.1 TRibbonCustomizeItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height

Width

TextBounds

77.1.16.2.2 TRibbonCustomizeItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.16.2.3 TRibbonCustomizeItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.16.2.4 TRibbonCustomizeItem.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

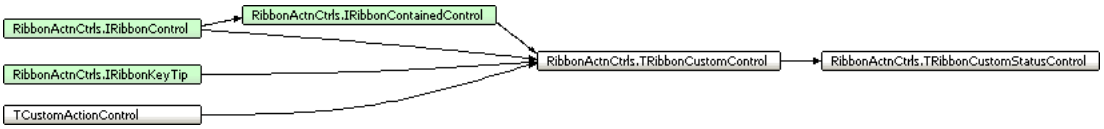
```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonCustomizeItem.

77.1.17 TRibbonCustomStatusControl Class

Class Hierarchy



Pascal

```
TRibbonCustomStatusControl = class(TRibbonCustomControl);
```

C++

```
class TRibbonCustomStatusControl : public TRibbonCustomControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonCustomStatusControl.

77.1.17.1 TRibbonCustomStatusControl Members

The following tables list the members exposed by TRibbonCustomStatusControl.

Protected Methods

	Name	Description
	CMVisiblechanged (see page 7064)	This is CMVisiblechanged, a member of class TRibbonCustomControl.
	DrawBackground (see page 7078)	Draws a background image on the control.
	GetCaptionWidth (see page 7065)	This is GetCaptionWidth, a member of class TRibbonCustomControl.
	GetGlyphWidth (see page 7065)	This is GetGlyphWidth, a member of class TRibbonCustomControl.
	GetLabelWidth (see page 7066)	This is GetLabelWidth, a member of class TRibbonCustomControl.
	Notification (see page 7067)	Responds to notifications that components are being created or destroyed.
	Paint (see page 7068)	Render control surface.
	SetActionClient (see page 7068)	This is SetActionClient, a member of class TRibbonCustomControl.
	SetDragMode (see page 7071)	Sets the drag mode for the control.
	SetParent (see page 7072)	Sets the parent of the control.

Public Methods

	Name	Description
	CalcBounds (see page 7077)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 7064)	This is CanAddToQuickAccessToolbar, a member of class TRibbonCustomControl.
	Click (see page 7064)	Executes the control's function in response to a mouse click.
	Create (see page 7062)	Creates an instance of TCustomActionControl.
	Destroy (see page 7062)	Destroys an instance of TCustomActionControl.
	GetContainedControl (see page 7275)	This is GetContainedControl, a member of class IRibbonContainedControl.
	GetContainedControl (see page 7065)	This is GetContainedControl, a member of class TRibbonCustomControl.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 7066)	Returns current **KeyTip** for command.

	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 7066)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 7067)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 7067)	Handle clicking a **KeyTip**.
V	SetBounds (see page 7068)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7275)	This is SetContainedControl, a member of class IRibbonContainedControl.
	SetContainedControl (see page 7071)	This is SetContainedControl, a member of class TRibbonCustomControl.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 7071)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 7072)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	ContainedControl (see page 7275)	This is ContainedControl, a member of class IRibbonContainedControl.
	ContainedControl (see page 7073)	This is ContainedControl, a member of class TRibbonCustomControl.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.17.2 TRibbonCustomStatusControl Methods

The methods of the TRibbonCustomStatusControl class are listed here.

Protected Methods

	Name	Description
V	DrawBackground (see page 7078)	Draws a background image on the control.

Public Methods

	Name	Description
V	CalcBounds (see page 7077)	Calculates the size of the control.

77.1.17.2.1 TRibbonCustomStatusControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.17.2.2 TRibbonCustomStatusControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.18 TRibbonDropDownButton Class

Class Hierarchy

Pascal

```
TRibbonDropDownButton = class(TRibbonBaseButtonControl);
```

C++

```
class TRibbonDropDownButton : public TRibbonBaseButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonDropDownButton.

77.1.18.1 TRibbonDropDownButton Members

The following tables list the members exposed by TRibbonDropDownButton.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7080)	This is CalcLayout, a member of class TRibbonDropDownButton.
	DoDropDown (see page 7081)	This is DoDropDown, a member of class TRibbonDropDownButton.
	DrawArrow (see page 7081)	This is DrawArrow, a member of class TRibbonDropDownButton.
	DrawBackground (see page 7081)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.

	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7081)	This is DrawShadowedText, a member of class TRibbonDropDownButton.
	DrawText (see page 7082)	Draws text on the control.
	GetArrowPosition (see page 7082)	This is GetArrowPosition, a member of class TRibbonDropDownButton.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	GetPopupClass (see page 7082)	This is GetPopupClass, a member of class TRibbonDropDownButton.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7080)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 7080)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7082)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.18.2 TRibbonDropDownButton Methods

The methods of the TRibbonDropDownButton class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7080)	This is CalcLayout, a member of class TRibbonDropDownButton.
	DoDropDown (see page 7081)	This is DoDropDown, a member of class TRibbonDropDownButton.
	DrawArrow (see page 7081)	This is DrawArrow, a member of class TRibbonDropDownButton.
	DrawBackground (see page 7081)	Draws a background image on the control.
	DrawShadowedText (see page 7081)	This is DrawShadowedText, a member of class TRibbonDropDownButton.
	DrawText (see page 7082)	Draws text on the control.
	GetArrowPosition (see page 7082)	This is GetArrowPosition, a member of class TRibbonDropDownButton.

	GetPopupClass (see page 7082)	This is GetPopupClass, a member of class TRibbonDropDownButton.

Public Methods

	Name	Description
	CalcBounds (see page 7080)	Calculates the size of the control.
	Click (see page 7080)	Executes the control's function.
	KeyTipClick (see page 7082)	Handle clicking a **KeyTip**.

77.1.18.2.1 TRibbonDropDownButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.18.2.2 TRibbonDropDownButton.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonDropDownButton.

77.1.18.2.3 TRibbonDropDownButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.18.2.4 TRibbonDropDownButton.DoDropDown Method

Pascal

```
procedure DoDropDown(const SelectFirst: Boolean); virtual;
```

C++

```
virtual __fastcall DoDropDown(const Boolean SelectFirst);
```

Description

This is DoDropDown, a member of class TRibbonDropDownButton.

77.1.18.2.5 TRibbonDropDownButton.DrawArrow Method

Pascal

```
procedure DrawArrow(Location: TPoint; Color: TColor; ShadowColor: TColor; Size: Integer);
virtual;
```

C++

```
virtual __fastcall DrawArrow(TPoint Location, TColor Color, TColor ShadowColor, int Size);
```

Description

This is DrawArrow, a member of class TRibbonDropDownButton.

77.1.18.2.6 TRibbonDropDownButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.18.2.7 TRibbonDropDownButton.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonDropDownButton.

77.1.18.2.8 TRibbonDropDownButton.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.18.2.9 TRibbonDropDownButton.GetArrowPosition Method

Pascal

```
function GetArrowPosition(const ARect: TRect): TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetArrowPosition(const TRect ARect);
```

Description

This is GetArrowPosition, a member of class TRibbonDropDownButton.

77.1.18.2.10 TRibbonDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomPopupClass; virtual;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TRibbonDropDownButton.

77.1.18.2.11 TRibbonDropDownButton.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick; override;
```

C++

```
virtual __fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.19 TRibbonGallery Class

Class Hierarchy

Pascal

```
TRibbonGallery = class(TControl);
```

C++

```
class TRibbonGallery : public TControl;
```

File

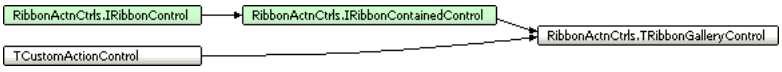
RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGallery.

77.1.20 TRibbonGalleryControl Class

Class Hierarchy



Pascal

```
TRibbonGalleryControl = class(TCustomActionControl, IRibbonContainedControl);
```

C++

```
class TRibbonGalleryControl : public TCustomActionControl, public IRibbonContainedControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGalleryControl.

77.1.20.1 TRibbonGalleryControl Members

The following tables list the members exposed by TRibbonGalleryControl.

Public Methods

	Name	Description
	CanAddToQuickAccessToolbar (see page 7084)	This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControl.

	GetContainedControl (see page 7084)	This is GetContainedControl, a member of class TRibbonGalleryControl.
	SetContainedControl (see page 7084)	This is SetContainedControl, a member of class TRibbonGalleryControl.

Public Properties

	Name	Description
	ContainedControl (see page 7085)	This is ContainedControl, a member of class TRibbonGalleryControl.
R	Gallery (see page 7085)	This is Gallery, a member of class TRibbonGalleryControl.

77.1.20.2 TRibbonGalleryControl Methods

The methods of the TRibbonGalleryControl class are listed here.

Public Methods

	Name	Description
	CanAddToQuickAccessToolbar (see page 7084)	This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControl.
	GetContainedControl (see page 7084)	This is GetContainedControl, a member of class TRibbonGalleryControl.
	SetContainedControl (see page 7084)	This is SetContainedControl, a member of class TRibbonGalleryControl.

77.1.20.2.1 TRibbonGalleryControl.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControl.

77.1.20.2.2 TRibbonGalleryControl.GetContainedControl Method

Pascal

```
function GetContainedControl: TControl;
```

C++

```
__fastcall TControl * GetContainedControl();
```

Description

This is GetContainedControl, a member of class TRibbonGalleryControl.

77.1.20.2.3 TRibbonGalleryControl.SetContainedControl Method

Pascal

```
procedure SetContainedControl(const Value: TControl);
```

C++

```
__fastcall SetContainedControl(const TControl * Value);
```

Description

This is SetContainedControl, a member of class TRibbonGalleryControl.

77.1.20.3 TRibbonGalleryControl Properties

The properties of the TRibbonGalleryControl class are listed here.

Public Properties

	Name	Description
	ContainedControl (see page 7085)	This is ContainedControl, a member of class TRibbonGalleryControl.
	Gallery (see page 7085)	This is Gallery, a member of class TRibbonGalleryControl.

77.1.20.3.1 TRibbonGalleryControl.ContainedControl Property

Pascal

```
property ContainedControl: TControl;
```

C++

```
__property TControl * ContainedControl;
```

Description

This is ContainedControl, a member of class TRibbonGalleryControl.

77.1.20.3.2 TRibbonGalleryControl.Gallery Property

Pascal

```
property Gallery: TRibbonGallery;
```

C++

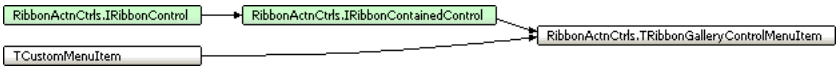
```
__property TRibbonGallery Gallery;
```

Description

This is Gallery, a member of class TRibbonGalleryControl.

77.1.21 TRibbonGalleryControlMenuItem Class

Class Hierarchy



Pascal

```
TRibbonGalleryControlMenuItem = class(TCustomMenuItem, IRibbonContainedControl);
```

C++

```
class TRibbonGalleryControlMenuItem : public TCustomMenuItem, public IRibbonContainedControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGalleryControlMenuItem.

77.1.21.1 TRibbonGalleryControlMenuItem Members

The following tables list the members exposed by TRibbonGalleryControlMenuItem.

Protected Methods

	Name	Description
	Paint (see page 7087)	Render control surface.
	SetParent (see page 7090)	Sets the parent of the control.

Public Methods

	Name	Description
	CalcBounds (see page 7086)	Calculates the size of the menu item.
	CanAddToQuickAccessToolbar (see page 7087)	This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControlMenuItem.
	GetContainedControl (see page 7087)	This is GetContainedControl, a member of class TRibbonGalleryControlMenuItem.
	SetBounds (see page 7088)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7090)	This is SetContainedControl, a member of class TRibbonGalleryControlMenuItem.

Public Properties

	Name	Description
	ContainedControl (see page 7091)	This is ContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.21.2 TRibbonGalleryControlMenuItem Methods

The methods of the TRibbonGalleryControlMenuItem class are listed here.

Protected Methods

	Name	Description
	Paint (see page 7087)	Render control surface.
	SetParent (see page 7090)	Sets the parent of the control.

Public Methods

	Name	Description
	CalcBounds (see page 7086)	Calculates the size of the menu item.
	CanAddToQuickAccessToolbar (see page 7087)	This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControlMenuItem.
	GetContainedControl (see page 7087)	This is GetContainedControl, a member of class TRibbonGalleryControlMenuItem.
	SetBounds (see page 7088)	Sets the Left, Top, Width, and Height properties all at once.
	SetContainedControl (see page 7090)	This is SetContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.21.2.1 TRibbonGalleryControlMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.21.2.2 TRibbonGalleryControlMenuItem.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class TRibbonGalleryControlMenuItem.

77.1.21.2.3 TRibbonGalleryControlMenuItem.GetContainedControl Method

Pascal

```
function GetContainedControl: TControl;
```

C++

```
__fastcall TControl * GetContainedControl();
```

Description

This is GetContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.21.2.4 TRibbonGalleryControlMenuItem.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

77.1.21.2.5 TRibbonGalleryControlMenuItem.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the component's boundary properties at one time. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately, but SetBounds changes all four properties at once ensuring that the control will not repaint between changes.

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

Calling SetBounds does not necessarily result in the Left, Top, Width, and Height properties changing to the specified values. Before the properties are changed, the AutoSize or Constraints property may limit the changes, and an OnCanResize (or OnConstrainedResize) event handler may change the new values. After the control's Left, Top, Width, and Height properties are changed, SetBounds generates an OnResize event.

Note: Component writers can change the Left, Top, Width, and Height properties while bypassing all resize events and constraint or autosize logic by using the UpdateBoundsRect method instead.

C++ Examples:

```
//
//This example demostrates the usage of Margins and AlignWithMargins
//properties. Note that Margins will only work when AlignWithMargins
//is set to true and Align is not alNone.
//

#define TopPanelMargin 10
#define BottomPanelMargin 20

TPanel *TopPanel, *BottomPanel;

void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Create a new panel and align it to top */
    TopPanel = new TPanel(this);
    TopPanel->Parent = this;
    TopPanel->Align = alTop;
    TopPanel->Height = (ClientHeight / 2);

    /* Create a new panel and align it to the remaining client size */
    BottomPanel = new TPanel(this);
    BottomPanel->Parent = this;
    BottomPanel->Align = alClient;

    /* Set Top panel margins */
    /* Note: Margins will not have an effect if Align is alNone */
    TopPanel->Margins->SetBounds(TopPanelMargin, TopPanelMargin,
                                 TopPanelMargin, TopPanelMargin);
    TopPanel->AlignWithMargins = true;
    TopPanel->Caption = "TopPanel";
```

```
    /* Set Top Bottom margins */
    /* Note: Margins will not have an effect if Align is alNone */
    BottomPanel->Margins->SetBounds(BottomPanelMargin, BottomPanelMargin,
                                    BottomPanelMargin, BottomPanelMargin);
    BottomPanel->AlignWithMargins = true;
    BottomPanel->Caption = "BottomPanel";

    if ((TopPanel->Top != TopPanelMargin) ||
       (TopPanel->Left != TopPanelMargin) ||
       (TopPanel->Width != (ClientWidth - (TopPanelMargin * 2))) ||
       (BottomPanel->Top != (TopPanel->Height + (TopPanelMargin * 2) +
                                               BottomPanelMargin)) ||
       (BottomPanel->Left != BottomPanelMargin) ||
       (BottomPanel->Width != (ClientWidth - (BottomPanelMargin * 2))))
       {
          MessageDlg(AnsiString(
            "This should not happen! Position and size are ")+
            "calculated relative to the margins",
            mtError, TMsgDlgButtons() << mbOK, 0);
       }
}
```

Delphi Examples:

```
{
This example demostrates the usage of Margins and AlignWithMargins
properties. Note that Margins will only work when AlignWithMargins
is set to true and Align is not alNone.
}
const
  TopPanelMargin = 10;
  BottomPanelMargin = 30;

var
  TopPanel, BottomPanel : TPanel;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Create a new panel and align it to top }
  TopPanel := TPanel.Create(Self);
  TopPanel.Parent := Self;
  TopPanel.Align := alTop;
  TopPanel.Height := (ClientHeight div 2);

  { Create a new panel and align it to the remaining client size }
  BottomPanel := TPanel.Create(Self);
  BottomPanel.Parent := Self;
  BottomPanel.Align := alClient;

  { Set Top panel margins }
  { Note: Margins will not have an effect if Align is alNone }
  TopPanel.Margins.SetBounds(TopPanelMargin, TopPanelMargin,
                             TopPanelMargin, TopPanelMargin);
  TopPanel.AlignWithMargins := true;
  TopPanel.Caption:= 'TopPanel';

  { Set Bottom panel margins }
  { Note: Margins will not have an effect if Align is alNone }
  BottomPanel.Margins.SetBounds(BottomPanelMargin, BottomPanelMargin,
                                BottomPanelMargin, BottomPanelMargin);
  BottomPanel.AlignWithMargins := true;
  BottomPanel.Caption:= 'BottomPanel';

  if (TopPanel.Top <> TopPanelMargin) or
     (TopPanel.Left <> TopPanelMargin) or
     (TopPanel.Width <> (ClientWidth - (TopPanelMargin * 2))) or
```

```
      (BottomPanel.Top <> (TopPanel.Height + (TopPanelMargin * 2) +
                                             BottomPanelMargin)) or
      (BottomPanel.Left <> BottomPanelMargin) or
      (BottomPanel.Width <> (ClientWidth - (BottomPanelMargin * 2)))
      then
        MessageDlg('This should not happen! Position and size are ' +
             'calculated relative to the margins', mtError, [mbOK], 0);
end;
```

See Also

Height

Left

Top

Width

BoundsRect

AutoSize

Constraints

OnCanResize

OnConstrainedResize

OnResize

UpdateBoundsRect

77.1.21.2.6 TRibbonGalleryControlMenuItem.SetContainedControl Method

Pascal

```
procedure SetContainedControl(const Value: TControl);
```

C++

```
__fastcall SetContainedControl(const TControl * Value);
```

Description

This is SetContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.21.2.7 TRibbonGalleryControlMenuItem.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's

RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.21.3 TRibbonGalleryControlMenuItem Properties

The properties of the TRibbonGalleryControlMenuItem class are listed here.

Public Properties

	Name	Description
	ContainedControl (see page 7091)	This is ContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.21.3.1 TRibbonGalleryControlMenuItem.ContainedControl Property

Pascal

```
property ContainedControl: TControl;
```

C++

```
__property TControl * ContainedControl;
```

Description

This is ContainedControl, a member of class TRibbonGalleryControlMenuItem.

77.1.22 TRibbonGalleryItem Class

Class Hierarchy

Pascal

```
TRibbonGalleryItem = class(TRibbonBaseButtonControl);
```

C++

```
class TRibbonGalleryItem : public TRibbonBaseButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGalleryItem.

77.1.22.1 TRibbonGalleryItem Members

The following tables list the members exposed by TRibbonGalleryItem.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7093)	This is CalcLayout, a member of class TRibbonGalleryItem.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7093)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.22.2 TRibbonGalleryItem Methods

The methods of the TRibbonGalleryItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7093)	This is CalcLayout, a member of class TRibbonGalleryItem.

Public Methods

	Name	Description
	CalcBounds (see page 7093)	Calculates the size of the control.

77.1.22.2.1 TRibbonGalleryItem.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.22.2.2 TRibbonGalleryItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonGalleryItem.

77.1.23 TRibbonGalleryMenuItem Class

Class Hierarchy

Pascal

```
TRibbonGalleryMenuItem = class(TCustomMenuItem);
```

C++

```
class TRibbonGalleryMenuItem : public TCustomMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGalleryMenuItem.

77.1.23.1 TRibbonGalleryMenuItem Members

The following tables list the members exposed by TRibbonGalleryMenuItem.

Protected Methods

	Name	Description
	DrawBackground (see page 7095)	Draws a background image on the control.
	DrawGlyph (see page 7095)	Draws a symbol on the control.
	DrawShadowedText (see page 7095)	This is DrawShadowedText, a member of class TRibbonGalleryMenuItem.
	DrawText (see page 7096)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7094)	Calculates the size of the menu item.

77.1.23.2 TRibbonGalleryMenuItem Methods

The methods of the TRibbonGalleryMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7095)	Draws a background image on the control.
	DrawGlyph (see page 7095)	Draws a symbol on the control.
	DrawShadowedText (see page 7095)	This is DrawShadowedText, a member of class TRibbonGalleryMenuItem.
	DrawText (see page 7096)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7094)	Calculates the size of the menu item.

77.1.23.2.1 TRibbonGalleryMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines

to ensure that the values are set properly.

See Also

SetBounds

77.1.23.2.2 TRibbonGalleryMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.23.2.3 TRibbonGalleryMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.23.2.4 TRibbonGalleryMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonGalleryMenuItem.

77.1.23.2.5 TRibbonGalleryMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

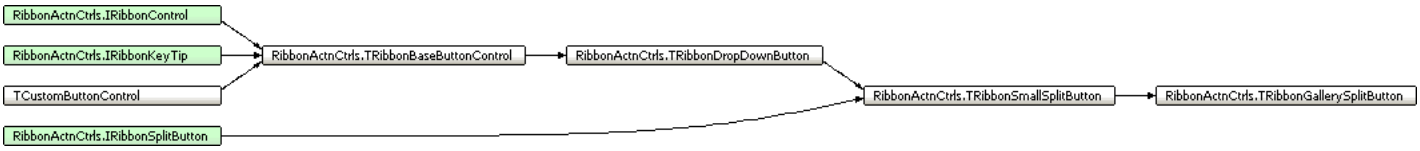
See Also

DrawBackground

DrawGlyph

77.1.24 TRibbonGallerySplitButton Class

Class Hierarchy



Pascal

```
TRibbonGallerySplitButton = class(TRibbonSmallSplitButton);
```

C++

```
class TRibbonGallerySplitButton : public TRibbonSmallSplitButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGallerySplitButton.

77.1.24.1 TRibbonGallerySplitButton Members

The following tables list the members exposed by TRibbonGallerySplitButton.

Protected Fields

	Name	Description
	FCursorInSplit (see page 7154)	This is FCursorInSplit, a member of class TRibbonSmallSplitButton.
	FSplitRect (see page 7154)	This is FSplitRect, a member of class TRibbonSmallSplitButton.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7080)	This is CalcLayout, a member of class TRibbonDropDownButton.
	CalcLayout (see page 7155)	This is CalcLayout, a member of class TRibbonSmallSplitButton.
	CMMouseLeave (see page 7156)	This is CMMouseLeave, a member of class TRibbonSmallSplitButton.
	DoDropDown (see page 7081)	This is DoDropDown, a member of class TRibbonDropDownButton.
	DrawArrow (see page 7081)	This is DrawArrow, a member of class TRibbonDropDownButton.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawBackground (see page 7081)	Draws a background image on the control.
	DrawBackground (see page 7156)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7081)	This is DrawShadowedText, a member of class TRibbonDropDownButton.
	DrawText (see page 6986)	Draws text on the control.
	DrawText (see page 7082)	Draws text on the control.
	GetArrowPosition (see page 7082)	This is GetArrowPosition, a member of class TRibbonDropDownButton.
	GetArrowPosition (see page 7157)	This is GetArrowPosition, a member of class TRibbonSmallSplitButton.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	GetPopupClass (see page 7098)	This is GetPopupClass, a member of class TRibbonGallerySplitButton.
	InSplit (see page 7157)	This is InSplit, a member of class TRibbonSmallSplitButton.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseDown (see page 7157)	OnMouseDown event dispatcher.
	MouseMove (see page 7158)	Respond to mouse moving over control area..
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CalcBounds (see page 7080)	Calculates the size of the control.
	CalcBounds (see page 7155)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Click (see page 7080)	Executes the control's function.
	Click (see page 7156)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Create (see page 7153)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.

	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
V	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
V	KeyTipClick (see page 7082)	Handle clicking a **KeyTip**.
V	KeyTipClick (see page 7157)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.
	UpdateSplitBounds (see page 7283)	This is UpdateSplitBounds, a member of class IRibbonSplitButton.
V	UpdateSplitBounds (see page 7158)	This is UpdateSplitBounds, a member of class TRibbonSmallSplitButton.

Public Properties

	Name	Description
	CursorInSplit (see page 7159)	This is CursorInSplit, a member of class TRibbonSmallSplitButton.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.24.2 TRibbonGallerySplitButton Methods

The methods of the TRibbonGallerySplitButton class are listed here.

Protected Methods

	Name	Description
V	GetPopupClass (see page 7098)	This is GetPopupClass, a member of class TRibbonGallerySplitButton.

77.1.24.2.1 TRibbonGallerySplitButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomPopupClass; override;
```

C++

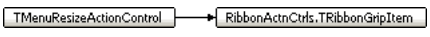
```
virtual __fastcall TCustomPopupClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TRibbonGallerySplitButton.

77.1.25 TRibbonGripItem Class

Class Hierarchy



Pascal

```
TRibbonGripItem = class(TMenuResizeActionControl);
```

C++

```
class TRibbonGripItem : public TMenuResizeActionControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonGripItem.

77.1.25.1 TRibbonGripItem Members

The following tables list the members exposed by TRibbonGripItem.

Protected Methods

	Name	Description
	Paint (see page 7100)	Render control surface.
	RibbonStyle (see page 7101)	This is RibbonStyle, a member of class TRibbonGripItem.

Public Methods

	Name	Description
	CalcBounds (see page 7100)	Calculates the size of the control.
	Create (see page 7099)	Creates an instance of TCustomActionControl.

Public Properties

	Name	Description
	GalleryResize (see page 7101)	This is GalleryResize, a member of class TRibbonGripItem.

77.1.25.2 TRibbonGripItem.Create Constructor

Creates an instance of TCustomActionControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomActionControl. However, you should never attempt to instantiate a TCustomActionControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the TCustomActionControl instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.25.3 TRibbonGripItem Methods

The methods of the TRibbonGripItem class are listed here.

Protected Methods

	Name	Description
	Paint (see page 7100)	Render control surface.
	RibbonStyle (see page 7101)	This is RibbonStyle, a member of class TRibbonGripItem.

Public Methods

	Name	Description
	CalcBounds (see page 7100)	Calculates the size of the control.

77.1.25.3.1 TRibbonGripItem.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.25.3.2 TRibbonGripItem.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

77.1.25.3.3 TRibbonGripItem.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonGripItem.

77.1.25.4 TRibbonGripItem Properties

The properties of the TRibbonGripItem class are listed here.

Public Properties

	Name	Description
	GalleryResize (see page 7101)	This is GalleryResize, a member of class TRibbonGripItem.

77.1.25.4.1 TRibbonGripItem.GalleryResize Property

Pascal

```
property GalleryResize: TGalleryResize;
```

C++

```
__property TGalleryResize GalleryResize;
```

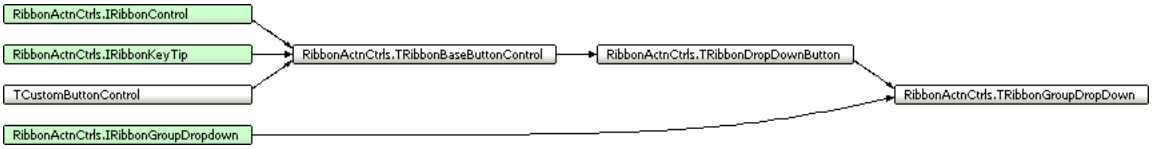
Description

This is GalleryResize, a member of class TRibbonGripItem.

77.1.26 TRibbonGroupDropDown Class

Display entire group as dropdown menu in Quick Access Toolbar.

Class Hierarchy



Pascal

```
TRibbonGroupDropDown = class(TRibbonDropDownButton, IRibbonGroupDropdown);
```

C++

```
class TRibbonGroupDropDown : public TRibbonDropDownButton, public IRibbonGroupDropdown;
```

File

RibbonActnCtrls

Description

TRibbonGroupDropDown is a control used to display an entire group as a dropdown menu in the Quick Access Toolbar.

See Also

TRibbonQuickAccessToolbar (see page 6911)

77.1.26.1 TRibbonGroupDropDown Members

The following tables list the members exposed by TRibbonGroupDropDown.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7080)	This is CalcLayout, a member of class TRibbonDropDownButton.
	DoDropDown (see page 7103)	This is DoDropDown, a member of class TRibbonGroupDropDown.
	DrawArrow (see page 7104)	This is DrawArrow, a member of class TRibbonGroupDropDown.
	DrawBackground (see page 7104)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7081)	This is DrawShadowedText, a member of class TRibbonDropDownButton.
	DrawText (see page 6986)	Draws text on the control.
	DrawText (see page 7082)	Draws text on the control.
	GetArrowPosition (see page 7082)	This is GetArrowPosition, a member of class TRibbonDropDownButton.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	GetPopupClass (see page 7082)	This is GetPopupClass, a member of class TRibbonDropDownButton.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CalcBounds (see page 7080)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Click (see page 7080)	Executes the control's function.
	CloseUp (see page 7103)	This is CloseUp, a member of class TRibbonGroupDropDown.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	GetParentControl (see page 7104)	This is GetParentControl, a member of class TRibbonGroupDropDown.
	GetTopPosition (see page 7104)	This is GetTopPosition, a member of class TRibbonGroupDropDown.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).

	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
V	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
V	KeyTipClick (see page 7082)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	Group (see page 7105)	This is Group, a member of class TRibbonGroupDropDown.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.26.2 TRibbonGroupDropDown Methods

The methods of the TRibbonGroupDropDown class are listed here.

Protected Methods

	Name	Description
V	DoDropDown (see page 7103)	This is DoDropDown, a member of class TRibbonGroupDropDown.
V	DrawArrow (see page 7104)	This is DrawArrow, a member of class TRibbonGroupDropDown.
V	DrawBackground (see page 7104)	Draws a background image on the control.

Public Methods

	Name	Description
	CloseUp (see page 7103)	This is CloseUp, a member of class TRibbonGroupDropDown.
	GetParentControl (see page 7104)	This is GetParentControl, a member of class TRibbonGroupDropDown.
	GetTopPosition (see page 7104)	This is GetTopPosition, a member of class TRibbonGroupDropDown.

77.1.26.2.1 TRibbonGroupDropDown.CloseUp Method

Pascal

```
procedure CloseUp;
```

C++

```
__fastcall CloseUp();
```

Description

This is CloseUp, a member of class TRibbonGroupDropDown.

77.1.26.2.2 TRibbonGroupDropDown.DoDropDown Method

Pascal

```
procedure DoDropDown(const SelectFirst: Boolean); override;
```

C++

```
virtual __fastcall DoDropDown(const Boolean SelectFirst);
```

Description

This is DoDropDown, a member of class TRibbonGroupDropDown.

77.1.26.2.3 TRibbonGroupDropDown.DrawArrow Method

Pascal

```
procedure DrawArrow(Location: TPoint; Color: TColor; ShadowColor: TColor; Size: Integer);
override;
```

C++

```
virtual __fastcall DrawArrow(TPoint Location, TColor Color, TColor ShadowColor, int Size);
```

Description

This is DrawArrow, a member of class TRibbonGroupDropDown.

77.1.26.2.4 TRibbonGroupDropDown.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.26.2.5 TRibbonGroupDropDown.GetParentControl Method

Pascal

```
function GetParentControl: TWinControl;
```

C++

```
__fastcall TWinControl * GetParentControl();
```

Description

This is GetParentControl, a member of class TRibbonGroupDropDown.

77.1.26.2.6 TRibbonGroupDropDown.GetTopPosition Method

Pascal

```
function GetTopPosition: Integer;
```

C++

```
__fastcall int GetTopPosition();
```

Description

This is GetTopPosition, a member of class TRibbonGroupDropDown.

77.1.26.3 TRibbonGroupDropDown Properties

The properties of the TRibbonGroupDropDown class are listed here.

Public Properties

	Name	Description
	Group (see page 7105)	This is Group, a member of class TRibbonGroupDropDown.

77.1.26.3.1 TRibbonGroupDropDown.Group Property

Pascal

```
property Group: TControl;
```

C++

```
__property TControl * Group;
```

Description

This is Group, a member of class TRibbonGroupDropDown.

77.1.27 TRibbonLargeButtonControl Class

Class Hierarchy

Pascal

```
TRibbonLargeButtonControl = class(TRibbonBaseButtonControl);
```

C++

```
class TRibbonLargeButtonControl : public TRibbonBaseButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonLargeButtonControl.

77.1.27.1 TRibbonLargeButtonControl Members

The following tables list the members exposed by TRibbonLargeButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7107)	This is CalcLayout, a member of class TRibbonLargeButtonControl.
	DrawBackground (see page 7107)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7108)	This is DrawShadowedText, a member of class TRibbonLargeButtonControl.
	DrawText (see page 7108)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7107)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.27.2 TRibbonLargeButtonControl Methods

The methods of the TRibbonLargeButtonControl class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7107)	This is CalcLayout, a member of class TRibbonLargeButtonControl.

	DrawBackground (see page 7107)	Draws a background image on the control.
	DrawShadowedText (see page 7108)	This is DrawShadowedText, a member of class TRibbonLargeButtonControl.
	DrawText (see page 7108)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7107)	Calculates the size of the control.

77.1.27.2.1 TRibbonLargeButtonControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.27.2.2 TRibbonLargeButtonControl.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonLargeButtonControl.

77.1.27.2.3 TRibbonLargeButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.27.2.4 TRibbonLargeButtonControl.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonLargeButtonControl.

77.1.27.2.5 TRibbonLargeButtonControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

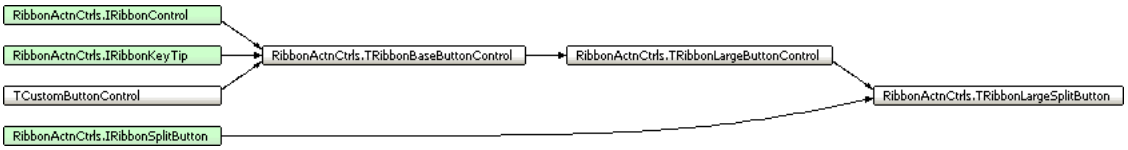
See Also

DrawBackground

DrawGlyph

77.1.28 TRibbonLargeSplitButton Class

Class Hierarchy



Pascal

```
TRibbonLargeSplitButton = class(TRibbonLargeButtonControl, IRibbonSplitButton);
```

C++

```
class TRibbonLargeSplitButton : public TRibbonLargeButtonControl, public IRibbonSplitButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonLargeSplitButton.

77.1.28.1 TRibbonLargeSplitButton Members

The following tables list the members exposed by TRibbonLargeSplitButton.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7111)	This is CalcLayout, a member of class TRibbonLargeSplitButton.
	CMMouseleave (see page 7111)	This is CMMouseleave, a member of class TRibbonLargeSplitButton.
	DoSplitButtonClick (see page 7111)	This is DoSplitButtonClick, a member of class TRibbonLargeSplitButton.
	DrawArrow (see page 7112)	This is DrawArrow, a member of class TRibbonLargeSplitButton.
	DrawBackground (see page 7112)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7108)	This is DrawShadowedText, a member of class TRibbonLargeButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	DrawText (see page 7108)	Draws text on the control.
	GetArrowPosition (see page 7112)	This is GetArrowPosition, a member of class TRibbonLargeSplitButton.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	InSplit (see page 7112)	This is InSplit, a member of class TRibbonLargeSplitButton.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseDown (see page 7113)	OnMouseDown event dispatcher.
	MouseMove (see page 7113)	Respond to mouse moving over control area..
	MouseUp (see page 7114)	OnMouseUp event dispatcher.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7110)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 7111)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7113)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.

	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.
	UpdateSplitBounds (see page 7114)	This is UpdateSplitBounds, a member of class TRibbonLargeSplitButton.

Public Properties

	Name	Description
	CursorInSplit (see page 7115)	This is CursorInSplit, a member of class TRibbonLargeSplitButton.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.28.2 TRibbonLargeSplitButton Methods

The methods of the TRibbonLargeSplitButton class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7111)	This is CalcLayout, a member of class TRibbonLargeSplitButton.
	CMMouseleave (see page 7111)	This is CMMouseleave, a member of class TRibbonLargeSplitButton.
	DoSplitButtonClick (see page 7111)	This is DoSplitButtonClick, a member of class TRibbonLargeSplitButton.
	DrawArrow (see page 7112)	This is DrawArrow, a member of class TRibbonLargeSplitButton.
	DrawBackground (see page 7112)	Draws a background image on the control.
	GetArrowPosition (see page 7112)	This is GetArrowPosition, a member of class TRibbonLargeSplitButton.
	InSplit (see page 7112)	This is InSplit, a member of class TRibbonLargeSplitButton.
	MouseDown (see page 7113)	OnMouseDown event dispatcher.
	MouseMove (see page 7113)	Respond to mouse moving over control area..
	MouseUp (see page 7114)	OnMouseUp event dispatcher.

Public Methods

	Name	Description
	CalcBounds (see page 7110)	Calculates the size of the control.
	Click (see page 7111)	Executes the control's function.
	KeyTipClick (see page 7113)	Handle clicking a **KeyTip**.
	UpdateSplitBounds (see page 7114)	This is UpdateSplitBounds, a member of class TRibbonLargeSplitButton.

77.1.28.2.1 TRibbonLargeSplitButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.28.2.2 TRibbonLargeSplitButton.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonLargeSplitButton.

77.1.28.2.3 TRibbonLargeSplitButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.28.2.4 TRibbonLargeSplitButton.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TRibbonLargeSplitButton.

77.1.28.2.5 TRibbonLargeSplitButton.DoSplitButtonClick Method

Pascal

```
procedure DoSplitButtonClick(const SelectFirst: Boolean); virtual;
```

C++

```
virtual __fastcall DoSplitButtonClick(const Boolean SelectFirst);
```

Description

This is DoSplitButtonClick, a member of class TRibbonLargeSplitButton.

77.1.28.2.6 TRibbonLargeSplitButton.DrawArrow Method

Pascal

```
procedure DrawArrow(Location: TPoint; Color: TColor; ShadowColor: TColor; Size: Integer);
virtual;
```

C++

```
virtual __fastcall DrawArrow(TPoint Location, TColor Color, TColor ShadowColor, int Size);
```

Description

This is DrawArrow, a member of class TRibbonLargeSplitButton.

77.1.28.2.7 TRibbonLargeSplitButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.28.2.8 TRibbonLargeSplitButton.GetArrowPosition Method

Pascal

```
function GetArrowPosition(const ARect: TRect): TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetArrowPosition(const TRect ARect);
```

Description

This is GetArrowPosition, a member of class TRibbonLargeSplitButton.

77.1.28.2.9 TRibbonLargeSplitButton.InSplit Method

Pascal

```
function InSplit(X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean InSplit(int X, int Y);
```

Description

This is InSplit, a member of class TRibbonLargeSplitButton.

77.1.28.2.10 TRibbonLargeSplitButton.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick; override;
```

C++

```
virtual __fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.28.2.11 TRibbonLargeSplitButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

77.1.28.2.12 TRibbonLargeSplitButton.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

77.1.28.2.13 TRibbonLargeSplitButton.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

77.1.28.2.14 TRibbonLargeSplitButton.UpdateSplitBounds Method

Pascal

```
procedure UpdateSplitBounds; virtual;
```

C++

```
virtual __fastcall UpdateSplitBounds();
```

Description

This is UpdateSplitBounds, a member of class TRibbonLargeSplitButton.

77.1.28.3 TRibbonLargeSplitButton Properties

The properties of the TRibbonLargeSplitButton class are listed here.

Public Properties

	Name	Description
	CursorInSplit (see page 7115)	This is CursorInSplit, a member of class TRibbonLargeSplitButton.

77.1.28.3.1 TRibbonLargeSplitButton.CursorInSplit Property

Pascal

```
property CursorInSplit: Boolean;
```

C++

```
__property Boolean CursorInSplit;
```

Description

This is CursorInSplit, a member of class TRibbonLargeSplitButton.

77.1.29 TRibbonMainMenuItem Class

Class Hierarchy

Pascal

```
TRibbonMainMenuItem = class(TRibbonApplicationMenuItem);
```

C++

```
class TRibbonMainMenuItem : public TRibbonApplicationMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonMainMenuItem.

77.1.29.1 TRibbonMainMenuItem Members

The following tables list the members exposed by TRibbonMainMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7121)	Draws a background image on the control.

	DrawGlyph (see page 6978)	Draws a symbol on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 6979)	This is DrawSelectedBackground, a member of class TRibbonApplicationMenuItem.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 6979)	This is DrawSeparator, a member of class TRibbonApplicationMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7116)	This is DrawShadowedText, a member of class TRibbonMainMenuItem.
	DrawSubMenuGlyph (see page 6979)	This is DrawSubMenuGlyph, a member of class TRibbonApplicationMenuItem.
	DrawText (see page 7116)	Draws text on the control.
	GetImageSize (see page 6980)	This is GetImageSize, a member of class TRibbonApplicationMenuItem.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 6978)	Calculates the size of the menu item.
	CalcBounds (see page 7120)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 6977)	Creates an instance of TCustomMenuItem.
	Create (see page 7119)	Creates an instance of TCustomMenuItem.

77.1.29.2 TRibbonMainMenuItem Methods

The methods of the TRibbonMainMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawShadowedText (see page 7116)	This is DrawShadowedText, a member of class TRibbonMainMenuItem.
	DrawText (see page 7116)	Draws text on the control.

77.1.29.2.1 TRibbonMainMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonMainMenuItem.

77.1.29.2.2 TRibbonMainMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.30 TRibbonMainMenuSeparator Class

Class Hierarchy

Pascal

```
TRibbonMainMenuSeparator = class(TRibbonSeparator);
```

C++

```
class TRibbonMainMenuSeparator : public TRibbonSeparator;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonMainMenuSeparator.

77.1.30.1 TRibbonMainMenuSeparator Members

The following tables list the members exposed by TRibbonMainMenuSeparator.

Protected Methods

	Name	Description
	DrawBackground (see page 7148)	Draws a background image on the control.
	DrawGlyph (see page 7149)	Draws a symbol on the control.
	DrawShadowedText (see page 7149)	This is DrawShadowedText, a member of class TRibbonSeparator.
	DrawText (see page 7149)	Draws text on the control.
	RibbonStyle (see page 7150)	This is RibbonStyle, a member of class TRibbonSeparator.

Public Methods

	Name	Description
	CalcBounds (see page 7118)	Calculates the size of the control.
	Create (see page 7147)	Creates an instance of TCustomActionControl.

77.1.30.2 TRibbonMainMenuSeparator Methods

The methods of the TRibbonMainMenuSeparator class are listed here.

Public Methods

	Name	Description
	CalcBounds (see page 7118)	Calculates the size of the control.

77.1.30.2.1 TRibbonMainMenuSeparator.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.31 TRibbonMenuItem Class

Class Hierarchy

Pascal

```
TRibbonMenuItem = class(TCustomMenuItem);
```

C++

```
class TRibbonMenuItem : public TCustomMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonMenuItem.

77.1.31.1 TRibbonMenuItem Members

The following tables list the members exposed by TRibbonMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7121)	Draws a background image on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawText (see page 7122)	Draws text on the control.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7120)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 7119)	Creates an instance of TCustomMenuItem.

77.1.31.2 TRibbonMenuItem.Create Constructor

Creates an instance of TCustomMenuItem.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuItem. However, you should never attempt to instantiate a TCustomMenuItem. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

77.1.31.3 TRibbonMenuItem Methods

The methods of the TRibbonMenuItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7121)	Draws a background image on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawText (see page 7122)	Draws text on the control.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7120)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.

77.1.31.3.1 TRibbonMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.31.3.2 TRibbonMenuItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonMenuItem.

77.1.31.3.3 TRibbonMenuItem.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl does little except bring the form containing the item into focus, and prevent the inherited method from being called. The reason for this lack of action is that at runtime, no action should be taken until the primary mouse button is pressed and released, which makes the OnClick event handler responsible for taking action. Click takes place only when the mouse button is first pressed, prior to release.

See Also

OnClick

77.1.31.3.4 TRibbonMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.31.3.5 TRibbonMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.31.3.6 TRibbonMenuItem.DrawSelectedBackground Method

Pascal

```
procedure DrawSelectedBackground(ARect: TRect); virtual;
```

C++

```
virtual __fastcall DrawSelectedBackground(TRect ARect);
```

Description

This is DrawSelectedBackground, a member of class TRibbonMenuItem.

77.1.31.3.7 TRibbonMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TRibbonMenuItem.

77.1.31.3.8 TRibbonMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonMenuItem.

77.1.31.3.9 TRibbonMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.31.3.10 TRibbonMenuItem.GetShowShortCut Method

Pascal

```
function GetShowShortCut: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetShowShortCut();
```

Description

This is GetShowShortCut, a member of class TRibbonMenuItem.

77.1.31.3.11 TRibbonMenuItem.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonMenuItem.

77.1.31.4 TRibbonMenuItem Properties

The properties of the TRibbonMenuItem class are listed here.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

77.1.31.4.1 TRibbonMenuItem.DrawBanner Property

Pascal

```
property DrawBanner: Boolean;
```

C++

```
__property Boolean DrawBanner;
```

Description

This is DrawBanner, a member of class TRibbonMenuItem.

77.1.32 TRibbonMenuSeparatorItem Class

Class Hierarchy

Pascal

```
TRibbonMenuSeparatorItem = class(TRibbonMenuItem);
```

C++

```
class TRibbonMenuSeparatorItem : public TRibbonMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonMenuSeparatorItem.

77.1.32.1 TRibbonMenuSeparatorItem Members

The following tables list the members exposed by TRibbonMenuSeparatorItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7121)	Draws a background image on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7125)	This is DrawSeparator, a member of class TRibbonMenuSeparatorItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawText (see page 7122)	Draws text on the control.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7124)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 7119)	Creates an instance of TCustomMenuItem.

77.1.32.2 TRibbonMenuSeparatorItem Methods

The methods of the TRibbonMenuSeparatorItem class are listed here.

Protected Methods

	Name	Description
	DrawSeparator (see page 7125)	This is DrawSeparator, a member of class TRibbonMenuSeparatorItem.

Public Methods

	Name	Description
	CalcBounds (see page 7124)	Calculates the size of the menu item.

77.1.32.2.1 TRibbonMenuSeparatorItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.32.2.2 TRibbonMenuSeparatorItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

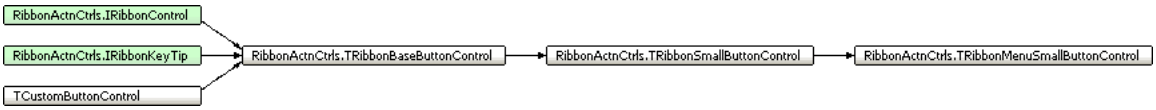
```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TRibbonMenuSeparatorItem.

77.1.33 TRibbonMenuSmallButtonControl Class

Class Hierarchy



Pascal

```
TRibbonMenuSmallButtonControl = class(TRibbonSmallButtonControl);
```

C++

```
class TRibbonMenuSmallButtonControl : public TRibbonSmallButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonMenuSmallButtonControl.

77.1.33.1 TRibbonMenuSmallButtonControl Members

The following tables list the members exposed by TRibbonMenuSmallButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.

	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseUp (see page 7127)	OnMouseUp event dispatcher.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CalcBounds (see page 7151)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 7126)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.33.2 TRibbonMenuSmallButtonControl.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.33.3 TRibbonMenuSmallButtonControl Methods

The methods of the TRibbonMenuSmallButtonControl class are listed here.

Protected Methods

	Name	Description
	MouseUp (see page 7127)	OnMouseUp event dispatcher.

77.1.33.3.1 TRibbonMenuSmallButtonControl.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

77.1.34 TRibbonOptionButtonControl Class

Class Hierarchy

Pascal

```
TRibbonOptionButtonControl = class(TRibbonMenuSmallButtonControl);
```

C++

```
class TRibbonOptionButtonControl : public TRibbonMenuSmallButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonOptionButtonControl.

77.1.34.1 TRibbonOptionButtonControl Members

The following tables list the members exposed by TRibbonOptionButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 7129)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseUp (see page 7127)	OnMouseUp event dispatcher.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CalcBounds (see page 7151)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.

	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Create (see page 7126)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.34.2 TRibbonOptionButtonControl Methods

The methods of the TRibbonOptionButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7129)	Draws a background image on the control.

77.1.34.2.1 TRibbonOptionButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.35 TRibbonQATScrollButton Class

Class Hierarchy

Pascal

```
TRibbonQATScrollButton = class(TCustomToolScrollBtn);
```

C++

```
class TRibbonQATScrollButton : public TCustomToolScrollBtn;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonQATScrollButton.

77.1.35.1 TRibbonQATScrollButton Members

The following tables list the members exposed by TRibbonQATScrollButton.

Protected Methods

	Name	Description
	CMMouseEnter (see page 7132)	This is CMMouseEnter, a member of class TRibbonQATScrollButton.
	CMMouseLeave (see page 7132)	This is CMMouseLeave, a member of class TRibbonQATScrollButton.
	DrawArrows (see page 7132)	This is DrawArrows, a member of class TRibbonQATScrollButton.
	DrawBackground (see page 7132)	Draws a background image on the control.
	MouseMove (see page 7133)	Respond to mouse moving over control area..
	RibbonStyle (see page 7133)	This is RibbonStyle, a member of class TRibbonQATScrollButton.

Protected Properties

	Name	Description
	CursorInBtn (see page 7134)	This is CursorInBtn, a member of class TRibbonQATScrollButton.

Public Methods

	Name	Description
	CalcBounds (see page 7131)	Calculates the size of the control.
	Click (see page 7131)	Executes the control's function.

Public Properties

	Name	Description
	DrawDown (see page 7134)	This is DrawDown, a member of class TRibbonQATScrollButton.
	ScrollMode (see page 7134)	This is ScrollMode, a member of class TRibbonQATScrollButton.

77.1.35.2 TRibbonQATScrollButton Methods

The methods of the TRibbonQATScrollButton class are listed here.

Protected Methods

	Name	Description
	CMMouseEnter (see page 7132)	This is CMMouseEnter, a member of class TRibbonQATScrollButton.
	CMMouseLeave (see page 7132)	This is CMMouseLeave, a member of class TRibbonQATScrollButton.
	DrawArrows (see page 7132)	This is DrawArrows, a member of class TRibbonQATScrollButton.
	DrawBackground (see page 7132)	Draws a background image on the control.
	MouseMove (see page 7133)	Respond to mouse moving over control area..
	RibbonStyle (see page 7133)	This is RibbonStyle, a member of class TRibbonQATScrollButton.

Public Methods

	Name	Description
	CalcBounds (see page 7131)	Calculates the size of the control.
	Click (see page 7131)	Executes the control's function.

77.1.35.2.1 TRibbonQATScrollButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.35.2.2 TRibbonQATScrollButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.35.2.3 TRibbonQATScrollButton.CMMouseEnter Method

Pascal

```
procedure CMMouseEnter(var Message: TMessage); override;
```

C++

```
virtual __fastcall CMMouseEnter(TMessage Message);
```

Description

This is CMMouseEnter, a member of class TRibbonQATScrollButton.

77.1.35.2.4 TRibbonQATScrollButton.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); override;
```

C++

```
virtual __fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TRibbonQATScrollButton.

77.1.35.2.5 TRibbonQATScrollButton.DrawArrows Method

Pascal

```
procedure DrawArrows; override;
```

C++

```
virtual __fastcall DrawArrows();
```

Description

This is DrawArrows, a member of class TRibbonQATScrollButton.

77.1.35.2.6 TRibbonQATScrollButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.35.2.7 TRibbonQATScrollButton.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

77.1.35.2.8 TRibbonQATScrollButton.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonQATScrollButton.

77.1.35.3 TRibbonQATScrollButton Properties

The properties of the TRibbonQATScrollButton class are listed here.

Protected Properties

	Name	Description
	CursorInBtn (see page 7134)	This is CursorInBtn, a member of class TRibbonQATScrollButton.

Public Properties

	Name	Description
	DrawDown (see page 7134)	This is DrawDown, a member of class TRibbonQATScrollButton.
	ScrollMode (see page 7134)	This is ScrollMode, a member of class TRibbonQATScrollButton.

77.1.35.3.1 TRibbonQATScrollButton.CursorInBtn Property

Pascal

```
property CursorInBtn: Boolean;
```

C++

```
__property Boolean CursorInBtn;
```

Description

This is CursorInBtn, a member of class TRibbonQATScrollButton.

77.1.35.3.2 TRibbonQATScrollButton.DrawDown Property

Pascal

```
property DrawDown: Boolean;
```

C++

```
__property Boolean DrawDown;
```

Description

This is DrawDown, a member of class TRibbonQATScrollButton.

77.1.35.3.3 TRibbonQATScrollButton.ScrollMode Property

Pascal

```
property ScrollMode: TQATScrollMode;
```

C++

```
__property TQATScrollMode ScrollMode;
```

Description

This is ScrollMode, a member of class TRibbonQATScrollButton.

77.1.36 TRibbonRadioButtonControl Class

Class Hierarchy

Pascal

```
TRibbonRadioButtonControl = class(TRibbonSmallButtonControl);
```

C++

```
class TRibbonRadioButtonControl : public TRibbonSmallButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonRadioButtonControl.

77.1.36.1 TRibbonRadioButtonControl Members

The following tables list the members exposed by TRibbonRadioButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 7136)	Draws a background image on the control.
	DrawGlyph (see page 7136)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7136)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.36.2 TRibbonRadioButtonControl Methods

The methods of the TRibbonRadioButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7136)	Draws a background image on the control.
	DrawGlyph (see page 7136)	Draws a symbol on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7136)	Calculates the size of the control.

77.1.36.2.1 TRibbonRadioButtonControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.36.2.2 TRibbonRadioButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.36.2.3 TRibbonRadioButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.37 TRibbonRecentButtonControl Class

Class Hierarchy

Pascal

```
TRibbonRecentButtonControl = class(TRibbonMenuSmallButtonControl);
```

C++

```
class TRibbonRecentButtonControl : public TRibbonMenuSmallButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonRecentButtonControl.

77.1.37.1 TRibbonRecentButtonControl Members

The following tables list the members exposed by TRibbonRecentButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 7138)	This is AllowHotKeys, a member of class TRibbonRecentButtonControl.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseUp (see page 7127)	OnMouseUp event dispatcher.

	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 6983)	Calculates the size of the control.
	CalcBounds (see page 7151)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Create (see page 7126)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.37.2 TRibbonRecentButtonControl Methods

The methods of the TRibbonRecentButtonControl class are listed here.

Protected Methods

	Name	Description
	AllowHotKeys (see page 7138)	This is AllowHotKeys, a member of class TRibbonRecentButtonControl.

77.1.37.2.1 TRibbonRecentButtonControl.AllowHotKeys Method

Pascal

```
function AllowHotKeys: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllowHotKeys();
```

Description

This is AllowHotKeys, a member of class TRibbonRecentButtonControl.

77.1.38 TRibbonRichContentItem Class

Class Hierarchy

Pascal

```
TRibbonRichContentItem = class(TRibbonMenuItem);
```

C++

```
class TRibbonRichContentItem : public TRibbonMenuItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonRichContentItem.

77.1.38.1 TRibbonRichContentItem Members

The following tables list the members exposed by TRibbonRichContentItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	CanCalculateHeight (see page 7141)	This is CanCalculateHeight, a member of class TRibbonRichContentItem.
	DrawBackground (see page 7141)	Draws a background image on the control.
	DrawGlyph (see page 7141)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7142)	This is DrawShadowedText, a member of class TRibbonRichContentItem.
	DrawText (see page 7142)	Draws text on the control.
	GetItemHeight (see page 7142)	This is GetItemHeight, a member of class TRibbonRichContentItem.
	GetItemWidth (see page 7143)	This is GetItemWidth, a member of class TRibbonRichContentItem.
	GetMenuProps (see page 7143)	This is GetMenuProps, a member of class TRibbonRichContentItem.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7140)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 7140)	Creates an instance of TCustomMenuItem.

77.1.38.2 TRibbonRichContentItem.Create Constructor

Creates an instance of TCustomMenuItem.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuItem. However, you should never attempt to instantiate a TCustomMenuItem. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

77.1.38.3 TRibbonRichContentItem Methods

The methods of the TRibbonRichContentItem class are listed here.

Protected Methods

	Name	Description
	CanCalculateHeight (see page 7141)	This is CanCalculateHeight, a member of class TRibbonRichContentItem.
	DrawBackground (see page 7141)	Draws a background image on the control.
	DrawGlyph (see page 7141)	Draws a symbol on the control.
	DrawShadowedText (see page 7142)	This is DrawShadowedText, a member of class TRibbonRichContentItem.
	DrawText (see page 7142)	Draws text on the control.
	GetItemHeight (see page 7142)	This is GetItemHeight, a member of class TRibbonRichContentItem.
	GetItemWidth (see page 7143)	This is GetItemWidth, a member of class TRibbonRichContentItem.
	GetMenuProps (see page 7143)	This is GetMenuProps, a member of class TRibbonRichContentItem.

Public Methods

	Name	Description
	CalcBounds (see page 7140)	Calculates the size of the menu item.

77.1.38.3.1 TRibbonRichContentItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.38.3.2 TRibbonRichContentItem.CanCalculateHeight Method

Pascal

```
function CanCalculateHeight: Boolean;
```

C++

```
__fastcall Boolean CanCalculateHeight();
```

Description

This is CanCalculateHeight, a member of class TRibbonRichContentItem.

77.1.38.3.3 TRibbonRichContentItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.38.3.4 TRibbonRichContentItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.38.3.5 TRibbonRichContentItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonRichContentItem.

77.1.38.3.6 TRibbonRichContentItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.38.3.7 TRibbonRichContentItem.GetItemHeight Method

Pascal

```
function GetItemHeight: Integer;
```

C++

```
__fastcall int GetItemHeight();
```

Description

This is GetItemHeight, a member of class TRibbonRichContentItem.

77.1.38.3.8 TRibbonRichContentItem.GetItemWidth Method

Pascal

```
function GetItemWidth: Integer;
```

C++

```
__fastcall int GetItemWidth();
```

Description

This is GetItemWidth, a member of class TRibbonRichContentItem.

77.1.38.3.9 TRibbonRichContentItem.GetMenuProps Method

Pascal

```
function GetMenuProps: TMenuProperties;
```

C++

```
__fastcall TMenuProperties GetMenuProps();
```

Description

This is GetMenuProps, a member of class TRibbonRichContentItem.

77.1.39 TRibbonRichContentMainMenuItem Class

Class Hierarchy

Pascal

```
TRibbonRichContentMainMenuItem = class(TRibbonRichContentItem);
```

C++

```
class TRibbonRichContentMainMenuItem : public TRibbonRichContentItem;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonRichContentMainMenuItem.

77.1.39.1 TRibbonRichContentMainMenuItem Members

The following tables list the members exposed by TRibbonRichContentMainMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7145)	This is CalcLayout, a member of class TRibbonRichContentMainMenuItem.
	CanCalculateHeight (see page 7141)	This is CanCalculateHeight, a member of class TRibbonRichContentItem.
	DrawBackground (see page 7121)	Draws a background image on the control.

	DrawBackground (see page 7141)	Draws a background image on the control.
	DrawGlyph (see page 7145)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7121)	This is DrawSelectedBackground, a member of class TRibbonMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 7142)	This is DrawShadowedText, a member of class TRibbonRichContentItem.
	DrawText (see page 7146)	Draws text on the control.
	GetImageSize (see page 7146)	This is GetImageSize, a member of class TRibbonRichContentMainMenuItem.
	GetItemHeight (see page 7142)	This is GetItemHeight, a member of class TRibbonRichContentItem.
	GetItemWidth (see page 7143)	This is GetItemWidth, a member of class TRibbonRichContentItem.
	GetMenuProps (see page 7143)	This is GetMenuProps, a member of class TRibbonRichContentItem.
	GetShowShortCut (see page 7146)	This is GetShowShortCut, a member of class TRibbonRichContentMainMenuItem.
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7145)	Calculates the size of the menu item.
	Click (see page 7120)	Executes the control's function.
	Create (see page 7144)	Creates an instance of TCustomMenuItem.

77.1.39.2 TRibbonRichContentMainMenuItem.Create Constructor

Creates an instance of TCustomMenuItem.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuItem. However, you should never attempt to instantiate a TCustomMenuItem. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

77.1.39.3 TRibbonRichContentMainMenuItem Methods

The methods of the TRibbonRichContentMainMenuItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7145)	This is CalcLayout, a member of class TRibbonRichContentMainMenuItem.
	DrawGlyph (see page 7145)	Draws a symbol on the control.
	DrawText (see page 7146)	Draws text on the control.
	GetImageSize (see page 7146)	This is GetImageSize, a member of class TRibbonRichContentMainMenuItem.
	GetShowShortCut (see page 7146)	This is GetShowShortCut, a member of class TRibbonRichContentMainMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7145)	Calculates the size of the menu item.

77.1.39.3.1 TRibbonRichContentMainMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.39.3.2 TRibbonRichContentMainMenuItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonRichContentMainMenuItem.

77.1.39.3.3 TRibbonRichContentMainMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.39.3.4 TRibbonRichContentMainMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.39.3.5 TRibbonRichContentMainMenuItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TRibbonRichContentMainMenuItem.

77.1.39.3.6 TRibbonRichContentMainMenuItem.GetShowShortCut Method

Pascal

```
function GetShowShortCut: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetShowShortCut();
```

Description

This is GetShowShortCut, a member of class TRibbonRichContentMainMenuItem.

77.1.40 TRibbonSeparator Class

Class Hierarchy

Pascal

```
TRibbonSeparator = class(TCustomActionControl);
```

C++

```
class TRibbonSeparator : public TCustomActionControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSeparator.

77.1.40.1 TRibbonSeparator Members

The following tables list the members exposed by TRibbonSeparator.

Protected Methods

	Name	Description
	DrawBackground (see page 7148)	Draws a background image on the control.
	DrawGlyph (see page 7149)	Draws a symbol on the control.
	DrawShadowedText (see page 7149)	This is DrawShadowedText, a member of class TRibbonSeparator.
	DrawText (see page 7149)	Draws text on the control.
	RibbonStyle (see page 7150)	This is RibbonStyle, a member of class TRibbonSeparator.

Public Methods

	Name	Description
	CalcBounds (see page 7148)	Calculates the size of the control.
	Create (see page 7147)	Creates an instance of TCustomActionControl.

77.1.40.2 TRibbonSeparator.Create Constructor

Creates an instance of TCustomActionControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomActionControl. However, you should never attempt to instantiate a TCustomActionControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the TCustomActionControl instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.40.3 TRibbonSeparator Methods

The methods of the TRibbonSeparator class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7148)	Draws a background image on the control.
	DrawGlyph (see page 7149)	Draws a symbol on the control.
	DrawShadowedText (see page 7149)	This is DrawShadowedText, a member of class TRibbonSeparator.
	DrawText (see page 7149)	Draws text on the control.
	RibbonStyle (see page 7150)	This is RibbonStyle, a member of class TRibbonSeparator.

Public Methods

	Name	Description
	CalcBounds (see page 7148)	Calculates the size of the control.

77.1.40.3.1 TRibbonSeparator.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.40.3.2 TRibbonSeparator.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.40.3.3 TRibbonSeparator.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

77.1.40.3.4 TRibbonSeparator.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TRibbonSeparator.

77.1.40.3.5 TRibbonSeparator.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.40.3.6 TRibbonSeparator.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonSeparator.

77.1.41 TRibbonSmallButtonControl Class

Class Hierarchy

Pascal

```
TRibbonSmallButtonControl = class(TRibbonBaseButtonControl);
```

C++

```
class TRibbonSmallButtonControl : public TRibbonBaseButtonControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSmallButtonControl.

77.1.41.1 TRibbonSmallButtonControl Members

The following tables list the members exposed by TRibbonSmallButtonControl.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	DrawBackground (see page 6985)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.
	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.

	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawText (see page 6986)	Draws text on the control.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7151)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 6984)	Executes the control's function.
	Create (see page 6981)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	KeyTipClick (see page 6988)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.41.2 TRibbonSmallButtonControl Methods

The methods of the TRibbonSmallButtonControl class are listed here.

Public Methods

	Name	Description
	CalcBounds (see page 7151)	Calculates the size of the control.

77.1.41.2.1 TRibbonSmallButtonControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

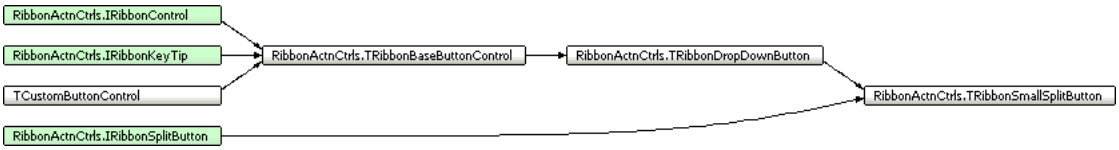
This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.42 TRibbonSmallSplitButton Class

Class Hierarchy



Pascal

```
TRibbonSmallSplitButton = class(TRibbonDropDownButton, IRibbonSplitButton);
```

C++

```
class TRibbonSmallSplitButton : public TRibbonDropDownButton, public IRibbonSplitButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSmallSplitButton.

77.1.42.1 TRibbonSmallSplitButton Members

The following tables list the members exposed by TRibbonSmallSplitButton.

Protected Fields

	Name	Description
	FCursorInSplit (see page 7154)	This is FCursorInSplit, a member of class TRibbonSmallSplitButton.
	FSplitRect (see page 7154)	This is FSplitRect, a member of class TRibbonSmallSplitButton.

Protected Methods

	Name	Description
	AllowHotKeys (see page 6983)	This is AllowHotKeys, a member of class TRibbonBaseButtonControl.
	CalcLayout (see page 7155)	This is CalcLayout, a member of class TRibbonSmallSplitButton.
	CMMouseLeave (see page 7156)	This is CMMouseLeave, a member of class TRibbonSmallSplitButton.
	DoDropDown (see page 7081)	This is DoDropDown, a member of class TRibbonDropDownButton.
	DrawArrow (see page 7081)	This is DrawArrow, a member of class TRibbonDropDownButton.
	DrawBackground (see page 7156)	Draws a background image on the control.
	DrawGlyph (see page 6985)	Draws a symbol on the control.

	DrawSeparator (see page 6985)	This is DrawSeparator, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 6986)	This is DrawShadowedText, a member of class TRibbonBaseButtonControl.
	DrawShadowedText (see page 7081)	This is DrawShadowedText, a member of class TRibbonDropDownButton.
	DrawText (see page 6986)	Draws text on the control.
	DrawText (see page 7082)	Draws text on the control.
	GetArrowPosition (see page 7157)	This is GetArrowPosition, a member of class TRibbonSmallSplitButton.
	GetImageSize (see page 6986)	This is GetImageSize, a member of class TRibbonBaseButtonControl.
	GetPopupClass (see page 7082)	This is GetPopupClass, a member of class TRibbonDropDownButton.
	InSplit (see page 7157)	This is InSplit, a member of class TRibbonSmallSplitButton.
	IsButton (see page 6988)	This is IsButton, a member of class TRibbonBaseButtonControl.
	IsRibbonAvailable (see page 6988)	This is IsRibbonAvailable, a member of class TRibbonBaseButtonControl.
	MouseDown (see page 7157)	OnMouseDown event dispatcher.
	MouseMove (see page 7158)	Respond to mouse moving over control area..
	RibbonStyle (see page 6988)	This is RibbonStyle, a member of class TRibbonBaseButtonControl.

Protected Properties

	Name	Description
	DrawDown (see page 6990)	This is DrawDown, a member of class TRibbonBaseButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 7155)	Calculates the size of the control.
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	CanAddToQuickAccessToolbar (see page 6984)	This is CanAddToQuickAccessToolbar, a member of class TRibbonBaseButtonControl.
	Click (see page 7156)	Executes the control's function.
	Create (see page 7153)	Creates an instance of TCustomButtonControl.
	Destroy (see page 6982)	Destroys an instance of TCustomActionControl.
	DragDrop (see page 6984)	Performs actions and generates an OnDragDrop event.
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	GetKeyTip (see page 6986)	Returns current **KeyTip** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HandleKeyTip (see page 6987)	Indicates if command's **KeyTip** matches currently input KeyTip.
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	HideKeyTip (see page 6987)	Hide **KeyTip** for command.
	KeyTipClick (see page 7157)	Handle clicking a **KeyTip**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	SetKeyTip (see page 6989)	Sets **KeyTip** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.
	ShowKeyTip (see page 6989)	Shows **KeyTip** for command in specified position.
	UpdateSplitBounds (see page 7158)	This is UpdateSplitBounds, a member of class TRibbonSmallSplitButton.

Public Properties

	Name	Description
	CursorInSplit (see page 7159)	This is CursorInSplit, a member of class TRibbonSmallSplitButton.
	KeyTip (see page 7282)	Current **KeyTip** for command

77.1.42.2 TRibbonSmallSplitButton.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

77.1.42.3 TRibbonSmallSplitButton Fields

The fields of the TRibbonSmallSplitButton class are listed here.

Protected Fields

	Name	Description
	FCursorInSplit (see page 7154)	This is FCursorInSplit, a member of class TRibbonSmallSplitButton.
	FSplitRect (see page 7154)	This is FSplitRect, a member of class TRibbonSmallSplitButton.

77.1.42.3.1 TRibbonSmallSplitButton.FCursorInSplit Field

Pascal

```
FCursorInSplit: Boolean;
```

C++

```
Boolean FCursorInSplit;
```

Description

This is FCursorInSplit, a member of class TRibbonSmallSplitButton.

77.1.42.3.2 TRibbonSmallSplitButton.FSplitRect Field

Pascal

```
FSplitRect: TRect;
```

C++

```
TRect FSplitRect;
```

Description

This is FSplitRect, a member of class TRibbonSmallSplitButton.

77.1.42.4 TRibbonSmallSplitButton Methods

The methods of the TRibbonSmallSplitButton class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7155)	This is CalcLayout, a member of class TRibbonSmallSplitButton.
	CMMouseLeave (see page 7156)	This is CMMouseLeave, a member of class TRibbonSmallSplitButton.
	DrawBackground (see page 7156)	Draws a background image on the control.
	GetArrowPosition (see page 7157)	This is GetArrowPosition, a member of class TRibbonSmallSplitButton.
	InSplit (see page 7157)	This is InSplit, a member of class TRibbonSmallSplitButton.
	MouseDown (see page 7157)	OnMouseDown event dispatcher.
	MouseMove (see page 7158)	Respond to mouse moving over control area..

Public Methods

	Name	Description
	CalcBounds (see page 7155)	Calculates the size of the control.
	Click (see page 7156)	Executes the control's function.
	KeyTipClick (see page 7157)	Handle clicking a **KeyTip**.
	UpdateSplitBounds (see page 7158)	This is UpdateSplitBounds, a member of class TRibbonSmallSplitButton.

77.1.42.4.1 TRibbonSmallSplitButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

77.1.42.4.2 TRibbonSmallSplitButton.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TRibbonSmallSplitButton.

77.1.42.4.3 TRibbonSmallSplitButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click

Click

UsageCount

77.1.42.4.4 TRibbonSmallSplitButton.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); override;
```

C++

```
virtual __fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TRibbonSmallSplitButton.

77.1.42.4.5 TRibbonSmallSplitButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.42.4.6 TRibbonSmallSplitButton.GetArrowPosition Method

Pascal

```
function GetArrowPosition(const ARect: TRect): TPoint; override;
```

C++

```
virtual __fastcall TPoint GetArrowPosition(const TRect ARect);
```

Description

This is GetArrowPosition, a member of class TRibbonSmallSplitButton.

77.1.42.4.7 TRibbonSmallSplitButton.InSplit Method

Pascal

```
function InSplit(X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean InSplit(int X, int Y);
```

Description

This is InSplit, a member of class TRibbonSmallSplitButton.

77.1.42.4.8 TRibbonSmallSplitButton.KeyTipClick Method

Handle clicking a **KeyTip**.

Pascal

```
procedure KeyTipClick; override;
```

C++

```
virtual __fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip**. Some requires need to be handled differently when clicked by a **KeyTip**.

77.1.42.4.9 TRibbonSmallSplitButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

77.1.42.4.10 TRibbonSmallSplitButton.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

77.1.42.4.11 TRibbonSmallSplitButton.UpdateSplitBounds Method

Pascal

```
procedure UpdateSplitBounds; virtual;
```

C++

```
virtual __fastcall UpdateSplitBounds();
```

Description

This is UpdateSplitBounds, a member of class TRibbonSmallSplitButton.

77.1.42.5 TRibbonSmallSplitButton Properties

The properties of the TRibbonSmallSplitButton class are listed here.

Public Properties

	Name	Description
	CursorInSplit (see page 7159)	This is CursorInSplit, a member of class TRibbonSmallSplitButton.

77.1.42.5.1 TRibbonSmallSplitButton.CursorInSplit Property

Pascal

```
property CursorInSplit: Boolean;
```

C++

```
__property Boolean CursorInSplit;
```

Description

This is CursorInSplit, a member of class TRibbonSmallSplitButton.

77.1.43 TRibbonSpinButton Class

Class Hierarchy

Pascal

```
TRibbonSpinButton = class(TWinControl);
```

C++

```
class TRibbonSpinButton : public TWinControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSpinButton.

77.1.43.1 TRibbonSpinButton Members

The following tables list the members exposed by TRibbonSpinButton.

Protected Methods

	Name	Description
	Control (see page 7161)	This is Control, a member of class TRibbonSpinButton.
	KeyDown (see page 7161)	Respond to key press events.
	Loaded (see page 7162)	Initializes the control after it is loaded from a stream.
	Notification (see page 7162)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
	Create (see page 7160)	Creates an instance of TWinControl.
	SetBounds (see page 7163)	Sets the windowed control's boundary properties all at once.

Published Events

	Name	Description
	OnDownClick (see page 7183)	This is OnDownClick, a member of class TRibbonSpinButton.
	OnDragDrop (see page 7183)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7186)	Occurs when the user drags an object over a control.
	OnEndDock (see page 7190)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 7190)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7191)	Occurs when a control receives the input focus.
	OnExit (see page 7194)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 7196)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 7197)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUpClick (see page 7198)	This is OnUpClick, a member of class TRibbonSpinButton.

Published Properties

	Name	Description
	Align (see page 7164)	Determines how the control aligns within its container (parent control).
	Anchors (see page 7167)	Specifies how the control is anchored to its parent.
	Constraints (see page 7168)	Specifies the size constraints for the control.
	Ctl3D (see page 7169)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DownGlyph (see page 7169)	This is DownGlyph, a member of class TRibbonSpinButton.
	DownNumGlyphs (see page 7169)	This is DownNumGlyphs, a member of class TRibbonSpinButton.
	DragCursor (see page 7170)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 7170)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 7170)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 7172)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 7172)	This is FocusControl, a member of class TRibbonSpinButton.
	ParentCtl3D (see page 7173)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 7173)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 7173)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 7178)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7180)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7181)	Determines if the user can tab to a control.
	UpGlyph (see page 7181)	This is UpGlyph, a member of class TRibbonSpinButton.
	UpNumGlyphs (see page 7181)	This is UpNumGlyphs, a member of class TRibbonSpinButton.
	Visible (see page 7181)	Determines whether the component appears onscreen.

77.1.43.2 TRibbonSpinButton.Create Constructor

Creates an instance of TWinControl.

Pascal

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

C++

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

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the

AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy method to free those resources.

See Also

Brush

TabOrder

CreateParented

InsertComponent

Create

Owner

77.1.43.3 TRibbonSpinButton Methods

The methods of the TRibbonSpinButton class are listed here.

Protected Methods

	Name	Description
	Control (see page 7161)	This is Control, a member of class TRibbonSpinButton.
V	KeyDown (see page 7161)	Respond to key press events.
V	Loaded (see page 7162)	Initializes the control after it is loaded from a stream.
V	Notification (see page 7162)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
V	SetBounds (see page 7163)	Sets the windowed control's boundary properties all at once.

77.1.43.3.1 TRibbonSpinButton.Control Method

Pascal

```
function Control: TRibbonSpinEdit;
```

C++

```
__fastcall TRibbonSpinEdit Control();
```

Description

This is Control, a member of class TRibbonSpinButton.

77.1.43.3.2 TRibbonSpinButton.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

77.1.43.3.3 TRibbonSpinButton.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

77.1.43.3.4 TRibbonSpinButton.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

77.1.43.3.5 TRibbonSpinButton.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

77.1.43.4 TRibbonSpinButton Properties

The properties of the TRibbonSpinButton class are listed here.

Published Properties

	Name	Description
	Align (see page 7164)	Determines how the control aligns within its container (parent control).
	Anchors (see page 7167)	Specifies how the control is anchored to its parent.
	Constraints (see page 7168)	Specifies the size constraints for the control.
	Ctl3D (see page 7169)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DownGlyph (see page 7169)	This is DownGlyph, a member of class TRibbonSpinButton.
	DownNumGlyphs (see page 7169)	This is DownNumGlyphs, a member of class TRibbonSpinButton.
	DragCursor (see page 7170)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 7170)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 7170)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 7172)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 7172)	This is FocusControl, a member of class TRibbonSpinButton.
	ParentCtl3D (see page 7173)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 7173)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 7173)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 7178)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7180)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7181)	Determines if the user can tab to a control.
	UpGlyph (see page 7181)	This is UpGlyph, a member of class TRibbonSpinButton.
	UpNumGlyphs (see page 7181)	This is UpNumGlyphs, a member of class TRibbonSpinButton.
	Visible (see page 7181)	Determines whether the component appears onscreen.

77.1.43.4.1 TRibbonSpinButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of

the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
```

```
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

77.1.43.4.2 TRibbonSpinButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

77.1.43.4.3 TRibbonSpinButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

77.1.43.4.4 TRibbonSpinButton.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

77.1.43.4.5 TRibbonSpinButton.DownGlyph Property

Pascal

```
property DownGlyph: TBitmap;
```

C++

```
__property TBitmap DownGlyph;
```

Description

This is DownGlyph, a member of class TRibbonSpinButton.

77.1.43.4.6 TRibbonSpinButton.DownNumGlyphs Property

Pascal

```
property DownNumGlyphs: TNumGlyphs;
```

C++

```
__property TNumGlyphs DownNumGlyphs;
```

Description

This is DownNumGlyphs, a member of class TRibbonSpinButton.

77.1.43.4.7 TRibbonSpinButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

77.1.43.4.8 TRibbonSpinButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

77.1.43.4.9 TRibbonSpinButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

77.1.43.4.10 TRibbonSpinButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

77.1.43.4.11 TRibbonSpinButton.FocusControl Property

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

This is FocusControl, a member of class TRibbonSpinButton.

77.1.43.4.12 TRibbonSpinButton.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

77.1.43.4.13 TRibbonSpinButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

77.1.43.4.14 TRibbonSpinButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}
```

```
/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
```

```
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

77.1.43.4.15 TRibbonSpinButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

77.1.43.4.16 TRibbonSpinButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

77.1.43.4.17 TRibbonSpinButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

77.1.43.4.18 TRibbonSpinButton.UpGlyph Property

Pascal

```
property UpGlyph: TBitmap;
```

C++

```
__property TBitmap UpGlyph;
```

Description

This is UpGlyph, a member of class TRibbonSpinButton.

77.1.43.4.19 TRibbonSpinButton.UpNumGlyphs Property

Pascal

```
property UpNumGlyphs: TNumGlyphs;
```

C++

```
__property TNumGlyphs UpNumGlyphs;
```

Description

This is UpNumGlyphs, a member of class TRibbonSpinButton.

77.1.43.4.20 TRibbonSpinButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

77.1.43.5 TRibbonSpinButton Events

The events of the TRibbonSpinButton class are listed here.

Published Events

	Name	Description
	OnDownClick (see page 7183)	This is OnDownClick, a member of class TRibbonSpinButton.
	OnDragDrop (see page 7183)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7186)	Occurs when the user drags an object over a control.
	OnEndDock (see page 7190)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 7190)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7191)	Occurs when a control receives the input focus.
	OnExit (see page 7194)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 7196)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 7197)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUpClick (see page 7198)	This is OnUpClick, a member of class TRibbonSpinButton.

77.1.43.5.1 TRibbonSpinButton.OnDownClick Event

Pascal

```
property OnDownClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownClick;
```

Description

This is OnDownClick, a member of class TRibbonSpinButton.

77.1.43.5.2 TRibbonSpinButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

77.1.43.5.3 TRibbonSpinButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

77.1.43.5.4 TRibbonSpinButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

77.1.43.5.5 TRibbonSpinButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

77.1.43.5.6 TRibbonSpinButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
  applied to the active control. Clicking the button sets the
  ActiveControl to the button.  That is why we need to save
  the ActiveControl in a shared OnEnter procedure.  In the rich
  edit, only the text selected converts.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
     FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

77.1.43.5.7 TRibbonSpinButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
```

```
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl (see page 2145)

77.1.43.5.8 TRibbonSpinButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

77.1.43.5.9 TRibbonSpinButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

77.1.43.5.10 TRibbonSpinButton.OnUpClick Event

Pascal

```
property OnUpClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpClick;
```

Description

This is OnUpClick, a member of class TRibbonSpinButton.

77.1.44 TRibbonSpinEdit Class

Class Hierarchy

Pascal

```
TRibbonSpinEdit = class(TCustomEdit);
```

C++

```
class TRibbonSpinEdit : public TCustomEdit;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSpinEdit.

77.1.44.1 TRibbonSpinEdit Members

The following tables list the members exposed by TRibbonSpinEdit.

Protected Methods

	Name	Description
	CMMouseenter (see page 7201)	This is CMMouseenter, a member of class TRibbonSpinEdit.
	CMMouseleave (see page 7201)	This is CMMouseleave, a member of class TRibbonSpinEdit.
V	CreateParams (see page 7201)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWnd (see page 7202)	Creates a Windows control corresponding to the edit control.
V	DownClick (see page 7202)	This is DownClick, a member of class TRibbonSpinEdit.
V	IsValidChar (see page 7203)	This is IsValidChar, a member of class TRibbonSpinEdit.
V	KeyDown (see page 7203)	Respond to key press events.
V	KeyPress (see page 7204)	Respond to keyboard input.
V	SetEnabled (see page 7204)	Sets the value of the Enabled property.
V	SetParent (see page 7205)	Sets the parent of the control.

	UpClick (see page 7206)	This is UpClick, a member of class TRibbonSpinEdit.
	WMKillFocus (see page 7206)	This is WMKillFocus, a member of class TRibbonSpinEdit.

Public Methods

	Name	Description
	Create (see page 7200)	Creates an instance of TCustomEdit.
	Destroy (see page 7200)	Destroys an instance of TWinControl.
	GetChildren (see page 7203)	Calls a specified method for each child of the control.

Public Properties

	Name	Description
	Button (see page 7209)	This is Button, a member of class TRibbonSpinEdit.

Published Events

	Name	Description
	OnChange (see page 7228)	Occurs when the text for the edit control may have changed.
	OnClick (see page 7228)	Occurs when the user clicks the control.
	OnDblClick (see page 7230)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 7230)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7233)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 7237)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7238)	Occurs when a control receives the input focus.
	OnExit (see page 7240)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 7243)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 7246)	Occurs when a key is pressed.
	OnKeyUp (see page 7248)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 7249)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 7252)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 7254)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 7256)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Anchors (see page 7207)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 7207)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 7208)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	Color (see page 7209)	Specifies the background color of the control.
	Constraints (see page 7210)	Specifies the size constraints for the control.
	Ctl3D (see page 7210)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 7211)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 7211)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorEnabled (see page 7213)	This is EditorEnabled, a member of class TRibbonSpinEdit.
	Enabled (see page 7213)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 7214)	Controls the attributes of text written on or in the control.
	Increment (see page 7215)	This is Increment, a member of class TRibbonSpinEdit.
	MaxLength (see page 7215)	Specifies the maximum number of characters the user can enter into the edit control.
	MaxValue (see page 7215)	This is MaxValue, a member of class TRibbonSpinEdit.

	MinValue (see page 7216)	This is MinValue, a member of class TRibbonSpinEdit.
	ParentColor (see page 7216)	Determines where a control looks for its color information.
	ParentCtl3D (see page 7216)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 7217)	Determines where a control looks for its font information.
	ParentShowHint (see page 7217)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 7218)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 7223)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 7223)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7224)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7225)	Determines if the user can tab to a control.
	Value (see page 7226)	This is Value, a member of class TRibbonSpinEdit.
	Visible (see page 7226)	Determines whether the component appears onscreen.

77.1.44.2 TRibbonSpinEdit.Create Constructor

Creates an instance of TCustomEdit.

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomEdit.

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner

77.1.44.3 TRibbonSpinEdit.Destroy Destructor

Destroys an instance of TWinControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonSpinEdit();
```

Description

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

See Also

System::TObject::Free

Destroy

77.1.44.4 TRibbonSpinEdit Methods

The methods of the TRibbonSpinEdit class are listed here.

Protected Methods

	Name	Description
	CMMouseenter (see page 7201)	This is CMMouseenter, a member of class TRibbonSpinEdit.
	CMMouseleave (see page 7201)	This is CMMouseleave, a member of class TRibbonSpinEdit.
V	CreateParams (see page 7201)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWnd (see page 7202)	Creates a Windows control corresponding to the edit control.
V	DownClick (see page 7202)	This is DownClick, a member of class TRibbonSpinEdit.
V	IsValidChar (see page 7203)	This is IsValidChar, a member of class TRibbonSpinEdit.
V	KeyDown (see page 7203)	Respond to key press events.
V	KeyPress (see page 7204)	Respond to keyboard input.
V	SetEnabled (see page 7204)	Sets the value of the Enabled property.
V	SetParent (see page 7205)	Sets the parent of the control.
V	UpClick (see page 7206)	This is UpClick, a member of class TRibbonSpinEdit.
	WMKillFocus (see page 7206)	This is WMKillFocus, a member of class TRibbonSpinEdit.

Public Methods

	Name	Description
V	GetChildren (see page 7203)	Calls a specified method for each child of the control.

77.1.44.4.1 TRibbonSpinEdit.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TRibbonSpinEdit.

77.1.44.4.2 TRibbonSpinEdit.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TRibbonSpinEdit.

77.1.44.4.3 TRibbonSpinEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. TCustomEdit overrides the inherited method to specify a scrollable single-line edit control that reflects the current values of the BorderStyle, HideSelection, PasswordChar, and ReadOnly properties.

See Also

BorderStyle

CreateWindowHandle

CreateWnd

HideSelection

PasswordChar

ReadOnly

77.1.44.4.4 TRibbonSpinEdit.CreateWnd Method

Creates a Windows control corresponding to the edit control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the edit control is created, and whenever a change to the edit control requires the window to be recreated. After calling the inherited method, CreateWnd sends the newly created window messages to enforce the MaxLength and PasswordChar properties.

See Also

CreateParams

CreateWindowHandle

DestroyWnd

MaxLength

PasswordChar

CreateWnd

77.1.44.4.5 TRibbonSpinEdit.DownClick Method

Pascal

```
procedure DownClick(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DownClick(TObject * Sender);
```

Description

This is DownClick, a member of class TRibbonSpinEdit.

77.1.44.4.6 TRibbonSpinEdit.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

77.1.44.4.7 TRibbonSpinEdit.IsValidChar Method

Pascal

```
function IsValidChar(Key: Char): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsValidChar(Char Key);
```

Description

This is IsValidChar, a member of class TRibbonSpinEdit.

77.1.44.4.8 TRibbonSpinEdit.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

77.1.44.4.9 TRibbonSpinEdit.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown

KeyUp

OnKeyPress

77.1.44.4.10 TRibbonSpinEdit.SetEnabled Method

Sets the value of the Enabled property.

Pascal

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

C++

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

Description

SetEnabled is the protected write implementation of the Enabled property. After changing the internal representation of the Enabled property, SetEnabled sends a Windows message back to the control so that it can respond to the change. When TControl receives this message, it invalidates the control so that its appearance reflects the change (for example, so that it appears grayed). Some descendants respond to this message in other ways (for example, windowed controls ensure that a disabled control does not have focus).

Override GetEnabled and SetEnabled to change the implementation of the Enabled property. For example, A descendant of TControl might change the way the Enabled property interacts with an associated action. When overriding SetEnabled, be sure to call the inherited method.

See Also

GetEnabled

Enabled

Invalidate

77.1.44.4.11 TRibbonSpinEdit.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy

InsertControl

Parent

RemoveControl

77.1.44.4.12 TRibbonSpinEdit.UpClick Method

Pascal

```
procedure UpClick(Sender: TObject); virtual;
```

C++

```
virtual __fastcall UpClick(TObject * Sender);
```

Description

This is UpClick, a member of class TRibbonSpinEdit.

77.1.44.4.13 TRibbonSpinEdit.WMKillFocus Method

Pascal

```
procedure WMKillFocus(var Message: TWMKillFocus); message WM_KILLFOCUS;
```

C++

```
__fastcall WMKillFocus(TWMKillFocus Message);
```

Description

This is WMKillFocus, a member of class TRibbonSpinEdit.

77.1.44.5 TRibbonSpinEdit Properties

The properties of the TRibbonSpinEdit class are listed here.

Public Properties

	Name	Description
R	Button (see page 7209)	This is Button, a member of class TRibbonSpinEdit.

Published Properties

	Name	Description
	Anchors (see page 7207)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 7207)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 7208)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	Color (see page 7209)	Specifies the background color of the control.
	Constraints (see page 7210)	Specifies the size constraints for the control.
	Ctl3D (see page 7210)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 7211)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 7211)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorEnabled (see page 7213)	This is EditorEnabled, a member of class TRibbonSpinEdit.
	Enabled (see page 7213)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 7214)	Controls the attributes of text written on or in the control.
	Increment (see page 7215)	This is Increment, a member of class TRibbonSpinEdit.
	MaxLength (see page 7215)	Specifies the maximum number of characters the user can enter into the edit control.
	MaxValue (see page 7215)	This is MaxValue, a member of class TRibbonSpinEdit.
	MinValue (see page 7216)	This is MinValue, a member of class TRibbonSpinEdit.
	ParentColor (see page 7216)	Determines where a control looks for its color information.

	ParentCtl3D (see page 7216)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 7217)	Determines where a control looks for its font information.
	ParentShowHint (see page 7217)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 7218)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 7223)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 7223)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 7224)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 7225)	Determines if the user can tab to a control.
	Value (see page 7226)	This is Value, a member of class TRibbonSpinEdit.
	Visible (see page 7226)	Determines whether the component appears onscreen.

77.1.44.5.1 TRibbonSpinEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

77.1.44.5.2 TRibbonSpinEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect;
```

C++

```
__property AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll

SelLength

SelStart

SelText

SetSelTextBuf

77.1.44.5.3 TRibbonSpinEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle

ClientHeight

ControlStyle

Height

77.1.44.5.4 TRibbonSpinEdit.Button Property

Pascal

```
property Button: TRibbonSpinButton;
```

C++

```
__property TRibbonSpinButton Button;
```

Description

This is Button, a member of class TRibbonSpinEdit.

77.1.44.5.5 TRibbonSpinEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
```

```
  user to select a color with the dialog box. When a color
  value is selected with the dialog box, it is assigned to
  the shape control.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if ColorDialog1.Execute then
      Shape1.Brush.Color := ColorDialog1.Color;
  end;
```

See Also

ParentColor

77.1.44.5.6 TRibbonSpinEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

77.1.44.5.7 TRibbonSpinEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

77.1.44.5.8 TRibbonSpinEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

77.1.44.5.9 TRibbonSpinEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
```

```
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

77.1.44.5.10 TRibbonSpinEdit.EditorEnabled Property

Pascal

```
property EditorEnabled: Boolean;
```

C++

```
__property Boolean EditorEnabled;
```

Description

This is EditorEnabled, a member of class TRibbonSpinEdit.

77.1.44.5.11 TRibbonSpinEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

77.1.44.5.12 TRibbonSpinEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

77.1.44.5.13 TRibbonSpinEdit.Increment Property

Pascal

```
property Increment: LongInt;
```

C++

```
__property LongInt Increment;
```

Description

This is Increment, a member of class TRibbonSpinEdit.

77.1.44.5.14 TRibbonSpinEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength;
```

C++

```
__property MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

77.1.44.5.15 TRibbonSpinEdit.MaxValue Property

Pascal

```
property MaxValue: LongInt;
```

C++

```
__property LongInt MaxValue;
```

Description

This is MaxValue, a member of class TRibbonSpinEdit.

77.1.44.5.16 TRibbonSpinEdit.MinValue Property

Pascal

```
property MinValue: LongInt;
```

C++

```
__property LongInt MinValue;
```

Description

This is MinValue, a member of class TRibbonSpinEdit.

77.1.44.5.17 TRibbonSpinEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

77.1.44.5.18 TRibbonSpinEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

77.1.44.5.19 TRibbonSpinEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

77.1.44.5.20 TRibbonSpinEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

77.1.44.5.21 TRibbonSpinEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
```

```
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
```

```
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup (see page 4942)

Popup (see page 4939)

OnContextPopup

77.1.44.5.22 TRibbonSpinEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard

GetSelTextBuf

77.1.44.5.23 TRibbonSpinEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint (see page 1986)

Displaying Fly-over Help

77.1.44.5.24 TRibbonSpinEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

77.1.44.5.25 TRibbonSpinEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

77.1.44.5.26 TRibbonSpinEdit.Value Property

Pascal

```
property Value: LongInt;
```

C++

```
__property LongInt Value;
```

Description

This is Value, a member of class TRibbonSpinEdit.

77.1.44.5.27 TRibbonSpinEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

77.1.44.6 TRibbonSpinEdit Events

The events of the TRibbonSpinEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 7228)	Occurs when the text for the edit control may have changed.
	OnClick (see page 7228)	Occurs when the user clicks the control.
	OnDblClick (see page 7230)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 7230)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 7233)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 7237)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 7238)	Occurs when a control receives the input focus.

	OnExit (see page 7240)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 7243)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 7246)	Occurs when a key is pressed.
	OnKeyUp (see page 7248)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 7249)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 7252)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 7254)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 7256)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

77.1.44.6.1 TRibbonSpinEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange;
```

C++

```
__property OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change

Text

77.1.44.6.2 TRibbonSpinEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).

- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel

Default

Click (see page 4881)

OnExecute

Action

Adding Silent Video Clips to an Application

77.1.44.6.3 TRibbonSpinEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

77.1.44.6.4 TRibbonSpinEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
```

```
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

77.1.44.6.5 TRibbonSpinEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
  property OnDragOver;
```

C++

```
  __property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
```

```
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

77.1.44.6.6 TRibbonSpinEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

77.1.44.6.7 TRibbonSpinEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl (see page 2145)

OnActivate (see page 2196)

Using a Single Navigator for Multiple Datasets

77.1.44.6.8 TRibbonSpinEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl (see page 2145)

77.1.44.6.9 TRibbonSpinEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  Behavior is dependent on keyboard driver(USB, serial, etc).
  */
  void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
        TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      FormColor = Form1->Color;
      Form1->Color = clAqua;
      if (ComboBox1->Text == "KeyDown")
        ComboBox1->Text = "KeyDown2";
      else
        ComboBox1->Text = "KeyDown";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }

  void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      Form1->Color = FormColor;
      if (ComboBox1->Text == "KeyUp")
        ComboBox1->Text = "KeyUp2";
      else
        ComboBox1->Text = "KeyUp";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }
  /*
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  */
  #include <Printers.hpp>

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
     KeyPreview = True;

  }

  void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

  {
    if (Key == VK_ESCAPE && Printer()->Printing)
    {
      Printer()->Abort();
      MessageDlg("Printing aborted", mtInformation,
                 TMsgDlgButtons() << mbOK,0);
    }
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
    Memo1->Parent = Form1;
    Memo1->Visible = True;
    Memo1->Width = 700;
    if (OpenDialog1->Execute())
    {
      Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
```

```
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 2167)

OnKeyPress

OnKeyUp

77.1.44.6.10 TRibbonSpinEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview (see page 2167)

77.1.44.6.11 TRibbonSpinEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

77.1.44.6.12 TRibbonSpinEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

77.1.44.6.13 TRibbonSpinEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

77.1.44.6.14 TRibbonSpinEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

77.1.44.6.15 TRibbonSpinEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

77.1.45 TRibbonSplitMenuItem Class

Class Hierarchy

Pascal

```
TRibbonSplitMenuItem = class(TRibbonApplicationMenuItem, IRibbonSplitButton);
```

C++

```
class TRibbonSplitMenuItem : public TRibbonApplicationMenuItem, public IRibbonSplitButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonSplitMenuItem.

77.1.45.1 TRibbonSplitMenuItem Members

The following tables list the members exposed by TRibbonSplitMenuItem.

Protected Fields

	Name	Description
	FMouseInSplit (see page 7259)	This is FMouseInSplit, a member of class TRibbonSplitMenuItem.
	FSplitRect (see page 7259)	This is FSplitRect, a member of class TRibbonSplitMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7120)	This is CalcLayout, a member of class TRibbonMenuItem.
	DrawBackground (see page 7260)	Draws a background image on the control.
	DrawGlyph (see page 6978)	Draws a symbol on the control.
	DrawGlyph (see page 7121)	Draws a symbol on the control.
	DrawSelectedBackground (see page 7260)	This is DrawSelectedBackground, a member of class TRibbonSplitMenuItem.
	DrawSeparator (see page 6979)	This is DrawSeparator, a member of class TRibbonApplicationMenuItem.
	DrawSeparator (see page 7122)	This is DrawSeparator, a member of class TRibbonMenuItem.
	DrawShadowedText (see page 6979)	This is DrawShadowedText, a member of class TRibbonApplicationMenuItem.
	DrawShadowedText (see page 7122)	This is DrawShadowedText, a member of class TRibbonMenuItem.
	DrawSubMenuGlyph (see page 6979)	This is DrawSubMenuGlyph, a member of class TRibbonApplicationMenuItem.
	DrawText (see page 6980)	Draws text on the control.

	DrawText (see page 7122)	Draws text on the control.
	GetImageSize (see page 6980)	This is GetImageSize, a member of class TRibbonApplicationMenuItem.
	GetShowShortCut (see page 7122)	This is GetShowShortCut, a member of class TRibbonMenuItem.
	IsInSplit (see page 7261)	This is IsInSplit, a member of class TRibbonSplitMenuItem.
	MouseMove (see page 7261)	Respond to mouse moving over control area..
	RibbonStyle (see page 7123)	This is RibbonStyle, a member of class TRibbonMenuItem.

Protected Properties

	Name	Description
	DrawBanner (see page 7123)	This is DrawBanner, a member of class TRibbonMenuItem.
	MouseInSplit (see page 7262)	This is MouseInSplit, a member of class TRibbonSplitMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7259)	Calculates the size of the menu item.
	Click (see page 7260)	Executes the control's function.
	Create (see page 7258)	Creates an instance of TCustomMenuItem.
	UpdateSplitBounds (see page 7261)	This is UpdateSplitBounds, a member of class TRibbonSplitMenuItem.

77.1.45.2 TRibbonSplitMenuItem.Create Constructor

Creates an instance of TCustomMenuItem.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomMenuItem. However, you should never attempt to instantiate a TCustomMenuItem. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

77.1.45.3 TRibbonSplitMenuItem Fields

The fields of the TRibbonSplitMenuItem class are listed here.

Protected Fields

	Name	Description
	FMouseInSplit (see page 7259)	This is FMouseInSplit, a member of class TRibbonSplitMenuItem.
	FSplitRect (see page 7259)	This is FSplitRect, a member of class TRibbonSplitMenuItem.

77.1.45.3.1 TRibbonSplitMenuItem.FMouseInSplit Field

Pascal

```
FMouseInSplit: Boolean;
```

C++

```
Boolean FMouseInSplit;
```

Description

This is FMouseInSplit, a member of class TRibbonSplitMenuItem.

77.1.45.3.2 TRibbonSplitMenuItem.FSplitRect Field

Pascal

```
FSplitRect: TRect;
```

C++

```
TRect FSplitRect;
```

Description

This is FSplitRect, a member of class TRibbonSplitMenuItem.

77.1.45.4 TRibbonSplitMenuItem Methods

The methods of the TRibbonSplitMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7260)	Draws a background image on the control.
	DrawSelectedBackground (see page 7260)	This is DrawSelectedBackground, a member of class TRibbonSplitMenuItem.
	IsInSplit (see page 7261)	This is IsInSplit, a member of class TRibbonSplitMenuItem.
	MouseMove (see page 7261)	Respond to mouse moving over control area..

Public Methods

	Name	Description
	CalcBounds (see page 7259)	Calculates the size of the menu item.
	Click (see page 7260)	Executes the control's function.
	UpdateSplitBounds (see page 7261)	This is UpdateSplitBounds, a member of class TRibbonSplitMenuItem.

77.1.45.4.1 TRibbonSplitMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.45.4.2 TRibbonSplitMenuItem.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl does little except bring the form containing the item into focus, and prevent the inherited method from being called. The reason for this lack of action is that at runtime, no action should be taken until the primary mouse button is pressed and released, which makes the OnClick event handler responsible for taking action. Click takes place only when the mouse button is first pressed, prior to release.

See Also

OnClick

77.1.45.4.3 TRibbonSplitMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.45.4.4 TRibbonSplitMenuItem.DrawSelectedBackground Method

Pascal

```
procedure DrawSelectedBackground(ARect: TRect); override;
```

C++

```
virtual __fastcall DrawSelectedBackground(TRect ARect);
```

Description

This is DrawSelectedBackground, a member of class TRibbonSplitMenuItem.

77.1.45.4.5 TRibbonSplitMenuItem.IsInSplit Method

Pascal

```
function IsInSplit(const Pt: TPoint): Boolean;
```

C++

```
__fastcall Boolean IsInSplit(const TPoint Pt);
```

Description

This is IsInSplit, a member of class TRibbonSplitMenuItem.

77.1.45.4.6 TRibbonSplitMenuItem.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown

MouseUp

OnMouseMove

77.1.45.4.7 TRibbonSplitMenuItem.UpdateSplitBounds Method

Pascal

```
procedure UpdateSplitBounds; virtual;
```

C++

```
virtual __fastcall UpdateSplitBounds();
```

Description

This is UpdateSplitBounds, a member of class TRibbonSplitMenuItem.

77.1.45.5 TRibbonSplitMenuItem Properties

The properties of the TRibbonSplitMenuItem class are listed here.

Protected Properties

	Name	Description
	MouseInSplit (see page 7262)	This is MouseInSplit, a member of class TRibbonSplitMenuItem.

77.1.45.5.1 TRibbonSplitMenuItem.MouseInSplit Property

Pascal

```
property MouseInSplit: Boolean;
```

C++

```
__property Boolean MouseInSplit;
```

Description

This is MouseInSplit, a member of class TRibbonSplitMenuItem.

77.1.46 TRibbonStyleMenuButton Class

Top level menus when using Ribbon (see page 6713) Styles in applications not using Ribbon (see page 6713) interface.

Class Hierarchy

Pascal

```
TRibbonStyleMenuButton = class(TCustomMenuButton);
```

C++

```
class TRibbonStyleMenuButton : public TCustomMenuButton;
```

File

RibbonActnCtrls

Description

TRibbonStyleMenuButton is used for the top level menus when using one of the Ribbon (see page 6713) Styles in an application that doesn't use the Ribbon (see page 6713) interface.

77.1.46.1 TRibbonStyleMenuButton Members

The following tables list the members exposed by TRibbonStyleMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 7263)	Draws a background image on the control.
	RibbonStyle (see page 7263)	This is RibbonStyle, a member of class TRibbonStyleMenuButton.

77.1.46.2 TRibbonStyleMenuButton Methods

The methods of the TRibbonStyleMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7263)	Draws a background image on the control.
	RibbonStyle (see page 7263)	This is RibbonStyle, a member of class TRibbonStyleMenuButton.

77.1.46.2.1 TRibbonStyleMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.46.2.2 TRibbonStyleMenuButton.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonStyleMenuButton.

77.1.47 TRibbonTextControl Class

Class Hierarchy

Pascal

```
TRibbonTextControl = class(TCustomActionControl);
```

C++

```
class TRibbonTextControl : public TCustomActionControl;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonTextControl.

77.1.47.1 TRibbonTextControl Members

The following tables list the members exposed by TRibbonTextControl.

Protected Methods

	Name	Description
	DrawBackground (see page 7265)	Draws a background image on the control.
	DrawText (see page 7265)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7264)	Calculates the size of the control.

77.1.47.2 TRibbonTextControl Methods

The methods of the TRibbonTextControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7265)	Draws a background image on the control.
	DrawText (see page 7265)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 7264)	Calculates the size of the control.

77.1.47.2.1 TRibbonTextControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

77.1.47.2.2 TRibbonTextControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.47.2.3 TRibbonTextControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

77.1.48 TRibbonToolScrollButton Class

Class Hierarchy

Pascal

```
TRibbonToolScrollButton = class(TCustomToolScrollBtn);
```

C++

```
class TRibbonToolScrollButton : public TCustomToolScrollBtn;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TRibbonToolScrollButton.

77.1.48.1 TRibbonToolScrollButton Members

The following tables list the members exposed by TRibbonToolScrollButton.

Protected Methods

	Name	Description
	DrawArrows (see page 7266)	This is DrawArrows, a member of class TRibbonToolScrollButton.
	DrawBackground (see page 7266)	Draws a background image on the control.

77.1.48.2 TRibbonToolScrollButton Methods

The methods of the TRibbonToolScrollButton class are listed here.

Protected Methods

	Name	Description
	DrawArrows (see page 7266)	This is DrawArrows, a member of class TRibbonToolScrollButton.
	DrawBackground (see page 7266)	Draws a background image on the control.

77.1.48.2.1 TRibbonToolScrollButton.DrawArrows Method

Pascal

```
procedure DrawArrows; override;
```

C++

```
virtual __fastcall DrawArrows();
```

Description

This is DrawArrows, a member of class TRibbonToolScrollButton.

77.1.48.2.2 TRibbonToolScrollButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

77.1.49 TTimerSpeedButton Class

Class Hierarchy

Pascal

```
TTimerSpeedButton = class(TSpeedButton);
```

C++

```
class TTimerSpeedButton : public TSpeedButton;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.TTimerSpeedButton.

77.1.49.1 TTimerSpeedButton Members

The following tables list the members exposed by TTimerSpeedButton.

Protected Methods

	Name	Description
	CMMouseenter (see page 7268)	This is CMMouseenter, a member of class TTimerSpeedButton.
	CMMouseleave (see page 7269)	This is CMMouseleave, a member of class TTimerSpeedButton.
	MouseDown (see page 7269)	OnMouseDown event dispatcher.
	MouseUp (see page 7269)	OnMouseUp event dispatcher.
	Paint (see page 7270)	Render control surface.

Protected Properties

	Name	Description
	MouseInControl (see page 7271)	This is MouseInControl, a member of class TTimerSpeedButton.

Public Methods

	Name	Description
	Create (see page 7267)	Creates and initializes an instance of TSpeedButton.
	Destroy (see page 7268)	Destroys an instance of TSpeedButton.

Public Properties

	Name	Description
	ButtonStyle (see page 7270)	This is ButtonStyle, a member of class TTimerSpeedButton.
	TimeBtnState (see page 7271)	This is TimeBtnState, a member of class TTimerSpeedButton.

77.1.49.2 TTimerSpeedButton.Create Constructor

Creates and initializes an instance of TSpeedButton.

Pascal

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

C++

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

Description

Call Create to create a speed button at runtime. Speed buttons placed on forms at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the speed button.

See Also

Owner

77.1.49.3 TTimerSpeedButton.Destroy Destructor

Destroys an instance of TSpeedButton.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTimerSpeedButton();
```

Description

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

See Also

System::TObject::Free

77.1.49.4 TTimerSpeedButton Methods

The methods of the TTimerSpeedButton class are listed here.

Protected Methods

	Name	Description
	CMMouseenter (see page 7268)	This is CMMouseenter, a member of class TTimerSpeedButton.
	CMMouseleave (see page 7269)	This is CMMouseleave, a member of class TTimerSpeedButton.
V	MouseDown (see page 7269)	OnMouseDown event dispatcher.
V	MouseUp (see page 7269)	OnMouseUp event dispatcher.
V	Paint (see page 7270)	Render control surface.

77.1.49.4.1 TTimerSpeedButton.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TTimerSpeedButton.

77.1.49.4.2 TTimerSpeedButton.CMMouseleave Method

Pascal

```
procedure CMMouseleave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseleave(TMessage Message);
```

Description

This is CMMouseleave, a member of class TTimerSpeedButton.

77.1.49.4.3 TTimerSpeedButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

77.1.49.4.4 TTimerSpeedButton.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown

MouseMove

OnMouseUp

77.1.49.4.5 TTimerSpeedButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

77.1.49.5 TTimerSpeedButton Properties

The properties of the TTimerSpeedButton class are listed here.

Protected Properties

	Name	Description
	MouseInControl (see page 7271)	This is MouseInControl, a member of class TTimerSpeedButton.

Public Properties

	Name	Description
	ButtonStyle (see page 7270)	This is ButtonStyle, a member of class TTimerSpeedButton.
	TimeBtnState (see page 7271)	This is TimeBtnState, a member of class TTimerSpeedButton.

77.1.49.5.1 TTimerSpeedButton.ButtonStyle Property

Pascal

```
property ButtonStyle: TSpinButtonStyle;
```

C++

```
__property TSpinButtonStyle ButtonStyle;
```

Description

This is ButtonStyle, a member of class TTimerSpeedButton.

77.1.49.5.2 TTimerSpeedButton.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TTimerSpeedButton.

77.1.49.5.3 TTimerSpeedButton.TimeBtnState Property

Pascal

```
property TimeBtnState: TTimeBtnState;
```

C++

```
__property TTimeBtnState TimeBtnState;
```

Description

This is TimeBtnState, a member of class TTimerSpeedButton.

77.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IRibbonBarKeyTip (see page 7272)	Action Bars that support KeyTips should implement this interface.
	IRibbonContainedControl (see page 7274)	Interface implemented by controls used as containers for displaying other controls.
	IRibbonControl (see page 7276)	Interface implemented by controls visible in Ribbon (see page 6713) Groups.
	IRibbonGroupDropdown (see page 7277)	This is class RibbonActnCtrls.IRibbonGroupDropdown.
	IRibbonKeyTip (see page 7278)	Commands with a **KeyTip (see page 7282)** should implement this interface.
	IRibbonSplitButton (see page 7282)	Interface implemented by controls that have split button.

77.2.1 IRibbonBarKeyTip Interface

Action Bars that support KeyTips should implement this interface.

Class Hierarchy

RibbonActnCtrls.IRibbonBarKeyTip

Pascal

```
IRibbonBarKeyTip = interface;
```

C++

```
__interface IRibbonBarKeyTip;
```

File

RibbonActnCtrls

Description

Action Bars that support KeyTips should implement IRibbonBarKeyTip. The TRibbonQuickAccessToolbar and TRibbonGroup classes implement this interface.

See Also

TRibbonGroup (see page 6865)

TRibbonQuickAccessToolbar (see page 6911)

77.2.1.1 IRibbonBarKeyTip Members

The following tables list the members exposed by IRibbonBarKeyTip.

Public Methods

	Name	Description
	BarHandleKeyTips (see page 7273)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 7273)	Hide any visible **KeyTip** windows.

	BarShowKeyTips (see page 7274)	Show **KeyTip**s for any commands that support them.

77.2.1.2 IRibbonBarKeyTip Methods

The methods of the IRibbonBarKeyTip class are listed here.

Public Methods

	Name	Description
	BarHandleKeyTips (see page 7273)	Check if any commands on Action Bar are partial or full match for **KeyTip**.
	BarHideKeyTips (see page 7273)	Hide any visible **KeyTip** windows.
	BarShowKeyTips (see page 7274)	Show **KeyTip**s for any commands that support them.

77.2.1.2.1 IRibbonBarKeyTip.BarHandleKeyTips Method

Check if any commands on Action Bar are partial or full match for **KeyTip**.

Pascal

```
procedure BarHandleKeyTips(const KeyTip: string; var KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHandleKeyTips(const UnicodeString KeyTip, TKeyTipList KeyTips);
```

Description

BarHandleKeyTips checks if any commands on the Action Bar are a partial or full match for the **KeyTip**. Matches are returned as a TKeyTipMatchItem (see page 6962) in the `List` parameter. If only one item is a match, that command is executed.

See Also

TKeyTipMatchItem (see page 6962)

BarHideKeyTips (see page 7273)

BarShowKeyTips (see page 7274)

77.2.1.2.2 IRibbonBarKeyTip.BarHideKeyTips Method

Hide any visible **KeyTip** windows.

Pascal

```
procedure BarHideKeyTips(KeyTips: TKeyTipList);
```

C++

```
__fastcall BarHideKeyTips(TKeyTipList KeyTips);
```

Description

BarHideKeyTips hides any visible **KeyTip** windows. `ExceptList` may contain a list of TKeyTipMatchItem (see page 6962) objects for commands that should not be hidden. These items are current partial matches and require additional handling.

See Also

TKeyTipMatchItem (see page 6962)

BarHandleKeyTips (see page 7273)

BarShowKeyTips (see page 7274)

77.2.1.2.3 IRibbonBarKeyTip.BarShowKeyTips Method

Show **KeyTip**s for any commands that support them.

Pascal

```
procedure BarShowKeyTips;
```

C++

```
__fastcall BarShowKeyTips();
```

Description

BarShowKeyTips shows **KeyTip**s for any commands that support them.

See Also

BarHandleKeyTips (see page 7273)

BarHideKeyTips (see page 7273)

77.2.2 IRibbonContainedControl Interface

Interface implemented by controls used as containers for displaying other controls.

Class Hierarchy

Pascal

```
IRibbonContainedControl = interface(IRibbonControl);
```

C++

```
__interface IRibbonContainedControl : public IRibbonControl;
```

File

RibbonActnCtrls

Description

IRibbonContainedControl is an interface implemented by a control that is used as a container for displaying other controls.

See Also

IRibbonControl (see page 7276)

IRibbonSplitButton (see page 7282)

77.2.2.1 IRibbonContainedControl Members

The following tables list the members exposed by IRibbonContainedControl.

Public Methods

	Name	Description
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.
	GetContainedControl (see page 7275)	This is GetContainedControl, a member of class IRibbonContainedControl.
	SetContainedControl (see page 7275)	This is SetContainedControl, a member of class IRibbonContainedControl.

Public Properties

	Name	Description
	ContainedControl (see page 7275)	This is ContainedControl, a member of class IRibbonContainedControl.

77.2.2.2 IRibbonContainedControl Methods

The methods of the IRibbonContainedControl class are listed here.

Public Methods

	Name	Description
	GetContainedControl (see page 7275)	This is GetContainedControl, a member of class IRibbonContainedControl.
	SetContainedControl (see page 7275)	This is SetContainedControl, a member of class IRibbonContainedControl.

77.2.2.2.1 IRibbonContainedControl.GetContainedControl Method

Pascal

```
function GetContainedControl: TControl;
```

C++

```
__fastcall TControl * GetContainedControl();
```

Description

This is GetContainedControl, a member of class IRibbonContainedControl.

77.2.2.2.2 IRibbonContainedControl.SetContainedControl Method

Pascal

```
procedure SetContainedControl(const Value: TControl);
```

C++

```
__fastcall SetContainedControl(const TControl * Value);
```

Description

This is SetContainedControl, a member of class IRibbonContainedControl.

77.2.2.3 IRibbonContainedControl Properties

The properties of the IRibbonContainedControl class are listed here.

Public Properties

	Name	Description
	ContainedControl (see page 7275)	This is ContainedControl, a member of class IRibbonContainedControl.

77.2.2.3.1 IRibbonContainedControl.ContainedControl Property

Pascal

```
property ContainedControl: TControl;
```

C++

```
__property TControl * ContainedControl;
```

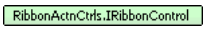
Description

This is ContainedControl, a member of class IRibbonContainedControl.

77.2.3 IRibbonControl Interface

Interface implemented by controls visible in Ribbon (see page 6713) Groups.

Class Hierarchy



Pascal

```
IRibbonControl = interface;
```

C++

```
__interface IRibbonControl;
```

File

RibbonActnCtrls

Description

IRibbonControl is an interface implemented by a control that is visible in Ribbon (see page 6713) Groups.

See Also

IRibbonContainedControl (see page 7274)

IRibbonSplitButton (see page 7282)

77.2.3.1 IRibbonControl Members

The following tables list the members exposed by IRibbonControl.

Public Methods

	Name	Description
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.

77.2.3.2 IRibbonControl Methods

The methods of the IRibbonControl class are listed here.

Public Methods

	Name	Description
	CanAddToQuickAccessToolbar (see page 7276)	This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.

77.2.3.2.1 IRibbonControl.CanAddToQuickAccessToolbar Method

Pascal

```
function CanAddToQuickAccessToolbar: Boolean;
```

C++

```
__fastcall Boolean CanAddToQuickAccessToolbar();
```

Description

This is CanAddToQuickAccessToolbar, a member of class IRibbonControl.

77.2.4 IRibbonGroupDropdown Interface

Class Hierarchy

RibbonActnCtrls.IRibbonGroupDropdown

Pascal

```
IRibbonGroupDropdown = interface;
```

C++

```
__interface IRibbonGroupDropdown;
```

File

RibbonActnCtrls

Description

This is class RibbonActnCtrls.IRibbonGroupDropdown.

77.2.4.1 IRibbonGroupDropdown Members

The following tables list the members exposed by IRibbonGroupDropdown.

Public Methods

	Name	Description
	CloseUp (see page 7277)	This is CloseUp, a member of class IRibbonGroupDropdown.
	GetParentControl (see page 7278)	This is GetParentControl, a member of class IRibbonGroupDropdown.
	GetTopPosition (see page 7278)	This is GetTopPosition, a member of class IRibbonGroupDropdown.

77.2.4.2 IRibbonGroupDropdown Methods

The methods of the IRibbonGroupDropdown class are listed here.

Public Methods

	Name	Description
	CloseUp (see page 7277)	This is CloseUp, a member of class IRibbonGroupDropdown.
	GetParentControl (see page 7278)	This is GetParentControl, a member of class IRibbonGroupDropdown.
	GetTopPosition (see page 7278)	This is GetTopPosition, a member of class IRibbonGroupDropdown.

77.2.4.2.1 IRibbonGroupDropdown.CloseUp Method

Pascal

```
procedure CloseUp;
```

C++

```
__fastcall CloseUp();
```

Description

This is CloseUp, a member of class IRibbonGroupDropdown.

77.2.4.2.2 IRibbonGroupDropdown.GetParentControl Method

Pascal

```
function GetParentControl: TWinControl;
```

C++

```
__fastcall TWinControl * GetParentControl();
```

Description

This is GetParentControl, a member of class IRibbonGroupDropdown.

77.2.4.2.3 IRibbonGroupDropdown.GetTopPosition Method

Pascal

```
function GetTopPosition: Integer;
```

C++

```
__fastcall int GetTopPosition();
```

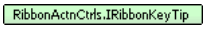
Description

This is GetTopPosition, a member of class IRibbonGroupDropdown.

77.2.5 IRibbonKeyTip Interface

Commands with a **KeyTip (see page 7282)** should implement this interface.

Class Hierarchy



Pascal

```
IRibbonKeyTip = interface;
```

C++

```
__interface IRibbonKeyTip;
```

File

RibbonActnCtrls

Description

Commands that have a **KeyTip (see page 7282)** should implement IRibbonKeyTip.

See Also

Chapter Title

Procedure Title

ClassName

PropertyName

MethodName

EventName

RoutineName

TypeName

VariableName

ConstantName

77.2.5.1 IRibbonKeyTip Members

The following tables list the members exposed by IRibbonKeyTip.

Public Methods

	Name	Description
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.2.5.2 IRibbonKeyTip Methods

The methods of the IRibbonKeyTip class are listed here.

Public Methods

	Name	Description
	GetKeyTip (see page 7279)	Returns current **KeyTip (see page 7282)** for command.
	HandleKeyTip (see page 7280)	Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).
	HideKeyTip (see page 7280)	Hide **KeyTip (see page 7282)** for command.
	KeyTipClick (see page 7281)	Handle clicking a **KeyTip (see page 7282)**.
	SetKeyTip (see page 7281)	Sets **KeyTip (see page 7282)** for command.
	ShowKeyTip (see page 7281)	Shows **KeyTip (see page 7282)** for command in specified position.

77.2.5.2.1 IRibbonKeyTip.GetKeyTip Method

Returns current **KeyTip (see page 7282)** for command.

Pascal

```
function GetKeyTip: string;
```

C++

```
__fastcall UnicodeString GetKeyTip();
```

Description

GetKeyTip returns the current **KeyTip (see page 7282)** for the command

See Also

KeyTip (see page 7282)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.2.5.2.2 IRibbonKeyTip.HandleKeyTip Method

Indicates if command's **KeyTip (see page 7282)** matches currently input KeyTip (see page 7282).

Pascal

```
procedure HandleKeyTip(const KeyTip: string; var Handled: TKeyTipHandled);
```

C++

```
__fastcall HandleKeyTip(const UnicodeString KeyTip, TKeyTipHandled Handled);
```

Description

HandleKeyTip indicates whether this command's

KeyTip (see page 7282) matches the currently input KeyTip (see page 7282) by the user. Full and partial matches are taken into consideration.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.2.5.2.3 IRibbonKeyTip.HideKeyTip Method

Hide **KeyTip (see page 7282)** for command.

Pascal

```
procedure HideKeyTip(const Match: string);
```

C++

```
__fastcall HideKeyTip(const UnicodeString Match);
```

Description

This method hides the **KeyTip (see page 7282)** for this command. If the **Match** parameter matches (full or partial) the command's **KeyTip (see page 7282)**, the **KeyTip (see page 7282)** remains visible.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

SetKeyTip (see page 7281)

ShowKeyTip (see page 7281)

77.2.5.2.4 IRibbonKeyTip.KeyTipClick Method

Handle clicking a **KeyTip (see page 7282)**.

Pascal

```
procedure KeyTipClick;
```

C++

```
__fastcall KeyTipClick();
```

Description

KeyTipClick Handle clicking a **KeyTip (see page 7282)**. Some requires need to be handled differently when clicked by a **KeyTip (see page 7282)**.

77.2.5.2.5 IRibbonKeyTip.SetKeyTip Method

Sets **KeyTip (see page 7282)** for command.

Pascal

```
procedure SetKeyTip(const Value: string);
```

C++

```
__fastcall SetKeyTip(const UnicodeString Value);
```

Description

This method sets the **KeyTip (see page 7282)** for the command.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

ShowKeyTip (see page 7281)

77.2.5.2.6 IRibbonKeyTip.ShowKeyTip Method

Shows **KeyTip (see page 7282)** for command in specified position.

Pascal

```
procedure ShowKeyTip(const Position: TKeyTipPosition);
```

C++

```
__fastcall ShowKeyTip(const TKeyTipPosition Position);
```

Description

This method shows the **KeyTip (see page 7282)** for the command in the position specified. Only the Application (see page 2612) Button uses a **KeyTip (see page 7282)** position.

See Also

KeyTip (see page 7282)

GetKeyTip (see page 7279)

HandleKeyTip (see page 7280)

HideKeyTip (see page 7280)

SetKeyTip (see page 7281)

77.2.5.3 IRibbonKeyTip Properties

The properties of the IRibbonKeyTip class are listed here.

Public Properties

	Name	Description
	KeyTip (see page 7282)	Current **KeyTip** for command

77.2.5.3.1 IRibbonKeyTip.KeyTip Property

Current **KeyTip** for command

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

KeyTip is the current **KeyTip** for the command

See Also

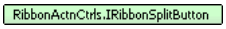
GetKeyTip (see page 7279)

SetKeyTip (see page 7281)

77.2.6 IRibbonSplitButton Interface

Interface implemented by controls that have split button.

Class Hierarchy



Pascal

```
IRibbonSplitButton = interface;
```

C++

```
__interface IRibbonSplitButton;
```

File

RibbonActnCtrls

Description

IRibbonSplitButton is an interface implemented by controls that have a split button.

See Also

IRibbonContainedControl (see page 7274)

IRibbonControl (see page 7276)

77.2.6.1 IRibbonSplitButton Members

The following tables list the members exposed by IRibbonSplitButton.

Public Methods

	Name	Description
	UpdateSplitBounds (see page 7283)	This is UpdateSplitBounds, a member of class IRibbonSplitButton.

77.2.6.2 IRibbonSplitButton Methods

The methods of the IRibbonSplitButton class are listed here.

Public Methods

	Name	Description
	UpdateSplitBounds (see page 7283)	This is UpdateSplitBounds, a member of class IRibbonSplitButton.

77.2.6.2.1 IRibbonSplitButton.UpdateSplitBounds Method

Pascal

```
procedure UpdateSplitBounds;
```

C++

```
__fastcall UpdateSplitBounds();
```

Description

This is UpdateSplitBounds, a member of class IRibbonSplitButton.

77.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DrawLine (see page 7284)	This is function RibbonActnCtrls.DrawLine.
	IsLeftMouseButtonPressed (see page 7284)	This is function RibbonActnCtrls.IsLeftMouseButtonPressed.
	ShowKeyTip (see page 7284)	Show **KeyTip** for control.
	ShowKeyTip (see page 7284)	Show **KeyTip** for control.

77.3.1 RibbonActnCtrls.DrawLine Function

Pascal

```
procedure DrawLine(ACanvas: TCanvas; const AFromPt: TPoint; const AToPt: TPoint);
```

C++

```
DrawLine(TCanvas ACanvas, const TPoint AFromPt, const TPoint AToPt);
```

File

RibbonActnCtrls

Description

This is function RibbonActnCtrls.DrawLine.

77.3.2 RibbonActnCtrls.IsLeftMouseButtonPressed Function

Pascal

```
function IsLeftMouseButtonPressed(Control: TControl): Boolean;
```

C++

```
Boolean IsLeftMouseButtonPressed(TControl * Control);
```

File

RibbonActnCtrls

Description

This is function RibbonActnCtrls.IsLeftMouseButtonPressed.

77.3.3 RibbonActnCtrls.ShowKeyTip Function

Show **KeyTip** for control.

Pascal

```
function ShowKeyTip(const KeyTip: string; const Enabled: Boolean; const Control: TControl;
const Position: TKeyTipPosition): TCustomControl; overload;
function ShowKeyTip(const KeyTip: string; const Enabled: Boolean; const Center: Boolean; const
Rect: TRect; const Position: TKeyTipPosition): TCustomControl; overload;
```

C++

```
TCustomControl ShowKeyTip(const UnicodeString KeyTip, const Boolean Enabled, const TControl *
Control, const TKeyTipPosition Position);
TCustomControl ShowKeyTip(const UnicodeString KeyTip, const Boolean Enabled, const Boolean
Center, const TRect Rect, const TKeyTipPosition Position);
```

File

RibbonActnCtrls

Description

ShowKeyTip shows the **KeyTip** for control.

The parameters are:

- **KeyTip**: The **KeyTip** to display
- **Enabled**: If the **KeyTip** should have an enabled appearance
- **Control**: The control that the **KeyTip** is for. The **KeyTip** displays close to the control
- **Position** - The position of the **KeyTip**. The position of the **KeyTip** can be either **ktpCenter** or **ktpBottom**. Most **KeyTip**s are displayed with the bottom third of the control covered with the **KeyTip**.

77.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TKeyTipHandled (see page 7286)	Validate if command's **KeyTip** matches current user input.
	TKeyTipMatchType (see page 7287)	Type of match command's **KeyTip** is for current **KeyTip** value.
	TKeyTipPosition (see page 7287)	Position of KeyTips.
	TKeyTipState (see page 7288)	Current state of KeyTips.
	TQATScrollMode (see page 7289)	This is record RibbonActnCtrls.TQATScrollMode.
	TSpinButtonStyle (see page 7290)	This is record RibbonActnCtrls.TSpinButtonStyle.

77.4.1 RibbonActnCtrls.TKeyTipHandled Enumeration

Validate if command's **KeyTip** matches current user input.

Pascal

```
TKeyTipHandled = (
  ktNone,
  ktPartial,
  ktFull
);
```

C++

```
enum TKeyTipHandled {
  ktNone,
  ktPartial,
  ktFull
};
```

File

RibbonActnCtrls

Description

Use TKeyTipHandled to validate if a command's **KeyTip** matches the current user input. It can have one of the following values:

Value	Meaning
ktNone	Current KeyTip didn't match the command's **KeyTip**.
ktPartial	Current **KeyTip** matches the first character of the commands **KeyTip**.
ktFull	Current **KeyTip** is a full match for the command. The source **KeyTip** could be either one character or two characters.

See Also

TKeyTipMatchType (see page 7287)

TKeyTipPosition (see page 7287)

TKeyTipState (see page 7288)

77.4.2 RibbonActnCtrls.TKeyTipMatchType Enumeration

Type of match command's **KeyTip** is for current **KeyTip** value.

Pascal

```
TKeyTipMatchType = (
  mtApplicationMenu,
  mtTabs,
  mtControl,
  mtGroup,
  mtCollapsed,
  mtDialogButton
);
```

C++

```
enum TKeyTipMatchType {
  mtApplicationMenu,
  mtTabs,
  mtControl,
  mtGroup,
  mtCollapsed,
  mtDialogButton
};
```

File

RibbonActnCtrls

Description

TKeyTipMatchType is the type of match the command's **KeyTip** is for the current **KeyTip** value. It can have one of the following values:

Value	Meaning
mtApplicationMenu	For ApplicationMenu.
mtTabs	For Tabs.
mtControl	For a Control.
mtGroup	For a Group.
mtCollapsed	For a Collapsed item.
mtDialogButton	For a Dialog Button.

See Also

TKeyTipHandled (see page 7286)

TKeyTipPosition (see page 7287)

TKeyTipState (see page 7288)

77.4.3 RibbonActnCtrls.TKeyTipPosition Enumeration

Position of KeyTips.

Pascal

```
TKeyTipPosition = (
  ktpApplicationButton,
  ktpQuickAccessToolbar,
  ktpTabs,
  ktpGroupA,
  ktpGroupB,
  ktpGroupC,
  ktpGroupD
);
```

C++

```
enum TKeyTipPosition {
  ktpApplicationButton,
  ktpQuickAccessToolbar,
  ktpTabs,
  ktpGroupA,
  ktpGroupB,
  ktpGroupC,
  ktpGroupD
};
```

File

RibbonActnCtrls

Description

TKeyTipPosition is the position of KeyTips. It can have one of the following values:

Value	Meaning
ktpApplicationButton	On Application (see page 2612) Button.
ktpQuickAccessToolbar	On Quick Access Toolbar.
ktpTabs	On tabs.
ktpGroupA	On Group A
ktpGroupB	On Group B
ktpGroupC	On Group C
ktpGroupD	On Group D

See Also

TKeyTipHandled (see page 7286)

TKeyTipMatchType (see page 7287)

TKeyTipState (see page 7288)

77.4.4 RibbonActnCtrls.TKeyTipState Enumeration

Current state of KeyTips.

Pascal

```
TKeyTipState = (
  ktsTabs,
  ktsControls,
```

```
    ktsCollapsedGroup
  );
```

C++

```
  enum TKeyTipState {
    ktsTabs,
    ktsControls,
    ktsCollapsedGroup
  };
```

File

RibbonActnCtrls

Description

TKeyTipState is the current state KeyTips are in. It can have one of the following values:

Value	Meaning
ktsTabs	Initial state where the user hasn't yet selected a Tab to display the `KeyTips` for. In this state, `KeyTips` are displayed for the Application (see page 2612) Button, Quick Access Toolbar and the Tabs.
ktsControls	`KeyTips` are displayed for all commands on the Active Tab. `KeyTips` are also displayed for Groups dialog buttons and collapsed group buttons.
ktsCollapsedGroup	KeyTips are being displayed for commands that are on a collapsed group.

See Also

TKeyTipHandled (see page 7286)

TKeyTipMatchType (see page 7287)

TKeyTipPosition (see page 7287)

77.4.5 RibbonActnCtrls.TQATScrollMode Enumeration

Pascal

```
  TQATScrollMode = (
    qsCustomize,
    qsMoreControls
  );
```

C++

```
  enum TQATScrollMode {
    qsCustomize,
    qsMoreControls
  };
```

File

RibbonActnCtrls

Description

This is record RibbonActnCtrls.TQATScrollMode.

77.4.6 RibbonActnCtrls.TSpinButtonStyle Enumeration

Pascal

```
TSpinButtonStyle = (
  sbsUp,
  sbsDown
);
```

C++

```
enum TSpinButtonStyle {
  sbsUp,
  sbsDown
};
```

File

RibbonActnCtrls

Description

This is record RibbonActnCtrls.TSpinButtonStyle.

77.5 Types

The following table lists types in this documentation.

Types

Name	Description
TNumGlyphs (see page 7291)	This is type RibbonActnCtrls.TNumGlyphs.
TTimeBtnState (see page 7291)	This is type RibbonActnCtrls.TTimeBtnState.

77.5.1 RibbonActnCtrls.TNumGlyphs Type

Pascal

```
TNumGlyphs = Buttons.TNumGlyphs;
```

C++

```
Buttons.TNumGlyphs TNumGlyphs;
```

File

RibbonActnCtrls

Description

This is type RibbonActnCtrls.TNumGlyphs.

77.5.2 RibbonActnCtrls.TTimeBtnState Type

Pascal

```
TTimeBtnState = set of (tbFocusRect, tbAllowTimer);
```

C++

```
(tbFocusRect, tbAllowTimer) TTimeBtnState;
```

File

RibbonActnCtrls

Description

This is type RibbonActnCtrls.TTimeBtnState.

77.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
UM_FADECONTROL (see page 7292)	This is constant RibbonActnCtrls.UM_FADECONTROL.

77.6.1 RibbonActnCtrls.UM_FADECONTROL Constant

Pascal

```
UM_FADECONTROL = WM_USER + 100;
```

C++

```
UM_FADECONTROL = WM_USER + 100;
```

File

RibbonActnCtrls

Description

This is constant RibbonActnCtrls.UM_FADECONTROL.

78 RibbonActnMenus Namespace

This is namespace RibbonActnMenus.

Topics

Name	Description
Classes (see page 7294)	The following table lists classes in this documentation.

Classes

	Name	Description
	TOptionItem (see page 7294)	Item added to OptionItems collection for TRibbonApplicationMenuBar (see page 7302).
	TOptionItems (see page 7298)	Stores additional option items appearing in border of Application Menu.
	TRibbonActionPopupMenu (see page 7300)	Display top level menus when using Ribbon (see page 6713) Style without Ribbon (see page 6713) interface.
	TRibbonApplicationMenuBar (see page 7302)	Display Application Menu for Ribbon (see page 6713).
	TRibbonApplicationPopupMenu (see page 7321)	Ddisplay popup menu when Application Button clicked.
	TRibbonQATCustomizeMenu (see page 7328)	This is class RibbonActnMenus.TRibbonQATCustomizeMenu.
	TRibbonRecentDocuments (see page 7330)	This is class RibbonActnMenus.TRibbonRecentDocuments.
	TRibbonStyleMainPopupMenu (see page 7335)	This is class RibbonActnMenus.TRibbonStyleMainPopupMenu.
	TRibbonStylePopupMenu (see page 7336)	This is class RibbonActnMenus.TRibbonStylePopupMenu.

78.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOptionItem (see page 7294)	Item added to OptionItems collection for TRibbonApplicationMenuBar (see page 7302).
	TOptionItems (see page 7298)	Stores additional option items appearing in border of Application Menu.
	TRibbonActionPopupMenu (see page 7300)	Display top level menus when using Ribbon (see page 6713) Style without Ribbon (see page 6713) interface.
	TRibbonApplicationMenuBar (see page 7302)	Display Application Menu for Ribbon (see page 6713).
	TRibbonApplicationPopupMenu (see page 7321)	Ddisplay popup menu when Application Button clicked.
	TRibbonQATCustomizeMenu (see page 7328)	This is class RibbonActnMenus.TRibbonQATCustomizeMenu.
	TRibbonRecentDocuments (see page 7330)	This is class RibbonActnMenus.TRibbonRecentDocuments.
	TRibbonStyleMainPopupMenu (see page 7335)	This is class RibbonActnMenus.TRibbonStyleMainPopupMenu.
	TRibbonStylePopupMenu (see page 7336)	This is class RibbonActnMenus.TRibbonStylePopupMenu.

78.1.1 TOptionItem Class

Item added to OptionItems collection for TRibbonApplicationMenuBar (see page 7302).

Class Hierarchy

Pascal

```
TOptionItem = class(TCollectionItem);
```

C++

```
class TOptionItem : public TCollectionItem;
```

File

RibbonActnMenus

Description

TOptionItem is an item added to the OptionItems collection for the TRibbonApplicationMenuBar (see page 7302) component. This class represents the action to use for the command as well as the **KeyTip (see page 7297)** to use.

See Also

TRibbonApplicationMenuBar (see page 7302)

OptionItems (see page 7316)

78.1.1.1 TOptionItem Members

The following tables list the members exposed by TOptionItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 7296)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 7295)	Copies the contents of another, similar object.

Published Properties

	Name	Description
	Action (see page 7296)	This is Action, a member of class TOptionItem.
	Caption (see page 7297)	This is Caption, a member of class TOptionItem.
	Hint (see page 7297)	This is Hint, a member of class TOptionItem.
	KeyTip (see page 7297)	This is KeyTip, a member of class TOptionItem.
	Tag (see page 7297)	This is Tag, a member of class TOptionItem.

78.1.1.2 TOptionItem Methods

The methods of the TOptionItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 7296)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 7295)	Copies the contents of another, similar object.

78.1.1.2.1 TOptionItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
```

```
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

78.1.1.2.2 TOptionItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

78.1.1.3 TOptionItem Properties

The properties of the TOptionItem class are listed here.

Published Properties

	Name	Description
	Action (see page 7296)	This is Action, a member of class TOptionItem.
	Caption (see page 7297)	This is Caption, a member of class TOptionItem.
	Hint (see page 7297)	This is Hint, a member of class TOptionItem.
	KeyTip (see page 7297)	This is KeyTip, a member of class TOptionItem.
	Tag (see page 7297)	This is Tag, a member of class TOptionItem.

78.1.1.3.1 TOptionItem.Action Property

Pascal

```
property Action: TContainedAction;
```

C++

```
__property TContainedAction Action;
```

Description

This is Action, a member of class TOptionItem.

78.1.1.3.2 TOptionItem.Caption Property

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

This is Caption, a member of class TOptionItem.

78.1.1.3.3 TOptionItem.Hint Property

Pascal

```
property Hint: string;
```

C++

```
__property UnicodeString Hint;
```

Description

This is Hint, a member of class TOptionItem.

78.1.1.3.4 TOptionItem.KeyTip Property

Pascal

```
property KeyTip: string;
```

C++

```
__property UnicodeString KeyTip;
```

Description

This is KeyTip, a member of class TOptionItem.

78.1.1.3.5 TOptionItem.Tag Property

Pascal

```
property Tag: Integer;
```

C++

```
__property int Tag;
```

Description

This is Tag, a member of class TOptionItem.

78.1.2 TOptionItems Class

Stores additional option items appearing in border of Application Menu.

Class Hierarchy

Pascal

```
TOptionItems = class(TOwnedCollection);
```

C++

```
class TOptionItems : public TOwnedCollection;
```

File

RibbonActnMenus

Description

TOptionItems stores the additional option items that appear in the border of the Application Menu.

See Also

TRibbonApplicationMenuBar (see page 7302)

78.1.2.1 TOptionItems Members

The following tables list the members exposed by TOptionItems.

Public Methods

	Name	Description
	Add (see page 7298)	Creates a new TCollectionItem instance and adds it to the Items array.
	Insert (see page 7299)	Creates a new TCollectionItem instance and adds it to the Items array.

Public Properties

	Name	Description
R	ApplicationMenu (see page 7299)	This is ApplicationMenu, a member of class TOptionItems.
	Items (see page 7300)	Lists the items in the collection.

78.1.2.2 TOptionItems Methods

The methods of the TOptionItems class are listed here.

Public Methods

	Name	Description
	Add (see page 7298)	Creates a new TCollectionItem instance and adds it to the Items array.
	Insert (see page 7299)	Creates a new TCollectionItem instance and adds it to the Items array.

78.1.2.2.1 TOptionItems.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TOptionItem;
```

C++

```
__fastcall TOptionItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items

Insert

Create

Delete

78.1.2.2.2 TOptionItems.Insert Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Insert(Index: Integer): TOptionItem;
```

C++

```
__fastcall TOptionItem Insert(int Index);
```

Description

Call Insert to add a new item at a specified position in the collection. Existing items (starting from the specified position) are moved up in the Items array.

Insert returns the new collection item.

See Also

Add

Items

78.1.2.3 TOptionItems Properties

The properties of the TOptionItems class are listed here.

Public Properties

	Name	Description
	ApplicationMenu (see page 7299)	This is ApplicationMenu, a member of class TOptionItems.
	Items (see page 7300)	Lists the items in the collection.

78.1.2.3.1 TOptionItems.ApplicationMenu Property

Pascal

```
property ApplicationMenu: TRibbonApplicationMenuBar;
```

C++

```
__property TRibbonApplicationMenuBar ApplicationMenu;
```

Description

This is ApplicationMenu, a member of class TOptionItems.

78.1.2.3.2 TOptionItems.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TOptionItem;
```

C++

```
__property TOptionItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count

ItemClass

Index

Add

Insert

78.1.3 TRibbonActionPopupMenu Class

Display top level menus when using Ribbon (see page 6713) Style without Ribbon (see page 6713) interface.

Class Hierarchy

Pascal

```
TRibbonActionPopupMenu = class(TXPStylePopupMenu);
```

C++

```
class TRibbonActionPopupMenu : public TXPStylePopupMenu;
```

File

RibbonActnMenus

Description

TRibbonActionPopupMenu displays top level menus when using the Ribbon (see page 6713) Style without the Ribbon (see page 6713) interface.

78.1.3.1 TRibbonActionPopupMenu Members

The following tables list the members exposed by TRibbonActionPopupMenu.

Protected Methods

	Name	Description
	NCPaint (see page 7301)	This is NCPaint, a member of class TRibbonActionPopupMenu.

Public Methods

	Name	Description
	CreateControl (see page 7301)	This is CreateControl, a member of class TRibbonActionPopupMenu.
	TrackMenu (see page 7302)	Executes the action associated with the menu item.

Public Properties

	Name	Description
	SelectFirstItem (see page 7302)	This is SelectFirstItem, a member of class TRibbonActionPopupMenu.

78.1.3.2 TRibbonActionPopupMenu Methods

The methods of the TRibbonActionPopupMenu class are listed here.

Protected Methods

	Name	Description
	NCPaint (see page 7301)	This is NCPaint, a member of class TRibbonActionPopupMenu.

Public Methods

	Name	Description
	CreateControl (see page 7301)	This is CreateControl, a member of class TRibbonActionPopupMenu.
	TrackMenu (see page 7302)	Executes the action associated with the menu item.

78.1.3.2.1 TRibbonActionPopupMenu.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TRibbonActionPopupMenu.

78.1.3.2.2 TRibbonActionPopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TRibbonActionPopupMenu.

78.1.3.2.3 TRibbonActionPopupMenu.TrackMenu Method

Executes the action associated with the menu item.

Pascal

```
procedure TrackMenu; override;
```

C++

```
virtual __fastcall TrackMenu();
```

Description

TrackMenu is called internally to execute the action associated with the selected menu item..

78.1.3.3 TRibbonActionPopupMenu Properties

The properties of the TRibbonActionPopupMenu class are listed here.

Public Properties

	Name	Description
	SelectFirstItem (see page 7302)	This is SelectFirstItem, a member of class TRibbonActionPopupMenu.

78.1.3.3.1 TRibbonActionPopupMenu.SelectFirstItem Property

Pascal

```
property SelectFirstItem: Boolean;
```

C++

```
__property Boolean SelectFirstItem;
```

Description

This is SelectFirstItem, a member of class TRibbonActionPopupMenu.

78.1.4 TRibbonApplicationMenuBar Class

Display Application Menu for Ribbon (see page 6713).

Class Hierarchy

Pascal

```
TRibbonApplicationMenuBar = class(TCustomActionMainMenuBar);
```

C++

```
class TRibbonApplicationMenuBar : public TCustomActionMainMenuBar;
```

File

RibbonActnMenus

Description

TRibbonApplicationMenuBar is used to display the Application Menu for the Ribbon (see page 6713).

The TRibbonApplicationMenuBar component is not available on the component palette and should not be created dynamically. To add an Application Menu to your Ribbon (see page 6713), select the Add Application Menu command in the Ribbon (see page 6713) component editor.

The Application Menu consists of two vertical panes. On the left is a list of Menu items. You can modify the menu items at design time.

On the right is a place to display Recent items. This area is mostly intended for use as a recently used file list. To use this as a Recent Items pane, use the TRibbon.AddRecentItem method to add items. There are a number of Recent Item functions to help you setup the list of items. You can also use the right pane to add more commands. Do this by setting the TApplicationMenu.CommandType to **ctCommand**. See the CommandType property for more details.

The bottom section of the Application button can also display commands. Usually this section is set aside for the Application Exit command and Application Options command. Set your commands in the OptionItems (see page 7316) collection.

To modify the contents of the Application Menu, you can drag actions from the customize dialog and drop them in the menu on the left of the application menu.

See Also

TApplicationMenu (see page 6716)

TRibbon (see page 6845)

CommandType (see page 6720)

AddRecentItem (see page 6735)

78.1.4.1 TRibbonApplicationMenuBar Members

The following tables list the members exposed by TRibbonApplicationMenuBar.

Protected Methods

	Name	Description
	CreateControls (see page 7306)	This is CreateControls, a member of class TRibbonApplicationMenuBar.
	DragOver (see page 7306)	OnDragOver event dispatcher.
	DrawBackground (see page 7306)	This is DrawBackground, a member of class TRibbonApplicationMenuBar.
	GetControlClass (see page 7306)	This is GetControlClass, a member of class TRibbonApplicationMenuBar.
	NewPopup (see page 7307)	This is NewPopup, a member of class TRibbonApplicationMenuBar.
	Notification (see page 7307)	Responds to notifications that components are being created or destroyed.
	Paint (see page 7308)	This is Paint, a member of class TRibbonApplicationMenuBar.
	RibbonStyle (see page 7308)	This is RibbonStyle, a member of class TRibbonApplicationMenuBar.
	ShowKeyTips (see page 7309)	This is ShowKeyTips, a member of class TRibbonApplicationMenuBar.
	WMSysCommand (see page 7309)	This is WMSysCommand, a member of class TRibbonApplicationMenuBar.

Public Methods

	Name	Description
	CloseMenu (see page 7305)	Closes the menu currently being displayed.
	Create (see page 7304)	Creates and initializes a TCustomActionMainMenuBar instance.
	CreateControl (see page 7305)	This is CreateControl, a member of class TRibbonApplicationMenuBar.
	Destroy (see page 7304)	Destroys an instance of TCustomActionMainMenuBar.
	GetRibbon (see page 7307)	This is GetRibbon, a member of class TRibbonApplicationMenuBar.
	SelectApplicationButton (see page 7308)	This is SelectApplicationButton, a member of class TRibbonApplicationMenuBar.
	SetBounds (see page 7308)	Sets the bounds based on whether the popup is expanded or not
	SetRibbon (see page 7309)	This is SetRibbon, a member of class TRibbonApplicationMenuBar.

Public Properties

	Name	Description
	Align (see page 7310)	Determines how the control aligns within its container (parent control).
	HorzMargin (see page 7315)	Specifies the width of the horizontal margins in pixels.
	Spacing (see page 7316)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	VertMargin (see page 7318)	Specifies the width of the vertical margins in pixels.
R	VisibleMenu (see page 7319)	This is VisibleMenu, a member of class TRibbonApplicationMenuBar.
R	VisibleMenuCount (see page 7319)	This is VisibleMenuCount, a member of class TRibbonApplicationMenuBar.

Published Properties

	Name	Description
	ActionManager (see page 7310)	Specifies the action manager component that is associated with the action band.
	Height (see page 7314)	Specifies the vertical size of the control in pixels.
	Left (see page 7315)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	OptionItems (see page 7316)	Commands displayed at bottom of Application Menu.
	RecentItems (see page 7316)	Recent items displayed in right pane of Application.
	Top (see page 7317)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 7319)	Specifies the horizontal size of the control or form in pixels.

78.1.4.2 TRibbonApplicationMenuBar.Create Constructor

Creates and initializes a TCustomActionMainMenuBar instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionMainMenuBar and initializes its properties.

```
__fastcall virtual TCustomActionMainMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMainMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

78.1.4.3 TRibbonApplicationMenuBar.Destroy Destructor

Destroys an instance of TCustomActionMainMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonApplicationMenuBar();
```

Description

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

See Also

System::TObject::Free

Destroy

78.1.4.4 TRibbonApplicationMenuBar Methods

The methods of the TRibbonApplicationMenuBar class are listed here.

Protected Methods

	Name	Description
	CreateControls (see page 7306)	This is CreateControls, a member of class TRibbonApplicationMenuBar.
	DragOver (see page 7306)	OnDragOver event dispatcher.
	DrawBackground (see page 7306)	This is DrawBackground, a member of class TRibbonApplicationMenuBar.
	GetControlClass (see page 7306)	This is GetControlClass, a member of class TRibbonApplicationMenuBar.
	NewPopup (see page 7307)	This is NewPopup, a member of class TRibbonApplicationMenuBar.
	Notification (see page 7307)	Responds to notifications that components are being created or destroyed.
	Paint (see page 7308)	This is Paint, a member of class TRibbonApplicationMenuBar.
	RibbonStyle (see page 7308)	This is RibbonStyle, a member of class TRibbonApplicationMenuBar.
	ShowKeyTips (see page 7309)	This is ShowKeyTips, a member of class TRibbonApplicationMenuBar.
	WMSysCommand (see page 7309)	This is WMSysCommand, a member of class TRibbonApplicationMenuBar.

Public Methods

	Name	Description
	CloseMenu (see page 7305)	Closes the menu currently being displayed.
	CreateControl (see page 7305)	This is CreateControl, a member of class TRibbonApplicationMenuBar.
	GetRibbon (see page 7307)	This is GetRibbon, a member of class TRibbonApplicationMenuBar.
	SelectApplicationButton (see page 7308)	This is SelectApplicationButton, a member of class TRibbonApplicationMenuBar.
	SetBounds (see page 7308)	Sets the bounds based on whether the popup is expanded or not
	SetRibbon (see page 7309)	This is SetRibbon, a member of class TRibbonApplicationMenuBar.

78.1.4.4.1 TRibbonApplicationMenuBar.CloseMenu Method

Closes the menu currently being displayed.

Pascal

```
procedure CloseMenu; override;
```

C++

```
virtual __fastcall CloseMenu();
```

Description

CloseMenu is called to close the menu currently being displayed, including any submenus that may be open.

78.1.4.4.2 TRibbonApplicationMenuBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TRibbonApplicationMenuBar.

78.1.4.4.3 TRibbonApplicationMenuBar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TRibbonApplicationMenuBar.

78.1.4.4.4 TRibbonApplicationMenuBar.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop

OnDragOver

78.1.4.4.5 TRibbonApplicationMenuBar.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TRibbonApplicationMenuBar.

78.1.4.4.6 TRibbonApplicationMenuBar.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TRibbonApplicationMenuBar.

78.1.4.4.7 TRibbonApplicationMenuBar.GetRibbon Method

Pascal

```
function GetRibbon: TWinControl;
```

C++

```
__fastcall TWinControl * GetRibbon();
```

Description

This is GetRibbon, a member of class TRibbonApplicationMenuBar.

78.1.4.4.8 TRibbonApplicationMenuBar.NewPopup Method

Pascal

```
function NewPopup: TCustomActionPopupMenu; override;
```

C++

```
virtual __fastcall TCustomActionPopupMenu NewPopup();
```

Description

This is NewPopup, a member of class TRibbonApplicationMenuBar.

78.1.4.4.9 TRibbonApplicationMenuBar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

78.1.4.4.10 TRibbonApplicationMenuBar.Paint Method

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

This is Paint, a member of class TRibbonApplicationMenuBar.

78.1.4.4.11 TRibbonApplicationMenuBar.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonApplicationMenuBar.

78.1.4.4.12 TRibbonApplicationMenuBar.SelectApplicationButton Method

Pascal

```
procedure SelectApplicationButton;
```

C++

```
__fastcall SelectApplicationButton();
```

Description

This is SelectApplicationButton, a member of class TRibbonApplicationMenuBar.

78.1.4.4.13 TRibbonApplicationMenuBar.SetBounds Method

Sets the bounds based on whether the popup is expanded or not

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

SetBounds calls the inherited method after altering the AWidth and AHeight parameters (if necessary) to include the width and height of the expanded popup menu. For more information about SetBounds, see the documentation for the inherited method.

See Also

SetBounds

78.1.4.4.14 TRibbonApplicationMenuBar.SetRibbon Method

Pascal

```
procedure SetRibbon(const Value: TWinControl);
```

C++

```
__fastcall SetRibbon(const TWinControl * Value);
```

Description

This is SetRibbon, a member of class TRibbonApplicationMenuBar.

78.1.4.4.15 TRibbonApplicationMenuBar.ShowKeyTips Method

Pascal

```
procedure ShowKeyTips;
```

C++

```
__fastcall ShowKeyTips();
```

Description

This is ShowKeyTips, a member of class TRibbonApplicationMenuBar.

78.1.4.4.16 TRibbonApplicationMenuBar.WMSysCommand Method

Pascal

```
procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
```

C++

```
__fastcall WMSysCommand(TWMSysCommand Message);
```

Description

This is WMSysCommand, a member of class TRibbonApplicationMenuBar.

78.1.4.5 TRibbonApplicationMenuBar Properties

The properties of the TRibbonApplicationMenuBar class are listed here.

Public Properties

	Name	Description
	Align (see page 7310)	Determines how the control aligns within its container (parent control).
	HorzMargin (see page 7315)	Specifies the width of the horizontal margins in pixels.
	Spacing (see page 7316)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	VertMargin (see page 7318)	Specifies the width of the vertical margins in pixels.
R	VisibleMenu (see page 7319)	This is VisibleMenu, a member of class TRibbonApplicationMenuBar.
R	VisibleMenuCount (see page 7319)	This is VisibleMenuCount, a member of class TRibbonApplicationMenuBar.

Published Properties

	Name	Description
	ActionManager (see page 7310)	Specifies the action manager component that is associated with the action band.
	Height (see page 7314)	Specifies the vertical size of the control in pixels.

	Left (see page 7315)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	OptionItems (see page 7316)	Commands displayed at bottom of Application Menu.
	RecentItems (see page 7316)	Recent items displayed in right pane of Application.
	Top (see page 7317)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 7319)	Specifies the horizontal size of the control or form in pixels.

78.1.4.5.1 TRibbonApplicationMenuBar.ActionManager Property

Specifies the action manager component that is associated with the action band.

Pascal

```
property ActionManager;
```

C++

```
__property ActionManager;
```

Description

Use ActionManager to access the action manager that uses this action band to render UI elements it manages. Typically, this association is created when you drag actions from the action manager and drop them on the action bar. When an action is dropped on an action band and the ActionManager property is not assigned, the ActionManager property is automatically assigned to the action manager that manages the action that was dropped on the action band.

At runtime, you can perform the equivalent operation to dragging an action onto the action band by setting the ActionBar property of one of the action clients that the action manager's ActionBars property lists so that it specifies this component.

See Also

ActionBars

ActionClient

ActionBar

78.1.4.5.2 TRibbonApplicationMenuBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

78.1.4.5.3 TRibbonApplicationMenuBar.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
```

```
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

78.1.4.5.4 TRibbonApplicationMenuBar.HorzMargin Property

Specifies the width of the horizontal margins in pixels.

Pascal

```
property HorzMargin;
```

C++

```
__property HorzMargin;
```

Description

HorzMargin specifies the width of the action band's left and right horizontal margins in pixels.

See Also

VertMargin

VertSeparator

78.1.4.5.5 TRibbonApplicationMenuBar.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left;
```

C++

```
__property Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
```

```
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top

78.1.4.5.6 TRibbonApplicationMenuBar.OptionItems Property

Commands displayed at bottom of Application Menu.

Pascal

```
property OptionItems: TOptionItems;
```

C++

```
__property TOptionItems OptionItems;
```

Description

OptionItems are displayed at the bottom of the Application Menu. Usually this section of commands has Application Exit and Application Options commands.

78.1.4.5.7 TRibbonApplicationMenuBar.RecentItems Property

Recent items displayed in right pane of Application.

Pascal

```
property RecentItems: TOptionItems;
```

C++

```
__property TOptionItems RecentItems;
```

Description

RecentItems are displayed in the right pane of the Application Menu. RecentItems can be added dynamically using one of the RecentItems helper methods. To handle the user clicking on a Recent Item, use the OnRecentItemClick event.

See Also

OnRecentItemClick (see page 6863)

78.1.4.5.8 TRibbonApplicationMenuBar.Spacing Property

Determines where the image and text appear on an action band's rendered control that represents the action client.

Pascal

```
property Spacing;
```

C++

```
__property Spacing;
```

Description

Spacing is used internally to set the number of pixels that should appear between the image specified in an action client's ImageIndex and the text specified in its Caption property.

See Also

Caption

ImageIndex

78.1.4.5.9 TRibbonApplicationMenuBar.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top;
```

C++

```
__property Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    dynamic_cast<TWinControl *>(GroupBox1->Controls[i])->Brush->Color = colorarray[i];
```

```
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Left

78.1.4.5.10 TRibbonApplicationMenuBar.VertMargin Property

Specifies the width of the vertical margins in pixels.

Pascal

```
property VertMargin;
```

C++

```
__property VertMargin;
```

Description

VertMargin specifies the width of the action band's top and bottom vertical margins in pixels.

See Also

HorzMargin

HorzSeparator

78.1.4.5.11 TRibbonApplicationMenuBar.VisibleMenu Property

Pascal

```
property VisibleMenu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar VisibleMenu;
```

Description

This is VisibleMenu, a member of class TRibbonApplicationMenuBar.

78.1.4.5.12 TRibbonApplicationMenuBar.VisibleMenuCount Property

Pascal

```
property VisibleMenuCount: Integer;
```

C++

```
__property int VisibleMenuCount;
```

Description

This is VisibleMenuCount, a member of class TRibbonApplicationMenuBar.

78.1.4.5.13 TRibbonApplicationMenuBar.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
```

```
  objects.
    TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

    // Line up the directory list box on the left of the form.
    pdlb->Parent = Form1;
    pdlb->Align = alLeft;
    pdlb->Width = Form1->ClientWidth/3;

    // Now use the splitter to divide the directory pane from the file pane.
    ps->Parent = Form1;
    ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
    ps->Align = pdlb->Align; // Give same alignment as directory list.

    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

78.1.5 TRibbonApplicationPopupMenu Class

Ddisplay popup menu when Application Button clicked.

Class Hierarchy

Pascal

```
TRibbonApplicationPopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TRibbonApplicationPopupMenu : public TCustomActionPopupMenu;
```

File

RibbonActnMenus

Description

TRibbonApplicationPopupMenu is used to display the popup menu when the Application Button has been clicked. The Application Popup Menu is a two paned menu that shows a normal list of commands on the left pane, as well as a specialized list of commands on the right pane. Additional commands can also be displayed in the bottom border of the menu.

78.1.5.1 TRibbonApplicationPopupMenu Members

The following tables list the members exposed by TRibbonApplicationPopupMenu.

Protected Methods

	Name	Description
	AlignControls (see page 7323)	Aligns any controls for which the control is the parent within a specified area of the control.
	CMTextchanged (see page 7323)	This is CMTextchanged, a member of class TRibbonApplicationPopupMenu.
	CreateControls (see page 7323)	This is CreateControls, a member of class TRibbonApplicationPopupMenu.
	CreateOptionsItems (see page 7324)	This is CreateOptionsItems, a member of class TRibbonApplicationPopupMenu.
	CreateRecentDocuments (see page 7324)	This is CreateRecentDocuments, a member of class TRibbonApplicationPopupMenu.
	CreateWnd (see page 7324)	Creates the underlying window.
	CustomAlignPosition (see page 7325)	Returns aligned position of custom-aligned child control.
	DoDropActions (see page 7325)	This is DoDropActions, a member of class TRibbonApplicationPopupMenu.
	DoDropItem (see page 7326)	This is DoDropItem, a member of class TRibbonApplicationPopupMenu.
	DrawBackground (see page 7326)	This is DrawBackground, a member of class TRibbonApplicationPopupMenu.
	GetBorderHeight (see page 7326)	This is GetBorderHeight, a member of class TRibbonApplicationPopupMenu.
	NCPaint (see page 7327)	This is NCPaint, a member of class TRibbonApplicationPopupMenu.
	PositionPopup (see page 7327)	This is PositionPopup, a member of class TRibbonApplicationPopupMenu.
	RibbonStyle (see page 7327)	This is RibbonStyle, a member of class TRibbonApplicationPopupMenu.
	SetColorMap (see page 7327)	This is SetColorMap, a member of class TRibbonApplicationPopupMenu.

Public Methods

	Name	Description
	Create (see page 7322)	Creates and initializes a TCustomActionPopupMenu instance.
	Destroy (see page 7322)	Destroys an instance of TCustomActionMenuBar.

	DoDropCategory (see page 7326)	This is DoDropCategory, a member of class TRibbonApplicationPopupMenu.
	GetRibbon (see page 7326)	This is GetRibbon, a member of class TRibbonApplicationPopupMenu.
	SetRibbon (see page 7328)	This is SetRibbon, a member of class TRibbonApplicationPopupMenu.

78.1.5.2 TRibbonApplicationPopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

78.1.5.3 TRibbonApplicationPopupMenu.Destroy Destructor

Destroys an instance of TCustomActionMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonApplicationPopupMenu();
```

Description

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

See Also

System::TObject::Free

Destroy

78.1.5.4 TRibbonApplicationPopupMenu Methods

The methods of the TRibbonApplicationPopupMenu class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 7323)	Aligns any controls for which the control is the parent within a specified area of the control.
	CMTextchanged (see page 7323)	This is CMTextchanged, a member of class TRibbonApplicationPopupMenu.
	CreateControls (see page 7323)	This is CreateControls, a member of class TRibbonApplicationPopupMenu.
	CreateOptionsItems (see page 7324)	This is CreateOptionsItems, a member of class TRibbonApplicationPopupMenu.
	CreateRecentDocuments (see page 7324)	This is CreateRecentDocuments, a member of class TRibbonApplicationPopupMenu.
	CreateWnd (see page 7324)	Creates the underlying window.

	CustomAlignPosition (see page 7325)	Returns aligned position of custom-aligned child control.
	DoDropActions (see page 7325)	This is DoDropActions, a member of class TRibbonApplicationPopupMenu.
	DoDropItem (see page 7326)	This is DoDropItem, a member of class TRibbonApplicationPopupMenu.
	DrawBackground (see page 7326)	This is DrawBackground, a member of class TRibbonApplicationPopupMenu.
	GetBorderHeight (see page 7326)	This is GetBorderHeight, a member of class TRibbonApplicationPopupMenu.
	NCPaint (see page 7327)	This is NCPaint, a member of class TRibbonApplicationPopupMenu.
	PositionPopup (see page 7327)	This is PositionPopup, a member of class TRibbonApplicationPopupMenu.
	RibbonStyle (see page 7327)	This is RibbonStyle, a member of class TRibbonApplicationPopupMenu.
	SetColorMap (see page 7327)	This is SetColorMap, a member of class TRibbonApplicationPopupMenu.

Public Methods

	Name	Description
	DoDropCategory (see page 7326)	This is DoDropCategory, a member of class TRibbonApplicationPopupMenu.
	GetRibbon (see page 7326)	This is GetRibbon, a member of class TRibbonApplicationPopupMenu.
	SetRibbon (see page 7328)	This is SetRibbon, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.1 TRibbonApplicationPopupMenu.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

78.1.5.4.2 TRibbonApplicationPopupMenu.CMTextchanged Method

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This is CMTextchanged, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.3 TRibbonApplicationPopupMenu.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.4 TRibbonApplicationPopupMenu.CreateOptionsItems Method

Pascal

```
procedure CreateOptionsItems; virtual;
```

C++

```
virtual __fastcall CreateOptionsItems();
```

Description

This is CreateOptionsItems, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.5 TRibbonApplicationPopupMenu.CreateRecentDocuments Method

Pascal

```
procedure CreateRecentDocuments; virtual;
```

C++

```
virtual __fastcall CreateRecentDocuments();
```

Description

This is CreateRecentDocuments, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.6 TRibbonApplicationPopupMenu.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

78.1.5.4.7 TRibbonApplicationPopupMenu.CustomAlignPosition Method

Returns aligned position of custom-aligned child control.

Pascal

```
procedure CustomAlignPosition(Control: TControl; var NewLeft: Integer; var NewTop: Integer;
var NewWidth: Integer; var NewHeight: Integer; var AlignRect: TRect; AlignInfo: TAlignInfo);
override;
```

C++

```
virtual __fastcall CustomAlignPosition(TControl * Control, int NewLeft, int NewTop, int
NewWidth, int NewHeight, TRect AlignRect, TAlignInfo AlignInfo);
```

Description

CustomAlignPosition determines the specific position of custom-aligned child controls. CustomAlignPosition is called once for each child control with an Align property of **alCustom**, in an order previously determined by calls to CustomAlignInsertBefore.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition.

These are the parameters:

Control	The child control being placed.
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. CustomAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. CustomAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, CustomAlignPosition does nothing.	

See Also

CustomAlignInsertBefore

Align

OnAlignPosition

TAlignInfo

78.1.5.4.8 TRibbonApplicationPopupMenu.DoDropActions Method

Pascal

```
procedure DoDropActions(Source: TActionDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropActions(TActionDragObject Source, const int X, const int Y);
```

Description

This is DoDropActions, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.9 TRibbonApplicationPopupMenu.DoDropCategory Method

Pascal

```
procedure DoDropCategory(Source: TCategoryDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropCategory(TCategoryDragObject Source, const int X, const int Y);
```

Description

This is DoDropCategory, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.10 TRibbonApplicationPopupMenu.DoDropItem Method

Pascal

```
procedure DoDropItem(Source: TActionItemDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropItem(TActionItemDragObject Source, const int X, const int Y);
```

Description

This is DoDropItem, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.11 TRibbonApplicationPopupMenu.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.12 TRibbonApplicationPopupMenu.GetBorderHeight Method

Pascal

```
function GetBorderHeight: Integer; virtual;
```

C++

```
virtual __fastcall int GetBorderHeight();
```

Description

This is GetBorderHeight, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.13 TRibbonApplicationPopupMenu.GetRibbon Method

Pascal

```
function GetRibbon: TWinControl;
```

C++

```
__fastcall TWinControl * GetRibbon();
```

Description

This is GetRibbon, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.14 TRibbonApplicationPopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.15 TRibbonApplicationPopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.16 TRibbonApplicationPopupMenu.RibbonStyle Method

Pascal

```
function RibbonStyle: TRibbonStyleActionBars;
```

C++

```
__fastcall TRibbonStyleActionBars RibbonStyle();
```

Description

This is RibbonStyle, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.17 TRibbonApplicationPopupMenu.SetColorMap Method

Pascal

```
procedure SetColorMap(const Value: TCustomActionBarColorMap); override;
```

C++

```
virtual __fastcall SetColorMap(const TCustomActionBarColorMap Value);
```

Description

This is SetColorMap, a member of class TRibbonApplicationPopupMenu.

78.1.5.4.18 TRibbonApplicationPopupMenu.SetRibbon Method

Pascal

```
procedure SetRibbon(const Value: TWinControl);
```

C++

```
__fastcall SetRibbon(const TWinControl * Value);
```

Description

This is SetRibbon, a member of class TRibbonApplicationPopupMenu.

78.1.6 TRibbonQATCustomizeMenu Class

Class Hierarchy

Pascal

```
TRibbonQATCustomizeMenu = class(TCustomizeActionToolBar);
```

C++

```
class TRibbonQATCustomizeMenu : public TCustomizeActionToolBar;
```

File

RibbonActnMenus

Description

This is class RibbonActnMenus.TRibbonQATCustomizeMenu.

78.1.6.1 TRibbonQATCustomizeMenu Members

The following tables list the members exposed by TRibbonQATCustomizeMenu.

Protected Methods

	Name	Description
	DisplayShadow (see page 7329)	This is DisplayShadow, a member of class TRibbonQATCustomizeMenu.
	DoAddCustomizeItems (see page 7329)	This is DoAddCustomizeItems, a member of class TRibbonQATCustomizeMenu.
	GetControlClass (see page 7329)	This is GetControlClass, a member of class TRibbonQATCustomizeMenu.
	HideShadow (see page 7329)	This is HideShadow, a member of class TRibbonQATCustomizeMenu.
	NCPaint (see page 7330)	This is NCPaint, a member of class TRibbonQATCustomizeMenu.
	PositionPopup (see page 7330)	This is PositionPopup, a member of class TRibbonQATCustomizeMenu.
	VisibleChanging (see page 7330)	Provides the interface for a method that responds to changes in the control's visibility.

78.1.6.2 TRibbonQATCustomizeMenu Methods

The methods of the TRibbonQATCustomizeMenu class are listed here.

Protected Methods

	Name	Description
	DisplayShadow (see page 7329)	This is DisplayShadow, a member of class TRibbonQATCustomizeMenu.
	DoAddCustomizeItems (see page 7329)	This is DoAddCustomizeItems, a member of class TRibbonQATCustomizeMenu.
	GetControlClass (see page 7329)	This is GetControlClass, a member of class TRibbonQATCustomizeMenu.
	HideShadow (see page 7329)	This is HideShadow, a member of class TRibbonQATCustomizeMenu.
	NCPaint (see page 7330)	This is NCPaint, a member of class TRibbonQATCustomizeMenu.
	PositionPopup (see page 7330)	This is PositionPopup, a member of class TRibbonQATCustomizeMenu.
	VisibleChanging (see page 7330)	Provides the interface for a method that responds to changes in the control's visibility.

78.1.6.2.1 TRibbonQATCustomizeMenu.DisplayShadow Method

Pascal

```
procedure DisplayShadow; override;
```

C++

```
virtual __fastcall DisplayShadow();
```

Description

This is DisplayShadow, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.2 TRibbonQATCustomizeMenu.DoAddCustomizeItems Method

Pascal

```
procedure DoAddCustomizeItems(const ActionClient: TActionClient); override;
```

C++

```
virtual __fastcall DoAddCustomizeItems(const TActionClient ActionClient);
```

Description

This is DoAddCustomizeItems, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.3 TRibbonQATCustomizeMenu.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.4 TRibbonQATCustomizeMenu.HideShadow Method

Pascal

```
procedure HideShadow; override;
```

C++

```
virtual __fastcall HideShadow();
```

Description

This is HideShadow, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.5 TRibbonQATCustomizeMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.6 TRibbonQATCustomizeMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TRibbonQATCustomizeMenu.

78.1.6.2.7 TRibbonQATCustomizeMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible

78.1.7 TRibbonRecentDocuments Class

Class Hierarchy

Pascal

```
TRibbonRecentDocuments = class(TGraphicControl);
```

C++

```
class TRibbonRecentDocuments : public TGraphicControl;
```

File

RibbonActnMenus

Description

This is class RibbonActnMenus.TRibbonRecentDocuments.

78.1.7.1 TRibbonRecentDocuments Members

The following tables list the members exposed by TRibbonRecentDocuments.

Protected Methods

	Name	Description
	Paint (see page 7332)	Render control surface.

Public Methods

	Name	Description
	Create (see page 7331)	Creates an instance of TGraphicControl.

Published Properties

	Name	Description
	Caption (see page 7332)	Specifies a text string that identifies the control to the user.

78.1.7.2 TRibbonRecentDocuments.Create Constructor

Creates an instance of TGraphicControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TGraphicControl. However, you should never attempt to instantiate a TGraphicControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component, typically a form, that is responsible for freeing the graphic control. It becomes the value of the Owner property. If you do not explicitly provide an Owner to the constructor (that is, if AOwner is nil (Delphi) or NULL (C++)), your application is responsible for explicitly freeing the graphic control.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner

Create

Destroy

78.1.7.3 TRibbonRecentDocuments Methods

The methods of the TRibbonRecentDocuments class are listed here.

Protected Methods

	Name	Description
	Paint (see page 7332)	Render control surface.

78.1.7.3.1 TRibbonRecentDocuments.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas

78.1.7.4 TRibbonRecentDocuments Properties

The properties of the TRibbonRecentDocuments class are listed here.

Published Properties

	Name	Description
	Caption (see page 7332)	Specifies a text string that identifies the control to the user.

78.1.7.4.1 TRibbonRecentDocuments.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined

character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());
```

```
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

78.1.8 TRibbonStyleMainPopupMenu Class

Class Hierarchy

Pascal

```
TRibbonStyleMainPopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TRibbonStyleMainPopupMenu : public TCustomActionPopupMenu;
```

File

RibbonActnMenus

Description

This is class RibbonActnMenus.TRibbonStyleMainPopupMenu.

78.1.8.1 TRibbonStyleMainPopupMenu Members

The following tables list the members exposed by TRibbonStyleMainPopupMenu.

Protected Methods

	Name	Description
	NCPaint (see page 7336)	This is NCPaint, a member of class TRibbonStyleMainPopupMenu.
	PositionPopup (see page 7336)	This is PositionPopup, a member of class TRibbonStyleMainPopupMenu.

Public Methods

	Name	Description
	Create (see page 7335)	Creates and initializes a TCustomActionPopupMenu instance.

78.1.8.2 TRibbonStyleMainPopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

78.1.8.3 TRibbonStyleMainPopupMenu Methods

The methods of the TRibbonStyleMainPopupMenu class are listed here.

Protected Methods

	Name	Description
	NCPaint (see page 7336)	This is NCPaint, a member of class TRibbonStyleMainPopupMenu.
	PositionPopup (see page 7336)	This is PositionPopup, a member of class TRibbonStyleMainPopupMenu.

78.1.8.3.1 TRibbonStyleMainPopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TRibbonStyleMainPopupMenu.

78.1.8.3.2 TRibbonStyleMainPopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TRibbonStyleMainPopupMenu.

78.1.9 TRibbonStylePopupMenu Class

Class Hierarchy

Pascal

```
TRibbonStylePopupMenu = class(TRibbonActionPopupMenu);
```

C++

```
class TRibbonStylePopupMenu : public TRibbonActionPopupMenu;
```

File

RibbonActnMenus

Description

This is class RibbonActnMenus.TRibbonStylePopupMenu.

78.1.9.1 TRibbonStylePopupMenu Members

The following tables list the members exposed by TRibbonStylePopupMenu.

Protected Methods

	Name	Description
	AlignControls (see page 7338)	Aligns any controls for which the control is the parent within a specified area of the control.
	CanAutoSize (see page 7339)	Indicates how the control can be resized.
	NCPaint (see page 7301)	This is NCPaint, a member of class TRibbonActionPopupMenu.
	WMExitsizemove (see page 7339)	This is WMExitsizemove, a member of class TRibbonStylePopupMenu.
	WMNCHitTest (see page 7340)	This is WMNCHitTest, a member of class TRibbonStylePopupMenu.

Public Methods

	Name	Description
	Create (see page 7337)	Creates and initializes a TCustomActionPopupMenu instance.
	CreateControl (see page 7339)	This is CreateControl, a member of class TRibbonStylePopupMenu.
	Destroy (see page 7338)	Destroys an instance of TCustomActionMenuBar.
	TrackMenu (see page 7302)	Executes the action associated with the menu item.

Public Properties

	Name	Description
	GalleryResize (see page 7340)	This is GalleryResize, a member of class TRibbonStylePopupMenu.
	SelectFirstItem (see page 7302)	This is SelectFirstItem, a member of class TRibbonActionPopupMenu.

78.1.9.2 TRibbonStylePopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

78.1.9.3 TRibbonStylePopupMenu.Destroy Destructor

Destroys an instance of TCustomActionMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonStylePopupMenu();
```

Description

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

See Also

System::TObject::Free

Destroy

78.1.9.4 TRibbonStylePopupMenu Methods

The methods of the TRibbonStylePopupMenu class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 7338)	Aligns any controls for which the control is the parent within a specified area of the control.
	CanAutoSize (see page 7339)	Indicates how the control can be resized.
	WMExitsizemove (see page 7339)	This is WMExitsizemove, a member of class TRibbonStylePopupMenu.
	WMNCHitTest (see page 7340)	This is WMNCHitTest, a member of class TRibbonStylePopupMenu.

Public Methods

	Name	Description
	CreateControl (see page 7339)	This is CreateControl, a member of class TRibbonStylePopupMenu.

78.1.9.4.1 TRibbonStylePopupMenu.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

78.1.9.4.2 TRibbonStylePopupMenu.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

78.1.9.4.3 TRibbonStylePopupMenu.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TRibbonStylePopupMenu.

78.1.9.4.4 TRibbonStylePopupMenu.WMExitsizemove Method

Pascal

```
procedure WMExitsizemove(var Message: TMessage); message WM_EXITSIZEMOVE;
```

C++

```
__fastcall WMExitsizemove(TMessage Message);
```

Description

This is WMExitsizemove, a member of class TRibbonStylePopupMenu.

78.1.9.4.5 TRibbonStylePopupMenu.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TRibbonStylePopupMenu.

78.1.9.5 TRibbonStylePopupMenu Properties

The properties of the TRibbonStylePopupMenu class are listed here.

Public Properties

	Name	Description
	GalleryResize (see page 7340)	This is GalleryResize, a member of class TRibbonStylePopupMenu.

78.1.9.5.1 TRibbonStylePopupMenu.GalleryResize Property

Pascal

```
property GalleryResize: TGalleryResize;
```

C++

```
__property TGalleryResize GalleryResize;
```

Description

This is GalleryResize, a member of class TRibbonStylePopupMenu.

79 RibbonConsts Namespace

This is namespace RibbonConsts.

Topics

Name	Description
Constants (see page 7342)	The following table lists constants in this documentation.

Constants

Name	Description
SAddToQuickAccessToolbar (see page 7342)	This is constant RibbonConsts.SAddToQuickAccessToolbar.
SCommandIsDisabled (see page 7343)	This is constant RibbonConsts.SCommandIsDisabled.
SCustomizeQuickAccessToolbar (see page 7343)	This is constant RibbonConsts.SCustomizeQuickAccessToolbar.
SDefaultFooterText (see page 7343)	This is constant RibbonConsts.SDefaultFooterText.
SDuplicateRibbonComponentsNotAllowed (see page 7344)	This is constant RibbonConsts.SDuplicateRibbonComponentsNotAllowed.
SGenerateScreenTipsError (see page 7344)	This is constant RibbonConsts.SGenerateScreenTipsError.
SInvalidRibbonGroupParent (see page 7344)	This is constant RibbonConsts.SInvalidRibbonGroupParent.
SInvalidRibbonPageChild (see page 7345)	This is constant RibbonConsts.SInvalidRibbonPageChild.
SInvalidRibbonPageParent (see page 7345)	This is constant RibbonConsts.SInvalidRibbonPageParent.
SInvalidScreenTipItemAssign (see page 7345)	This is constant RibbonConsts.SInvalidScreenTipItemAssign.
SMinimizeTheRibbon (see page 7346)	This is constant RibbonConsts.SMinimizeTheRibbon.
SMoreCommands (see page 7346)	This is constant RibbonConsts.SMoreCommands.
SMoreControls (see page 7346)	This is constant RibbonConsts.SMoreControls.
SPageCaptionExists (see page 7347)	This is constant RibbonConsts.SPageCaptionExists.
SQATShowAboveTheRibbon (see page 7347)	This is constant RibbonConsts.SQATShowAboveTheRibbon.
SQATShowBelowTheRibbon (see page 7347)	This is constant RibbonConsts.SQATShowBelowTheRibbon.
SQuickAccessToolbarGlyphRequired (see page 7348)	This is constant RibbonConsts.SQuickAccessToolbarGlyphRequired.
SRecentDocuments (see page 7348)	This is constant RibbonConsts.SRecentDocuments.
SRemoveFromQuickAccessToolbar (see page 7348)	This is constant RibbonConsts.SRemoveFromQuickAccessToolbar.
SRibbonControlInvalidParent (see page 7349)	This is constant RibbonConsts.SRibbonControlInvalidParent.
SRibbonTabIndexInvalid (see page 7349)	This is constant RibbonConsts.SRibbonTabIndexInvalid.
SShowQuickAccessToolbarAbove (see page 7349)	This is constant RibbonConsts.SShowQuickAccessToolbarAbove.
SShowQuickAccessToolbarBelow (see page 7350)	This is constant RibbonConsts.SShowQuickAccessToolbarBelow.
SUseTheGlyphProperty (see page 7350)	This is constant RibbonConsts.SUseTheGlyphProperty.

79.1 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SAddToQuickAccessToolbar (see page 7342)	This is constant RibbonConsts.SAddToQuickAccessToolbar.
SCommandIsDisabled (see page 7343)	This is constant RibbonConsts.SCommandIsDisabled.
SCustomizeQuickAccessToolbar (see page 7343)	This is constant RibbonConsts.SCustomizeQuickAccessToolbar.
SDefaultFooterText (see page 7343)	This is constant RibbonConsts.SDefaultFooterText.
SDuplicateRibbonComponentsNotAllowed (see page 7344)	This is constant RibbonConsts.SDuplicateRibbonComponentsNotAllowed.
SGenerateScreenTipsError (see page 7344)	This is constant RibbonConsts.SGenerateScreenTipsError.
SInvalidRibbonGroupParent (see page 7344)	This is constant RibbonConsts.SInvalidRibbonGroupParent.
SInvalidRibbonPageChild (see page 7345)	This is constant RibbonConsts.SInvalidRibbonPageChild.
SInvalidRibbonPageParent (see page 7345)	This is constant RibbonConsts.SInvalidRibbonPageParent.
SInvalidScreenTipItemAssign (see page 7345)	This is constant RibbonConsts.SInvalidScreenTipItemAssign.
SMinimizeTheRibbon (see page 7346)	This is constant RibbonConsts.SMinimizeTheRibbon.
SMoreCommands (see page 7346)	This is constant RibbonConsts.SMoreCommands.
SMoreControls (see page 7346)	This is constant RibbonConsts.SMoreControls.
SPageCaptionExists (see page 7347)	This is constant RibbonConsts.SPageCaptionExists.
SQATShowAboveTheRibbon (see page 7347)	This is constant RibbonConsts.SQATShowAboveTheRibbon.
SQATShowBelowTheRibbon (see page 7347)	This is constant RibbonConsts.SQATShowBelowTheRibbon.
SQuickAccessToolbarGlyphRequired (see page 7348)	This is constant RibbonConsts.SQuickAccessToolbarGlyphRequired.
SRecentDocuments (see page 7348)	This is constant RibbonConsts.SRecentDocuments.
SRemoveFromQuickAccessToolbar (see page 7348)	This is constant RibbonConsts.SRemoveFromQuickAccessToolbar.
SRibbonControlInvalidParent (see page 7349)	This is constant RibbonConsts.SRibbonControlInvalidParent.
SRibbonTabIndexInvalid (see page 7349)	This is constant RibbonConsts.SRibbonTabIndexInvalid.
SShowQuickAccessToolbarAbove (see page 7349)	This is constant RibbonConsts.SShowQuickAccessToolbarAbove.
SShowQuickAccessToolbarBelow (see page 7350)	This is constant RibbonConsts.SShowQuickAccessToolbarBelow.
SUseTheGlyphProperty (see page 7350)	This is constant RibbonConsts.SUseTheGlyphProperty.

79.1.1 RibbonConsts.SAddToQuickAccessToolbar Constant

Pascal

```
SAddToQuickAccessToolbar = 'Add to Quick Access Toolbar';
```

C++

```
SAddToQuickAccessToolbar = 'Add to Quick Access Toolbar';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SAddToQuickAccessToolbar.

79.1.2 RibbonConsts.SCommandIsDisabled Constant

Pascal

```
SCommandIsDisabled = 'This command is disabled.';
```

C++

```
SCommandIsDisabled = 'This command is disabled.';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SCommandIsDisabled.

79.1.3 RibbonConsts.SCustomizeQuickAccessToolbar Constant

Pascal

```
SCustomizeQuickAccessToolbar = 'Customize Quick Access Toolbar...';
```

C++

```
SCustomizeQuickAccessToolbar = 'Customize Quick Access Toolbar...';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SCustomizeQuickAccessToolbar.

79.1.4 RibbonConsts.SDefaultFooterText Constant

Pascal

```
SDefaultFooterText = 'Press F1 for more help.';
```

C++

```
SDefaultFooterText = 'Press F1 for more help.';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SDefaultFooterText.

79.1.5 RibbonConsts.SDuplicateRibbonComponentsNotAllowed Constant

Pascal

```
SDuplicateRibbonComponentsNotAllowed = 'Only one Ribbon control on a Form is allowed';
```

C++

```
SDuplicateRibbonComponentsNotAllowed = 'Only one Ribbon control on a Form is allowed';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SDuplicateRibbonComponentsNotAllowed.

79.1.6 RibbonConsts.SGenerateScreenTipsError Constant

Pascal

```
SGenerateScreenTipsError = 'Unable to generate Screen Tips because there are no linked Action
Lists.';
```

C++

```
SGenerateScreenTipsError = 'Unable to generate Screen Tips because there are no linked Action
Lists.';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SGenerateScreenTipsError.

79.1.7 RibbonConsts.SInvalidRibbonGroupParent Constant

Pascal

```
SInvalidRibbonGroupParent = 'Ribbon Group must have a Ribbon Page as its parent';
```

C++

```
SInvalidRibbonGroupParent = 'Ribbon Group must have a Ribbon Page as its parent';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SInvalidRibbonGroupParent.

79.1.8 RibbonConsts.SInvalidRibbonPageChild Constant

Pascal

```
SInvalidRibbonPageChild = 'Only Ribbon Groups can be inserted into a Ribbon Page';
```

C++

```
SInvalidRibbonPageChild = 'Only Ribbon Groups can be inserted into a Ribbon Page';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SInvalidRibbonPageChild.

79.1.9 RibbonConsts.SInvalidRibbonPageParent Constant

Pascal

```
SInvalidRibbonPageParent = 'A Ribbon Pages parent must be TCustomRibbon';
```

C++

```
SInvalidRibbonPageParent = 'A Ribbon Pages parent must be TCustomRibbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SInvalidRibbonPageParent.

79.1.10 RibbonConsts.SInvalidScreenTipItemAssign Constant

Pascal

```
SInvalidScreenTipItemAssign = 'Cannot assign a non-ScreenTipItem to a ScreenTipItem';
```

C++

```
SInvalidScreenTipItemAssign = 'Cannot assign a non-ScreenTipItem to a ScreenTipItem';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SInvalidScreenTipItemAssign.

79.1.11 RibbonConsts.SMinimizeTheRibbon Constant

Pascal

```
SMinimizeTheRibbon = 'Minimize the Ribbon';
```

C++

```
SMinimizeTheRibbon = 'Minimize the Ribbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SMinimizeTheRibbon.

79.1.12 RibbonConsts.SMoreCommands Constant

Pascal

```
SMoreCommands = 'More Commands...';
```

C++

```
SMoreCommands = 'More Commands...';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SMoreCommands.

79.1.13 RibbonConsts.SMoreControls Constant

Pascal

```
SMoreControls = 'More controls';
```

C++

```
SMoreControls = 'More controls';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SMoreControls.

79.1.14 RibbonConsts.SPageCaptionExists Constant

Pascal

```
SPageCaptionExists = 'A page with the caption "%s" already exists';
```

C++

```
SPageCaptionExists = 'A page with the caption "%s" already exists';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SPageCaptionExists.

79.1.15 RibbonConsts.SQATShowAboveTheRibbon Constant

Pascal

```
SQATShowAboveTheRibbon = 'Show Above the Ribbon';
```

C++

```
SQATShowAboveTheRibbon = 'Show Above the Ribbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SQATShowAboveTheRibbon.

79.1.16 RibbonConsts.SQATShowBelowTheRibbon Constant

Pascal

```
SQATShowBelowTheRibbon = 'Show Below the Ribbon';
```

C++

```
SQATShowBelowTheRibbon = 'Show Below the Ribbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SQATShowBelowTheRibbon.

79.1.17 RibbonConsts.SQuickAccessToolbarGlyphRequired Constant

Pascal

```
SQuickAccessToolbarGlyphRequired = 'Cannot place an item on the Quick Access Toolbar that does not have a glyph';
```

C++

```
SQuickAccessToolbarGlyphRequired = 'Cannot place an item on the Quick Access Toolbar that does not have a glyph';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SQuickAccessToolbarGlyphRequired.

79.1.18 RibbonConsts.SRecentDocuments Constant

Pascal

```
SRecentDocuments = 'Recent Documents';
```

C++

```
SRecentDocuments = 'Recent Documents';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SRecentDocuments.

79.1.19 RibbonConsts.SRemoveFromQuickAccessToolbar Constant

Pascal

```
SRemoveFromQuickAccessToolbar = 'Remove from Quick Access Toolbar';
```

C++

```
SRemoveFromQuickAccessToolbar = 'Remove from Quick Access Toolbar';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SRemoveFromQuickAccessToolbar.

79.1.20 RibbonConsts.SRibbonControlInvalidParent Constant

Pascal

```
SRibbonControlInvalidParent = 'A %s can only be placed on a Ribbon Group with an ActionManager assigned';
```

C++

```
SRibbonControlInvalidParent = 'A %s can only be placed on a Ribbon Group with an ActionManager assigned';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SRibbonControlInvalidParent.

79.1.21 RibbonConsts.SRibbonTabIndexInvalid Constant

Pascal

```
SRibbonTabIndexInvalid = 'Specified TabIndex is invalid';
```

C++

```
SRibbonTabIndexInvalid = 'Specified TabIndex is invalid';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SRibbonTabIndexInvalid.

79.1.22 RibbonConsts.SShowQuickAccessToolbarAbove Constant

Pascal

```
SShowQuickAccessToolbarAbove = 'Show Quick Access Toolbar Above the Ribbon';
```

C++

```
SShowQuickAccessToolbarAbove = 'Show Quick Access Toolbar Above the Ribbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SShowQuickAccessToolbarAbove.

79.1.23 RibbonConsts.SShowQuickAccessToolbarBelow Constant

Pascal

```
SShowQuickAccessToolbarBelow = 'Show Quick Access Toolbar Below the Ribbon';
```

C++

```
SShowQuickAccessToolbarBelow = 'Show Quick Access Toolbar Below the Ribbon';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SShowQuickAccessToolbarBelow.

79.1.24 RibbonConsts.SUseTheGlyphProperty Constant

Pascal

```
SUseTheGlyphProperty = 'Use the glyph property to set an image';
```

C++

```
SUseTheGlyphProperty = 'Use the glyph property to set an image';
```

File

RibbonConsts

Description

This is constant RibbonConsts.SUseTheGlyphProperty.

80 RibbonGalleryBar Namespace

This is namespace RibbonGalleryBar.

Topics

Name	Description
Classes (see page 7352)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomRibbonGalleryBar (see page 7352)	This is class RibbonGalleryBar.TCustomRibbonGalleryBar.
	TCustomRibbonGalleryPopupBar (see page 7358)	This is class RibbonGalleryBar.TCustomRibbonGalleryPopupBar.
	TRibbonGalleryBar (see page 7360)	This is class RibbonGalleryBar.TRibbonGalleryBar.
	TRibbonGalleryPopupBar (see page 7362)	This is class RibbonGalleryBar.TRibbonGalleryPopupBar.
	TRibbonPopupGalleryBar (see page 7362)	This is class RibbonGalleryBar.TRibbonPopupGalleryBar.

80.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomRibbonGalleryBar (see page 7352)	This is class RibbonGalleryBar.TCustomRibbonGalleryBar.
	TCustomRibbonGalleryPopupBar (see page 7358)	This is class RibbonGalleryBar.TCustomRibbonGalleryPopupBar.
	TRibbonGalleryBar (see page 7360)	This is class RibbonGalleryBar.TRibbonGalleryBar.
	TRibbonGalleryPopupBar (see page 7362)	This is class RibbonGalleryBar.TRibbonGalleryPopupBar.
	TRibbonPopupGalleryBar (see page 7362)	This is class RibbonGalleryBar.TRibbonPopupGalleryBar.

80.1.1 TCustomRibbonGalleryBar Class

Class Hierarchy

Pascal

```
TCustomRibbonGalleryBar = class(TActionToolBar);
```

C++

```
class TCustomRibbonGalleryBar : public TActionToolBar;
```

File

RibbonGalleryBar

Description

This is class RibbonGalleryBar.TCustomRibbonGalleryBar.

80.1.1.1 TCustomRibbonGalleryBar Members

The following tables list the members exposed by TCustomRibbonGalleryBar.

Protected Methods

	Name	Description
	AlignControls (see page 7353)	Aligns any controls for which the control is the parent within a specified area of the control.

Public Methods

	Name	Description
	Create (see page 7353)	Creates and initializes a TCustomActionToolBar instance.
	CreateControl (see page 7354)	This is CreateControl, a member of class TCustomRibbonGalleryBar.
	DoDropCategory (see page 7354)	This is DoDropCategory, a member of class TCustomRibbonGalleryBar.

Public Properties

	Name	Description
	ItemsPerRow (see page 7358)	This is ItemsPerRow, a member of class TCustomRibbonGalleryBar.

	ShowItemCaptions (see page 7358)	This is ShowItemCaptions, a member of class TCustomRibbonGalleryBar.

Published Properties

	Name	Description
	Align (see page 7354)	Determines how the control aligns within its container (parent control).

80.1.1.2 TCustomRibbonGalleryBar.Create Constructor

Creates and initializes a TCustomActionToolBar instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionToolBar and initializes its properties.

```
__fastcall virtual TCustomActionToolBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall (HWND ParentWindow) : Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

80.1.1.3 TCustomRibbonGalleryBar Methods

The methods of the TCustomRibbonGalleryBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 7353)	Aligns any controls for which the control is the parent within a specified area of the control.

Public Methods

	Name	Description
	CreateControl (see page 7354)	This is CreateControl, a member of class TCustomRibbonGalleryBar.
	DoDropCategory (see page 7354)	This is DoDropCategory, a member of class TCustomRibbonGalleryBar.

80.1.1.3.1 TCustomRibbonGalleryBar.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

80.1.1.3.2 TCustomRibbonGalleryBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomRibbonGalleryBar.

80.1.1.3.3 TCustomRibbonGalleryBar.DoDropCategory Method

Pascal

```
procedure DoDropCategory(Source: TCategoryDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropCategory(TCategoryDragObject Source, const int X, const int Y);
```

Description

This is DoDropCategory, a member of class TCustomRibbonGalleryBar.

80.1.1.4 TCustomRibbonGalleryBar Properties

The properties of the TCustomRibbonGalleryBar class are listed here.

Public Properties

	Name	Description
	ItemsPerRow (see page 7358)	This is ItemsPerRow, a member of class TCustomRibbonGalleryBar.
	ShowItemCaptions (see page 7358)	This is ShowItemCaptions, a member of class TCustomRibbonGalleryBar.

Published Properties

	Name	Description
	Align (see page 7354)	Determines how the control aligns within its container (parent control).

80.1.1.4.1 TCustomRibbonGalleryBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form,

panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/
```

```
#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
```

```
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
```

```
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

80.1.1.4.2 TCustomRibbonGalleryBar.ItemsPerRow Property

Pascal

```
property ItemsPerRow: Integer;
```

C++

```
__property int ItemsPerRow;
```

Description

This is ItemsPerRow, a member of class TCustomRibbonGalleryBar.

80.1.1.4.3 TCustomRibbonGalleryBar.ShowItemCaptions Property

Pascal

```
property ShowItemCaptions: Boolean;
```

C++

```
__property Boolean ShowItemCaptions;
```

Description

This is ShowItemCaptions, a member of class TCustomRibbonGalleryBar.

80.1.2 TCustomRibbonGalleryPopupBar Class

Class Hierarchy

Pascal

```
TCustomRibbonGalleryPopupBar = class(TRibbonActionPopupMenu);
```

C++

```
class TCustomRibbonGalleryPopupBar : public TRibbonActionPopupMenu;
```

File

RibbonGalleryBar

Description

This is class RibbonGalleryBar.TCustomRibbonGalleryPopupBar.

80.1.2.1 TCustomRibbonGalleryPopupBar Members

The following tables list the members exposed by TCustomRibbonGalleryPopupBar.

Protected Methods

	Name	Description
	AlignControls (see page 7359)	Aligns any controls for which the control is the parent within a specified area of the control.
	CreateControls (see page 7360)	This is CreateControls, a member of class TCustomRibbonGalleryPopupBar.
	DoGetControlClass (see page 7360)	This is DoGetControlClass, a member of class TCustomRibbonGalleryPopupBar.
	NCPaint (see page 7301)	This is NCPaint, a member of class TRibbonActionPopupMenu.
	SetColorMap (see page 7360)	This is SetColorMap, a member of class TCustomRibbonGalleryPopupBar.

Public Methods

	Name	Description
	CreateControl (see page 7301)	This is CreateControl, a member of class TRibbonActionPopupMenu.
	TrackMenu (see page 7302)	Executes the action associated with the menu item.

Public Properties

	Name	Description
	SelectFirstItem (see page 7302)	This is SelectFirstItem, a member of class TRibbonActionPopupMenu.

80.1.2.2 TCustomRibbonGalleryPopupBar Methods

The methods of the TCustomRibbonGalleryPopupBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 7359)	Aligns any controls for which the control is the parent within a specified area of the control.
	CreateControls (see page 7360)	This is CreateControls, a member of class TCustomRibbonGalleryPopupBar.
	DoGetControlClass (see page 7360)	This is DoGetControlClass, a member of class TCustomRibbonGalleryPopupBar.
	SetColorMap (see page 7360)	This is SetColorMap, a member of class TCustomRibbonGalleryPopupBar.

80.1.2.2.1 TCustomRibbonGalleryPopupBar.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

80.1.2.2.2 TCustomRibbonGalleryPopupBar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomRibbonGalleryPopupBar.

80.1.2.2.3 TCustomRibbonGalleryPopupBar.DoGetControlClass Method

Pascal

```
function DoGetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass DoGetControlClass(TActionClientItem AnItem);
```

Description

This is DoGetControlClass, a member of class TCustomRibbonGalleryPopupBar.

80.1.2.2.4 TCustomRibbonGalleryPopupBar.SetColorMap Method

Pascal

```
procedure SetColorMap(const Value: TCustomActionBarColorMap); override;
```

C++

```
virtual __fastcall SetColorMap(const TCustomActionBarColorMap Value);
```

Description

This is SetColorMap, a member of class TCustomRibbonGalleryPopupBar.

80.1.3 TRibbonGalleryBar Class

Class Hierarchy

Pascal

```
TRibbonGalleryBar = class(TCustomRibbonGalleryBar);
```

C++

```
class TRibbonGalleryBar : public TCustomRibbonGalleryBar;
```

File

RibbonGalleryBar

Description

This is class RibbonGalleryBar.TRibbonGalleryBar.

80.1.3.1 TRibbonGalleryBar Members

The following tables list the members exposed by TRibbonGalleryBar.

Protected Methods

	Name	Description
	AlignControls (see page 7353)	Aligns any controls for which the control is the parent within a specified area of the control.

Public Methods

	Name	Description
	Create (see page 7353)	Creates and initializes a TCustomActionToolBar instance.
	CreateControl (see page 7354)	This is CreateControl, a member of class TCustomRibbonGalleryBar.
	DoDropCategory (see page 7354)	This is DoDropCategory, a member of class TCustomRibbonGalleryBar.

Published Properties

	Name	Description
	Align (see page 7354)	Determines how the control aligns within its container (parent control).
	Items (see page 7361)	Lists the action clients that correspond to the child controls of this action band.
	ItemsPerRow (see page 7362)	This is ItemsPerRow, a member of class TRibbonGalleryBar.
	ShowItemCaptions (see page 7362)	This is ShowItemCaptions, a member of class TRibbonGalleryBar.

80.1.3.2 TRibbonGalleryBar Properties

The properties of the TRibbonGalleryBar class are listed here.

Published Properties

	Name	Description
	Items (see page 7361)	Lists the action clients that correspond to the child controls of this action band.
	ItemsPerRow (see page 7362)	This is ItemsPerRow, a member of class TRibbonGalleryBar.
	ShowItemCaptions (see page 7362)	This is ShowItemCaptions, a member of class TRibbonGalleryBar.

80.1.3.2.1 TRibbonGalleryBar.Items Property

Lists the action clients that correspond to the child controls of this action band.

Pascal

```
property Items;
```

C++

```
__property Items;
```

Description

Items is used internally to access the components that the action manager lists to represent the child controls of this action band. Each action client object in the Items collection corresponds to one of the controls that the ActionControls property lists. The entire collection of items belongs to the action client specified by this action band's ActionClient property.

See Also

ActionControls

ActionClient

ContextBar

80.1.3.2.2 TRibbonGalleryBar.ItemsPerRow Property

Pascal

```
property ItemsPerRow: Integer;
```

C++

```
__property int ItemsPerRow;
```

Description

This is ItemsPerRow, a member of class TRibbonGalleryBar.

80.1.3.2.3 TRibbonGalleryBar.ShowItemCaptions Property

Pascal

```
property ShowItemCaptions: Boolean;
```

C++

```
__property Boolean ShowItemCaptions;
```

Description

This is ShowItemCaptions, a member of class TRibbonGalleryBar.

80.1.4 TRibbonGalleryPopupBar Class

Class Hierarchy

Pascal

```
TRibbonGalleryPopupBar = class(TCustomRibbonGalleryPopupBar);
```

C++

```
class TRibbonGalleryPopupBar : public TCustomRibbonGalleryPopupBar;
```

File

RibbonGalleryBar

Description

This is class RibbonGalleryBar.TRibbonGalleryPopupBar.

80.1.5 TRibbonPopupGalleryBar Class

Class Hierarchy

Pascal

```
TRibbonPopupGalleryBar = class(TRibbonGalleryBar);
```

C++

```
class TRibbonPopupGalleryBar : public TRibbonGalleryBar;
```

File

RibbonGalleryBar

Description

This is class RibbonGalleryBar.TRibbonPopupGalleryBar.

80.1.5.1 TRibbonPopupGalleryBar Members

The following tables list the members exposed by TRibbonPopupGalleryBar.

Protected Methods

	Name	Description
	AlignControls (see page 7353)	Aligns any controls for which the control is the parent within a specified area of the control.
	CreateParams (see page 7364)	Initializes a window-creation parameter data structure.

Public Methods

	Name	Description
	Assign (see page 7364)	Copies the contents of another, similar object.
	Create (see page 7353)	Creates and initializes a TCustomActionToolBar instance.
	CreateControl (see page 7354)	This is CreateControl, a member of class TCustomRibbonGalleryBar.
	DisplayGallery (see page 7365)	This is DisplayGallery, a member of class TRibbonPopupGalleryBar.
	DoDropCategory (see page 7354)	This is DoDropCategory, a member of class TCustomRibbonGalleryBar.

Public Properties

	Name	Description
	ItemsPerRow (see page 7358)	This is ItemsPerRow, a member of class TCustomRibbonGalleryBar.
	ShowItemCaptions (see page 7358)	This is ShowItemCaptions, a member of class TCustomRibbonGalleryBar.

Published Properties

	Name	Description
	Align (see page 7354)	Determines how the control aligns within its container (parent control).
	Items (see page 7361)	Lists the action clients that correspond to the child controls of this action band.
	ItemsPerRow (see page 7362)	This is ItemsPerRow, a member of class TRibbonGalleryBar.
	ShowItemCaptions (see page 7362)	This is ShowItemCaptions, a member of class TRibbonGalleryBar.

80.1.5.2 TRibbonPopupGalleryBar Methods

The methods of the TRibbonPopupGalleryBar class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 7364)	Initializes a window-creation parameter data structure.

Public Methods

	Name	Description
	Assign (see page 7364)	Copies the contents of another, similar object.
	DisplayGallery (see page 7365)	This is DisplayGallery, a member of class TRibbonPopupGalleryBar.

80.1.5.2.1 TRibbonPopupGalleryBar.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError

80.1.5.2.2 TRibbonPopupGalleryBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

80.1.5.2.3 TRibbonPopupGalleryBar.DisplayGallery Method

Pascal

```
procedure DisplayGallery(AX: Integer; AY: Integer);
```

C++

```
__fastcall DisplayGallery(int AX, int AY);
```

Description

This is DisplayGallery, a member of class TRibbonPopupGalleryBar.

81 RibbonStyleActnCtrls Namespace

This is namespace RibbonStyleActnCtrls.

Topics

Name	Description
Classes (see page 7369)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 7383)	The following table lists structs, records, enums in this documentation.
Types (see page 7406)	The following table lists types in this documentation.

Classes

	Name	Description
	TAlphaBitmap (see page 7369)	Set AlphaFormat property to **afDefined**.
	TCustomRibbonSkin (see page 7372)	Load resources from file or application.
	TRibbonStyleActionBars (see page 7376)	Action bar style.

Structs, Records, Enums

	Name	Description
	TRibbonElement (see page 7383)	Elements of Ribbon (see page 6713) that need to be skinned.
	TSkinApplicationMenu (see page 7384)	Individual skinned items of Application Menu.
	TSkinButton (see page 7386)	Individual skinned items of Button.
	TSkinCheckBox (see page 7390)	This is record RibbonStyleActnCtrls.TSkinCheckBox.
	TSkinComboBox (see page 7391)	This is record RibbonStyleActnCtrls.TSkinComboBox.
	TSkinForm (see page 7391)	Individual skinned items of Form.
	TSkinGallery (see page 7394)	Individual skinned items of Gallery.
	TSkinMenu (see page 7395)	Individual skinned items of Menu.
	TSkinQuickAccessToolbar (see page 7397)	Individual skinned items of Quick Access Toolbar.
	TSkinRadioButton (see page 7398)	This is record RibbonStyleActnCtrls.TSkinRadioButton.
	TSkinRibbon (see page 7399)	Individual skinned items of Ribbon (see page 6713).
	TSkinRibbonGroup (see page 7400)	Individual skinned items of Group.
	TSkinScrollBar (see page 7402)	Individual skinned items of ScrollBar.
	TSkinSpinEdit (see page 7403)	This is record RibbonStyleActnCtrls.TSkinSpinEdit.
	TSkinStatusBar (see page 7403)	Individual skinned items of Status Bar.
	TSkinTab (see page 7404)	Individual skinned items of Tab.

Types

Name	Description
TCustomRibbonSkinClass (see page 7406)	This is type RibbonStyleActnCtrls.TCustomRibbonSkinClass.

81.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAlphaBitmap (see page 7369)	Set AlphaFormat property to **afDefined**.
	TCustomRibbonSkin (see page 7372)	Load resources from file or application.
	TRibbonStyleActionBars (see page 7376)	Action bar style.

81.1.1 TAlphaBitmap Class

Set AlphaFormat property to **afDefined**.

Class Hierarchy

Pascal

```
TAlphaBitmap = class(TBitmap);
```

C++

```
class TAlphaBitmap : public TBitmap;
```

File

RibbonStyleActnCtrls

Description

TAlphaBitmap is a TBitmap (see page 2628) descendant that automatically sets the AlphaFormat property to **afDefined**.

See Also

TBitmap (see page 2628)

AlphaFormat (see page 2651)

TAlphaFormat (see page 2860)

81.1.1.1 TAlphaBitmap Members

The following tables list the members exposed by TAlphaBitmap.

Protected Methods

	Name	Description
	Changed (see page 2635)	This is Changed, a member of class TBitmap.
	Changed (see page 2751)	This is Changed, a member of class TGraphic.
	DefineProperties (see page 2751)	Provides an interface for a method that reads and writes otherwise unpublished data.

	Draw (see page 2636)	This is Draw, a member of class TBitmap.
	Draw (see page 2752)	This is Draw, a member of class TGraphic.
	DrawTransparent (see page 2637)	This is DrawTransparent, a member of class TBitmap.
	DrawTransparent (see page 2752)	This is DrawTransparent, a member of class TGraphic.
	Equals (see page 2752)	This is the overview for the Equals method overload.
	GetEmpty (see page 2638)	This is GetEmpty, a member of class TBitmap.
	GetEmpty (see page 2753)	This is GetEmpty, a member of class TGraphic.
	GetHeight (see page 2638)	This is GetHeight, a member of class TBitmap.
	GetHeight (see page 2753)	This is GetHeight, a member of class TGraphic.
	GetPalette (see page 2639)	This is GetPalette, a member of class TBitmap.
	GetPalette (see page 2754)	This is GetPalette, a member of class TGraphic.
	GetSupportsPartialTransparency (see page 2639)	This is GetSupportsPartialTransparency, a member of class TBitmap.
	GetSupportsPartialTransparency (see page 2754)	This is GetSupportsPartialTransparency, a member of class TGraphic.
	GetTransparent (see page 2754)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 2639)	This is GetWidth, a member of class TBitmap.
	GetWidth (see page 2754)	This is GetWidth, a member of class TGraphic.
	HandleNeeded (see page 2640)	This is HandleNeeded, a member of class TBitmap.
	MaskHandleNeeded (see page 2644)	This is MaskHandleNeeded, a member of class TBitmap.
	PaletteNeeded (see page 2645)	This is PaletteNeeded, a member of class TBitmap.
	Progress (see page 2756)	This is Progress, a member of class TGraphic.
	ReadData (see page 2645)	This is ReadData, a member of class TBitmap.
	ReadData (see page 2757)	This is ReadData, a member of class TGraphic.
	SetHeight (see page 2649)	This is SetHeight, a member of class TBitmap.
	SetHeight (see page 2758)	This is SetHeight, a member of class TGraphic.
	SetPalette (see page 2649)	This is SetPalette, a member of class TBitmap.
	SetPalette (see page 2759)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 2759)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 2650)	This is SetWidth, a member of class TBitmap.
	SetWidth (see page 2759)	This is SetWidth, a member of class TGraphic.
	WriteData (see page 2650)	This is WriteData, a member of class TBitmap.
	WriteData (see page 2760)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 2763)	Occurs whenever a graphical image changes.
	OnProgress (see page 2764)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	Assign (see page 2633)	Copies a new bitmap image to the bitmap object.
	Create (see page 7371)	Create TAlphaBitmap (see page 7369) instance.
	Destroy (see page 2632)	Disposes of the instance of the bitmap object.
	Dormant (see page 2635)	Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.
	FreeImage (see page 2637)	Frees the cached file image stored in memory by the bitmap.
	HandleAllocated (see page 2639)	Indicates whether the TBitmap (see page 2628) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 2640)	Loads a bitmap from the Clipboard into the bitmap object.
	LoadFromClipboardFormat (see page 2754)	Introduces an abstract method that loads the image from a variable in Clipboard format.
	LoadFromFile (see page 2755)	Loads a graphic image stored in a file.
	LoadFromResourceID (see page 2642)	Loads a bitmap from a resource into the bitmap object.
	LoadFromResourceName (see page 2642)	Loads a bitmap resource into the bitmap object.
	LoadFromStream (see page 2643)	Loads the bitmap from a stream into the bitmap object.
	LoadFromStream (see page 2756)	Introduces an abstract method that loads the image from a stream.

	Mask (see page 2644)	Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 2660) with white and every other color with black.
	ReleaseHandle (see page 2645)	Returns the handle to the bitmap so that the TBitmap (see page 2628) object no longer knows about the handle.
	ReleaseMaskHandle (see page 2646)	Returns the handle to the bitmap's mask so that the TBitmap (see page 2628) object no longer knows about the mask handle.
	ReleasePalette (see page 2647)	Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 2628) object.
V	SaveToClipboardFormat (see page 2647)	Creates a local copy of the bitmap in Clipboard format.
A	SaveToClipboardFormat (see page 2757)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
V	SaveToFile (see page 2757)	Saves a graphics image to a file.
V	SaveToStream (see page 2649)	Saves the bitmap to a stream.
A	SaveToStream (see page 2758)	Introduces an abstract method that saves the graphic image to a stream.
V	SetSize (see page 2650)	Specifies the size of the bitmap.
V	SetSize (see page 2759)	Specifies the size of the graphic.

Public Properties

	Name	Description
	AlphaFormat (see page 2651)	This is AlphaFormat, a member of class TBitmap.
R	Canvas (see page 2651)	Provides access to a drawing surface that represents the bitmap.
R	Empty (see page 2760)	Indicates whether the graphics object contains a graphic.
	Handle (see page 2652)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	HandleType (see page 2653)	Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.
	Height (see page 2760)	Specifies the vertical size of the graphic in pixels.
	IgnorePalette (see page 2654)	Determines whether the bitmap realizes its palette when drawing its image.
	MaskHandle (see page 2654)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	Modified (see page 2761)	Indicates whether the graphics object has been changed or edited.
	Monochrome (see page 2654)	Determines whether the bitmap displays its image in monochrome.
	Palette (see page 2761)	Indicates the color palette of the graphical image.
	PaletteModified (see page 2761)	Indicates whether the palette has changed.
	PixelFormat (see page 2656)	Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.
R	ScanLine (see page 2658)	Provides indexed access to each line of pixels.
R	SupportsPartialTransparency (see page 2762)	This is SupportsPartialTransparency, a member of class TGraphic.
	Transparent (see page 2762)	Indicates whether the image covers its rectangular area.
	TransparentColor (see page 2660)	Determines which color of the bitmap is to be transparent when the bitmap is drawn.
	TransparentMode (see page 2661)	Determines whether the TransparentColor (see page 2660) property's value is automatically calculated or stored with the bitmap object.
	Width (see page 2763)	Determines the maximum width of the graphics object in pixels.

81.1.1.2 TAlphaBitmap.Create Constructor

Create TAlphaBitmap (see page 7369) instance.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TAlphaBitmap();
```

Description

This method creates an TAlphaBitmap (see page 7369) instance.

81.1.2 TCustomRibbonSkin Class

Load resources from file or application.

Class Hierarchy

Pascal

```
TCustomRibbonSkin = class;
```

C++

```
class TCustomRibbonSkin;
```

File

RibbonStyleActnCtrls

Description

TCustomRibbonSkin is responsible for loading the resources from a file or application. It also loads the common elements in each skin. Each descendant then modifies the resource or file to be loaded.

See Also

TRibbonSkinLoadType

81.1.2.1 TCustomRibbonSkin Members

The following tables list the members exposed by TCustomRibbonSkin.

Enumerations

	Name	Description
	TRibbonSkinLoadType (see page 7373)	This is record RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType.

Protected Methods

	Name	Description
	DoLoadByFile (see page 7374)	This is DoLoadByFile, a member of class TCustomRibbonSkin.
	DoLoadByResource (see page 7374)	This is DoLoadByResource, a member of class TCustomRibbonSkin.

Public Methods

	Name	Description
	Create (see page 7373)	Create TCustomRibbonSkin (see page 7372) instance.
	Destroy (see page 7373)	Destroy TCustomRibbonSkin (see page 7372) instance.
	GetLoadType (see page 7375)	Set method for loading skins.
	GetLoadTypeValue (see page 7375)	This is GetLoadTypeValue, a member of class TCustomRibbonSkin.
	GetResourceHandle (see page 7375)	Override `GetResourceHandle` to supply skins in separate DLL.
	LoadSkin (see page 7375)	This is LoadSkin, a member of class TCustomRibbonSkin.

Public Properties

	Name	Description
	Bitmap (see page 7376)	This is Bitmap, a member of class TCustomRibbonSkin.

81.1.2.2 TCustomRibbonSkin.Create Constructor

Create TCustomRibbonSkin (see page 7372) instance.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TCustomRibbonSkin();
```

Description

This method creates a TCustomRibbonSkin (see page 7372) instance.

See Also

Destroy (see page 7373)

81.1.2.3 TCustomRibbonSkin.Destroy Destructor

Destroy TCustomRibbonSkin (see page 7372) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRibbonSkin();
```

Description

This method destroys a TCustomRibbonSkin (see page 7372) instance.

See Also

Create (see page 7373)

81.1.2.4 TCustomRibbonSkin Enumerations

The enumerations of the TCustomRibbonSkin class are listed here.

Enumerations

	Name	Description
	TRibbonSkinLoadType (see page 7373)	This is record RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType.

81.1.2.4.1 RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType Enumeration

Pascal

```
TRibbonSkinLoadType = (
  ltResource,
  ltFile
);
```

C++

```
enum TRibbonSkinLoadType {
  ltResource,
  ltFile
};
```

File

RibbonStyleActnCtrls

Description

This is record RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType.

81.1.2.5 TCustomRibbonSkin Methods

The methods of the TCustomRibbonSkin class are listed here.

Protected Methods

	Name	Description
	DoLoadByFile (see page 7374)	This is DoLoadByFile, a member of class TCustomRibbonSkin.
	DoLoadByResource (see page 7374)	This is DoLoadByResource, a member of class TCustomRibbonSkin.

Public Methods

	Name	Description
	GetLoadType (see page 7375)	Set method for loading skins.
	GetLoadTypeValue (see page 7375)	This is GetLoadTypeValue, a member of class TCustomRibbonSkin.
	GetResourceHandle (see page 7375)	Override `GetResourceHandle` to supply skins in separate DLL.
	LoadSkin (see page 7375)	This is LoadSkin, a member of class TCustomRibbonSkin.

81.1.2.5.1 TCustomRibbonSkin.DoLoadByFile Method

Pascal

```
procedure DoLoadByFile; virtual;
```

C++

```
virtual __fastcall DoLoadByFile();
```

Description

This is DoLoadByFile, a member of class TCustomRibbonSkin.

81.1.2.5.2 TCustomRibbonSkin.DoLoadByResource Method

Pascal

```
procedure DoLoadByResource; virtual;
```

C++

```
virtual __fastcall DoLoadByResource();
```

Description

This is DoLoadByResource, a member of class TCustomRibbonSkin.

81.1.2.5.3 TCustomRibbonSkin.GetLoadType Method

Set method for loading skins.

Pascal

```
function GetLoadType: TRibbonSkinLoadType; virtual;
```

C++

```
virtual __fastcall TRibbonSkinLoadType GetLoadType();
```

Description

GetLoadType sets the method for loading the skins for use by the application.

81.1.2.5.4 TCustomRibbonSkin.GetLoadTypeValue Method

Pascal

```
function GetLoadTypeValue: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetLoadTypeValue();
```

Description

This is GetLoadTypeValue, a member of class TCustomRibbonSkin.

81.1.2.5.5 TCustomRibbonSkin.GetResourceHandle Method

Override `GetResourceHandle` to supply skins in separate DLL.

Pascal

```
function GetResourceHandle: THandle; virtual;
```

C++

```
virtual __fastcall THandle GetResourceHandle();
```

Description

This method overrides `GetResourceHandle` to supply skins in a separate DLL file.

81.1.2.5.6 TCustomRibbonSkin.LoadSkin Method

Pascal

```
procedure LoadSkin; virtual;
```

C++

```
virtual __fastcall LoadSkin();
```

Description

This is LoadSkin, a member of class TCustomRibbonSkin.

81.1.2.6 TCustomRibbonSkin Properties

The properties of the TCustomRibbonSkin class are listed here.

Public Properties

	Name	Description
	Bitmap (see page 7376)	This is Bitmap, a member of class TCustomRibbonSkin.

81.1.2.6.1 TCustomRibbonSkin.Bitmap Property

Pascal

```
property Bitmap: TAlphaBitmap;
```

C++

```
__property TAlphaBitmap Bitmap;
```

Description

This is Bitmap, a member of class TCustomRibbonSkin.

81.1.3 TRibbonStyleActionBars Class

Action bar style.

Class Hierarchy

Pascal

```
TRibbonStyleActionBars = class(TXPStyleActionBars);
```

C++

```
class TRibbonStyleActionBars : public TXPStyleActionBars;
```

File

RibbonStyleActnCtrls

Description

TRibbonStyleActionBars is an action bar style.

81.1.3.1 TRibbonStyleActionBars Members

The following tables list the members exposed by TRibbonStyleActionBars.

Public Methods

	Name	Description
	Create (see page 7377)	Create TRibbonStyleActionBars (see page 7376) instance.
	Destroy (see page 7377)	Destroy TRibbonStyleActionBars (see page 7376) instance.
	DrawElement (see page 7378)	This is the overview for the DrawElement method overload.
	GetAddRemoveItemClass (see page 7380)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 7380)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7380)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7380)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7381)	This is GetScrollBtnClass, a member of class TRibbonStyleActionBars.
	GetSkinClass (see page 7381)	This is GetSkinClass, a member of class TRibbonStyleActionBars.

Public Properties

	Name	Description
	RibbonSkin (see page 7381)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.

81.1.3.2 TRibbonStyleActionBars.Create Constructor

Create TRibbonStyleActionBars (see page 7376) instance.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TRibbonStyleActionBars();
```

Description

This method creates a TRibbonStyleActionBars (see page 7376) instance.

See Also

Destroy (see page 7377)

81.1.3.3 TRibbonStyleActionBars.Destroy Destructor

Destroy TRibbonStyleActionBars (see page 7376) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRibbonStyleActionBars();
```

Description

This method destroys a TRibbonStyleActionBars (see page 7376) instance.

See Also

Create (see page 7377)

81.1.3.4 TRibbonStyleActionBars Methods

The methods of the TRibbonStyleActionBars class are listed here.

Public Methods

	Name	Description
	DrawElement (see page 7378)	This is the overview for the DrawElement method overload.
	GetAddRemoveItemClass (see page 7380)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 7380)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7380)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7380)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7381)	This is GetScrollBtnClass, a member of class TRibbonStyleActionBars.
	GetSkinClass (see page 7381)	This is GetSkinClass, a member of class TRibbonStyleActionBars.

81.1.3.4.1 DrawElement Method

This is the overview for the DrawElement method overload.

Overload List

	Name	Description
	TRibbonStyleActionBars.DrawElement (TSkinApplicationMenu, TCanvas, Integer, Integer, Byte) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinApplicationMenu, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinButton, TCanvas, TRect, Byte) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinCheckBox, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinComboBox, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinForm, TCanvas, TRect, Integer) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinGallery, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinMenu, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinQuickAccessToolbar, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinRadioButton, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinRibbon, TCanvas, TRect, Integer) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinRibbonGroup, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinScrollBar, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinSpinEdit, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.
	TRibbonStyleActionBars.DrawElement (TSkinStatusBar, TCanvas, TRect) (see page 7379)	Methods to draw ribbon skin element.

	TRibbonStyleActionBars.DrawElement (TSkinTab, TCanvas, TRect, Byte) (see page 7379)	Methods to draw ribbon skin element.

81.1.3.4.1.1 TRibbonStyleActionBars.DrawElement Method (TSkinApplicationMenu, TCanvas, Integer, Integer, Byte)

Methods to draw ribbon skin element.

Pascal

```
procedure DrawElement(Element: TSkinRibbon; Canvas: TCanvas; Rect: TRect; Height: Integer);
overload;
procedure DrawElement(Element: TSkinForm; Canvas: TCanvas; Rect: TRect; Height: Integer);
overload;
procedure DrawElement(Element: TSkinApplicationMenu; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinApplicationMenu; Canvas: TCanvas; const X: Integer; const
Y: Integer; const Opacity: Byte); overload;
procedure DrawElement(Element: TSkinTab; Canvas: TCanvas; Rect: TRect; Opacity: Byte = 255);
overload;
procedure DrawElement(Element: TSkinRibbonGroup; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinMenu; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinButton; Canvas: TCanvas; Rect: TRect; Opacity: Byte =
255); overload;
procedure DrawElement(Element: TSkinComboBox; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinSpinEdit; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinGallery; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinQuickAccessToolbar; Canvas: TCanvas; Rect: TRect);
overload;
procedure DrawElement(Element: TSkinScrollBar; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinStatusBar; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinCheckBox; Canvas: TCanvas; Rect: TRect); overload;
procedure DrawElement(Element: TSkinRadioButton; Canvas: TCanvas; Rect: TRect); overload;
```

C++

```
__fastcall DrawElement(TSkinRibbon Element, TCanvas Canvas, TRect Rect, int Height);
__fastcall DrawElement(TSkinForm Element, TCanvas Canvas, TRect Rect, int Height);
__fastcall DrawElement(TSkinApplicationMenu Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinApplicationMenu Element, TCanvas Canvas, const int X, const int Y,
const Byte Opacity);
__fastcall DrawElement(TSkinTab Element, TCanvas Canvas, TRect Rect, Byte Opacity = 255);
__fastcall DrawElement(TSkinRibbonGroup Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinMenu Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinButton Element, TCanvas Canvas, TRect Rect, Byte Opacity = 255);
__fastcall DrawElement(TSkinComboBox Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinSpinEdit Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinGallery Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinQuickAccessToolbar Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinScrollBar Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinStatusBar Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinCheckBox Element, TCanvas Canvas, TRect Rect);
__fastcall DrawElement(TSkinRadioButton Element, TCanvas Canvas, TRect Rect);
```

Description

DrawElement are methods to draw a particular ribbon skin element.

The parameters are:

- **Element**: The item to draw from the Element group.
- **Canvas**: Where the element is to be drawn.
- **Rect**: The rectangle for drawing the element.

- **Center**: Set to True to center the element within the given Rectangle.

81.1.3.4.2 TRibbonStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

81.1.3.4.3 TRibbonStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

81.1.3.4.4 TRibbonStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

81.1.3.4.5 TRibbonStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

81.1.3.4.6 TRibbonStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TRibbonStyleActionBars.

81.1.3.4.7 TRibbonStyleActionBars.GetSkinClass Method

Pascal

```
function GetSkinClass: TCustomRibbonSkinClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomRibbonSkinClass GetSkinClass() = 0;
```

Description

This is GetSkinClass, a member of class TRibbonStyleActionBars.

81.1.3.5 TRibbonStyleActionBars Properties

The properties of the TRibbonStyleActionBars class are listed here.

Public Properties

	Name	Description
	RibbonSkin (see page 7381)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.

81.1.3.5.1 TRibbonStyleActionBars.RibbonSkin Property

Enter a short sentence, starting with a verb, that states the basic purpose of the property.

Pascal

```
property RibbonSkin: TCustomRibbonSkin;
```

C++

```
__property TCustomRibbonSkin RibbonSkin;
```

Description

The long description elaborates on the short description, containing no more than a paragraph or two.

See Also

Chapter Title

Procedure Title

ClassName

PropertyName

MethodName

EventName

RoutineName

TypeName

VariableName

ConstantName

81.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRibbonElement (see page 7383)	Elements of Ribbon (see page 6713) that need to be skinned.
	TSkinApplicationMenu (see page 7384)	Individual skinned items of Application Menu.
	TSkinButton (see page 7386)	Individual skinned items of Button.
	TSkinCheckBox (see page 7390)	This is record RibbonStyleActnCtrls.TSkinCheckBox.
	TSkinComboBox (see page 7391)	This is record RibbonStyleActnCtrls.TSkinComboBox.
	TSkinForm (see page 7391)	Individual skinned items of Form.
	TSkinGallery (see page 7394)	Individual skinned items of Gallery.
	TSkinMenu (see page 7395)	Individual skinned items of Menu.
	TSkinQuickAccessToolbar (see page 7397)	Individual skinned items of Quick Access Toolbar.
	TSkinRadioButton (see page 7398)	This is record RibbonStyleActnCtrls.TSkinRadioButton.
	TSkinRibbon (see page 7399)	Individual skinned items of Ribbon (see page 6713).
	TSkinRibbonGroup (see page 7400)	Individual skinned items of Group.
	TSkinScrollBar (see page 7402)	Individual skinned items of ScrollBar.
	TSkinSpinEdit (see page 7403)	This is record RibbonStyleActnCtrls.TSkinSpinEdit.
	TSkinStatusBar (see page 7403)	Individual skinned items of Status Bar.
	TSkinTab (see page 7404)	Individual skinned items of Tab.

81.2.1 RibbonStyleActnCtrls.TRibbonElement Enumeration

Elements of Ribbon (see page 6713) that need to be skinned.

Pascal

```
TRibbonElement = (
  reRibbon,
  reForm,
  reApplicationMenu,
  reTab,
  reGroup,
  reMenu,
  reButton,
  reCombo,
  reSpinner,
  reGallery,
  reQuickAccessToolbar,
  reScrollBar,
  reStatusBar,
  reCheckBox,
  reRadioButton
);
```

C++

```
enum TRibbonElement {
  reRibbon,
  reForm,
  reApplicationMenu,
  reTab,
```

```
    reGroup,
    reMenu,
    reButton,
    reCombo,
    reSpinner,
    reGallery,
    reQuickAccessToolbar,
    reScrollBar,
    reStatusBar,
    reCheckBox,
    reRadioButton
  };
```

File

RibbonStyleActnCtrls

Description

TRibbonElement enumerates the available elements of the Ribbon (see page 6713) that need to be skinned.

Value	Meaning
reRibbon	Ribbon (see page 6713)
reForm	Form
reApplicationMenu	Application Menu
reTab	Tab
reGroup	Group
reMenu	Menu
reButton	Button
reCombo	Combo
reSpinner	Spinner
reGallery	Gallery
reQuickAccessToolbar	Quick Access Toolbar
reScrollBar	Scroll Bar
reStatusBar	Status Bar

81.2.2 RibbonStyleActnCtrls.TSkinApplicationMenu Enumeration

Individual skinned items of Application Menu.

Pascal

```
  TSkinApplicationMenu = (
    samButton,
    samButtonHover,
    samButtonPressed,
    samButtonShadow,
    samBorder,
    samMenuNormal,
    samMenuDisabled,
    samMenuHover,
```

```
    samMenuPressed,
    samMenuSplitNormal,
    samMenuSplitHover,
    samMenuSplitPressed,
    samMenuSubMenu
  );
```

C++

```
  enum TSkinApplicationMenu {
    samButton,
    samButtonHover,
    samButtonPressed,
    samButtonShadow,
    samBorder,
    samMenuNormal,
    samMenuDisabled,
    samMenuHover,
    samMenuPressed,
    samMenuSplitNormal,
    samMenuSplitHover,
    samMenuSplitPressed,
    samMenuSubMenu
  };
```

File

RibbonStyleActnCtrls

Description

TSkinApplicationMenu enumerates the individual skinned items of the Application Menu.

Value	Meaning
samButton	Button
samButtonHover	Button hover
samButtonPressed	Button pressed
samButtonShadow	Button shadow
samBorder	Border
samMenuNormal	Normal menu
samMenuDisabled	Disabled menu
samMenuHover	Menu hover
samMenuPressed	Pressed split menu
samMenuSplitNormal	Normal split menu
samMenuSplitHover	Split menu hover
samMenuSplitPressed	Pressed split menu
samMenuSubMenu	Submenu menu
samBorderTopStart	Top border start
samBorderTop	Top border
samBorderTopEnd	Top border end
samBorderBottomStart	Bottom border start
samBorderBottom	Bottom border
samBorderBottomEnd	Bottom border end

See Also

TRibbonElement (see page 7383)

81.2.3 RibbonStyleActnCtrls.TSkinButton Enumeration

Individual skinned items of Button.

Pascal

```
TSkinButton = (
  sbSmall,
  sbSmallDisabled,
  sbSmallHover,
  sbSmallPressed,
  sbSmallSelected,
  sbSmallHoverSelected,
  sbSmallInactive,
  sbLarge,
  sbLargeDisabled,
  sbLargeSplitActive,
  sbLargeHover,
  sbLargePressed,
  sbLargeSelected,
  sbLargeHoverSelected,
  sbSmallSplit,
  sbSmallSplitDisabled,
  sbSmallSplitHover,
  sbSmallSplitPressed,
  sbSmallSplitSelected,
  sbSmallSplitHoverSelected,
  sbSmallSplitInactive,
  sbSmallSplitHoverSplit,
  sbSmallSplitPressedSplit,
  sbSmallSplitInactiveSplit,
  sbLargeSplit,
  sbLargeSplitDisabled,
  sbLargeSplitHover,
  sbLargeSplitPressed,
  sbLargeSplitSelected,
  sbLargeSplitHoverSelected,
  sbLargeSplitHoverSplit,
  sbLargeSplitPressedSplit,
  sbArrow,
  sbArrowDisabled,
  sbArrowPressed,
  sbLargeArrowDown,
  sbLargeArrowDownDisabled,
  sbSeparator,
  sbDisabledSeparator,
  sbGroup,
  sbGroupStart,
  sbGroupStartHover,
  sbGroupStartPressed,
  sbGroupStartSelected,
  sbGroupStartHoverSelected,
  sbGroupStartInactive,
  sbGroupMiddle,
  sbGroupMiddleHover,
  sbGroupMiddlePressed,
  sbGroupMiddleSelected,
  sbGroupMiddleHoverSelected,
```

```
    sbGroupMiddleInactive,
    sbGroupEndSplitInactive,
    sbGroupStartSplitInactive,
    sbGroupSingleSplitInactive,
    sbGroupMiddleSplitInactive,
    sbGroupEnd,
    sbGroupEndHover,
    sbGroupEndPressed,
    sbGroupEndSelected,
    sbGroupEndHoverSelected,
    sbGroupEndInactive,
    sbGroupSingle,
    sbGroupSingleHover,
    sbGroupSinglePressed,
    sbGroupSingleSelected,
    sbGroupSingleHoverSelected,
    sbGroupSingleInactive,
    sbGroupSplit,
    sbGroupStartSplitHover,
    sbGroupMiddleSplitHover,
    sbGroupEndSplitHover,
    sbGroupSingleSplitHover,
    sbGroupStartInactiveSplit,
    sbGroupEndSplitInactiveSplit,
    sbGroupStartSplitHoverSplit,
    sbGroupEndSplitHoverSplit,
    sbGroupSingleSplitHoverSplit,
    sbGroupSingleSplitInactiveSplit,
    sbGroupMiddleSplitPressed,
    sbGroupMiddleSplitInactiveSplit,
    sbGroupMiddleSplitHoverSplit,
    sbGroupStartSplitInactiveSplit,
    sbGroupStartSplitPressed,
    sbGroupEndSplitPressed,
    sbGroupSingleSplitPressed,
    sbGroupStartSplitPressedSplit,
    sbGroupMiddleSplitPressedSplit,
    sbGroupEndSplitPressedSplit,
    sbGroupSingleSplitPressedSplit
  );
```

C++

```
  enum TSkinButton {
    sbSmall,
    sbSmallDisabled,
    sbSmallHover,
    sbSmallPressed,
    sbSmallSelected,
    sbSmallHoverSelected,
    sbSmallInactive,
    sbLarge,
    sbLargeDisabled,
    sbLargeSplitActive,
    sbLargeHover,
    sbLargePressed,
    sbLargeSelected,
    sbLargeHoverSelected,
    sbSmallSplit,
    sbSmallSplitDisabled,
    sbSmallSplitHover,
    sbSmallSplitPressed,
    sbSmallSplitSelected,
    sbSmallSplitHoverSelected,
    sbSmallSplitInactive,
    sbSmallSplitHoverSplit,
    sbSmallSplitPressedSplit,
```

```
    sbSmallSplitInactiveSplit,
    sbLargeSplit,
    sbLargeSplitDisabled,
    sbLargeSplitHover,
    sbLargeSplitPressed,
    sbLargeSplitSelected,
    sbLargeSplitHoverSelected,
    sbLargeSplitHoverSplit,
    sbLargeSplitPressedSplit,
    sbArrow,
    sbArrowDisabled,
    sbArrowPressed,
    sbLargeArrowDown,
    sbLargeArrowDownDisabled,
    sbSeparator,
    sbDisabledSeparator,
    sbGroup,
    sbGroupStart,
    sbGroupStartHover,
    sbGroupStartPressed,
    sbGroupStartSelected,
    sbGroupStartHoverSelected,
    sbGroupStartInactive,
    sbGroupMiddle,
    sbGroupMiddleHover,
    sbGroupMiddlePressed,
    sbGroupMiddleSelected,
    sbGroupMiddleHoverSelected,
    sbGroupMiddleInactive,
    sbGroupEndSplitInactive,
    sbGroupStartSplitInactive,
    sbGroupSingleSplitInactive,
    sbGroupMiddleSplitInactive,
    sbGroupEnd,
    sbGroupEndHover,
    sbGroupEndPressed,
    sbGroupEndSelected,
    sbGroupEndHoverSelected,
    sbGroupEndInactive,
    sbGroupSingle,
    sbGroupSingleHover,
    sbGroupSinglePressed,
    sbGroupSingleSelected,
    sbGroupSingleHoverSelected,
    sbGroupSingleInactive,
    sbGroupSplit,
    sbGroupStartSplitHover,
    sbGroupMiddleSplitHover,
    sbGroupEndSplitHover,
    sbGroupSingleSplitHover,
    sbGroupStartInactiveSplit,
    sbGroupEndSplitInactiveSplit,
    sbGroupStartSplitHoverSplit,
    sbGroupEndSplitHoverSplit,
    sbGroupSingleSplitHoverSplit,
    sbGroupSingleSplitInactiveSplit,
    sbGroupMiddleSplitPressed,
    sbGroupMiddleSplitInactiveSplit,
    sbGroupMiddleSplitHoverSplit,
    sbGroupStartSplitInactiveSplit,
    sbGroupStartSplitPressed,
    sbGroupEndSplitPressed,
    sbGroupSingleSplitPressed,
    sbGroupStartSplitPressedSplit,
    sbGroupMiddleSplitPressedSplit,
    sbGroupEndSplitPressedSplit,
```

```
    sbGroupSingleSplitPressedSplit
  };
```

File

RibbonStyleActnCtrls

Description

TSkinButton enumerates the individual skinned items of a Button.

Value	Meaning
sbSmall	Small
sbSmallDisabled	Disabled small
sbSmallHover	Small hover
sbSmallPressed	Pressed small
sbSmallSelected	Selected small
sbSmallHoverSelected	Selected small hover
sbSmallGroup	Small group
sbSmallGroupHover	Small group hover
sbSmallGroupPressed	Pressed small group
sbSmallGroupSelected	Selected small group
sbSmallGroupHoverSelected	Selected small group hover
sbLarge	Large
sbLargeDisabled	Disabled Large
sbLargeSplitActive	Active large split
sbLargeHover	Large hover
sbLargePressed	Pressed Large
sbLargeSelected	Selected Large
sbLargeHoverSelected	Selected Large hover
sbSmallSplit	Small split
sbSmallSplitDisabled	Disabled small split
sbSmallSplitHover	Small split hover
sbSmallSplitPressed	Pressed small split
sbSmallSplitSelected	Selected small split
sbSmallSplitHoverSelected	Selected small split hover
sbSmallSplitHoverSplit	Split small split hover
sbSmallSplitPressedSplit	Pressed split small split
sbLargeSplit	Large split
sbLargeSplitDisabled	Disabled large split
sbLargeSplitHover	Large split hover
sbLargeSplitPressed	Pressed large split
sbLargeSplitSelected	Selected large split
sbLargeSplitHoverSelected	Selected large split hover

sbLargeSplitHoverSplit	Split large split hover
sbLargeSplitPressedSplit	Pressed split large split
sbGroup	Group
sbGroupStart	Group start
sbGroupStartHover	Group start hover
sbGroupStartPressed	Pressed group start
sbGroupStartSelected	Selected group start
sbGroupStartHoverSelected	Selected group start hover
sbGroupStartInactive	Inactive group start
sbGroupMiddle	Group middle
sbGroupMiddleHover	Group middle hover
sbGroupMiddlePressed	Pressed group middle
sbGroupMiddleSelected	Selected group middle
sbGroupMiddleHoverSelected	Selected group middle hover
sbGroupMiddleInactive	Inactive group middle
sbGroupEnd	Group end
sbGroupEndHover	Group end hover
sbGroupEndPressed	Pressed group end
sbGroupEndSelected	Selected group end
sbGroupEndHoverSelected	Selected group end hover
sbGroupEndInactive	Inactive group end
sbGroupSingle	Single group
sbGroupSingleHover	Single group hover
sbGroupSinglePressed	Pressed single group
sbGroupSingleSelected	Selected single group
sbGroupSingleHoverSelected	Selected single group hover
sbGroupSingleInactive	Inactive single group

See Also

TRibbonElement (see page 7383)

81.2.4 RibbonStyleActnCtrls.TSkinCheckBox Enumeration

Pascal

```
TSkinCheckBox = (
  scNormal,
  scDisabled,
  scHover,
  scPressed,
  scSelected,
  scHoverSelected,
```

```
    scDisabledSelected
  );
```

C++

```
  enum TSkinCheckBox {
    scNormal,
    scDisabled,
    scHover,
    scPressed,
    scSelected,
    scHoverSelected,
    scDisabledSelected
  };
```

File

RibbonStyleActnCtrls

Description

This is record RibbonStyleActnCtrls.TSkinCheckBox.

81.2.5 RibbonStyleActnCtrls.TSkinComboBox Enumeration

Pascal

```
  TSkinComboBox = (
    scButton,
    scButtonInactive,
    scButtonHover,
    scButtonPressed,
    scButtonIcon
  );
```

C++

```
  enum TSkinComboBox {
    scButton,
    scButtonInactive,
    scButtonHover,
    scButtonPressed,
    scButtonIcon
  };
```

File

RibbonStyleActnCtrls

Description

This is record RibbonStyleActnCtrls.TSkinComboBox.

81.2.6 RibbonStyleActnCtrls.TSkinForm Enumeration

Individual skinned items of Form.

Pascal

```
  TSkinForm = (
    sfLeftFrame,
    sfLeftFrameDisabled,
```

```
    sfLeftFrameTop,
    sfLeftFrameTopDisabled,
    sfRightFrame,
    sfRightFrameDisabled,
    sfRightFrameTop,
    sfRightFrameTopDisabled,
    sfBottomFrame,
    sfBottomFrameDisabled,
    sfButton,
    sfButtonHover,
    sfButtonPressed,
    sfNone,
    sfMinimize,
    sfMinimizeHover,
    sfMinimizePressed,
    sfMinimizeDisabled,
    sfMaximize,
    sfMaximizeHover,
    sfMaximizePressed,
    sfMaximizeDisabled,
    sfRestore,
    sfRestoreHover,
    sfRestorePressed,
    sfRestoreDisabled,
    sfClose,
    sfCloseHover,
    sfClosePressed,
    sfCloseDisabled,
    sfHelp,
    sfHelpHover,
    sfHelpPressed,
    sfHelpDisabled
  );
```

C++

```
  enum TSkinForm {
    sfLeftFrame,
    sfLeftFrameDisabled,
    sfLeftFrameTop,
    sfLeftFrameTopDisabled,
    sfRightFrame,
    sfRightFrameDisabled,
    sfRightFrameTop,
    sfRightFrameTopDisabled,
    sfBottomFrame,
    sfBottomFrameDisabled,
    sfButton,
    sfButtonHover,
    sfButtonPressed,
    sfNone,
    sfMinimize,
    sfMinimizeHover,
    sfMinimizePressed,
    sfMinimizeDisabled,
    sfMaximize,
    sfMaximizeHover,
    sfMaximizePressed,
    sfMaximizeDisabled,
    sfRestore,
    sfRestoreHover,
    sfRestorePressed,
    sfRestoreDisabled,
    sfClose,
    sfCloseHover,
    sfClosePressed,
    sfCloseDisabled,
```

```
    sfHelp,
    sfHelpHover,
    sfHelpPressed,
    sfHelpDisabled
  };
```

File

RibbonStyleActnCtrls

Description

TSkinForm enumerates the individual skinned items of the Form.

Value	Meaning
sfLeftFrame	Left frame
sfLeftFrameTop	Left frame top
sfLeftFrameTopDisabled	Disabled left frame top
sfRightFrame	Right frame
sfRightFrameTop	Right frame top
sfRightFrameTopDisabled	Disabled right frame top
sfBottomFrame	Bottom frame
sfButton	Button
sfButtonHover	Button hover
sfButtonPressed	Button pressed
sfMinimize	Minimize
sfMinimizeHover	Minimize hover
sfMinimizePressed	Minimize pressed
sfMinimizeDisabled	Minimize disabled
sfMaximize	Maximize
sfMaximizeHover	Maximize hover
sfMaximizePressed	Maximize pressed
sfMaximizeDisabled	Maximize disabled
sfRestore	Restore
sfRestoreHover	Restore hover
sfRestorePressed	Restore pressed
sfRestoreDisabled	Restore disabled
sfClose	Close
sfCloseHover	Close hover
sfClosePressed	Close pressed
sfCloseDisabled	Close disabled

See Also

TRibbonElement (see page 7383)

81.2.7 RibbonStyleActnCtrls.TSkinGallery Enumeration

Individual skinned items of Gallery.

Pascal

```
TSkinGallery = (
  sgButton,
  sgUpButton,
  sgUpButtonHover,
  sgUpButtonPressed,
  sgUpButtonDisabled,
  sgDownButton,
  sgDownButtonHover,
  sgDownButtonPressed,
  sgDownButtonDisabled,
  sgMoreButton,
  sgMoreButtonHover,
  sgMoreButtonHoverDisabled
);
```

C++

```
enum TSkinGallery {
  sgButton,
  sgUpButton,
  sgUpButtonHover,
  sgUpButtonPressed,
  sgUpButtonDisabled,
  sgDownButton,
  sgDownButtonHover,
  sgDownButtonPressed,
  sgDownButtonDisabled,
  sgMoreButton,
  sgMoreButtonHover,
  sgMoreButtonHoverDisabled
};
```

File

RibbonStyleActnCtrls

Description

TSkinGallery enumerates the individual skinned items of a Gallery.

Value	Meaning
sgBackground	Background
sgUpButton	Up button
sgUpButtonHover	Up button hover
sgUpButtonPressed	Pressed up button
sgupButtonDisabled	Disabled up button
sgDownButton	Down button
sgDownButtonHover	Down button hover
sgDownButtonPressed	Pressed down button
sgDownButtonDisabled	Disabled down button

sgMoreButton	More button
sgMoreButtonHover	More button hover
sgUpIcon	Up icon
sgDownIcon	Down icon
sgMoreIcon	More icon

See Also

TRibbonElement (see page 7383)

81.2.8 RibbonStyleActnCtrls.TSkinMenu Enumeration

Individual skinned items of Menu.

Pascal

```
TSkinMenu = (
  smBackground,
  smBanner,
  smNormal,
  smSelected,
  smHover,
  smDisabled,
  smCheck,
  smCheckDisabled,
  smCheckIcon,
  smCheckDisabledIcon,
  smRadio,
  smRadioDisabled,
  smSeparator,
  smArrowLeftIcon,
  smArrowLeftDisabledIcon,
  smArrowRightIcon,
  smArrowRightDisabledIcon,
  smResize,
  smResizeVertical,
  smResizeBoth,
  smRichContentHover,
  smRichContentDisabled,
  smScrollButton,
  smScrollButtonHover,
  smScrollButtonUpIcon,
  smScrollButtonDownIcon,
  smOptionButton,
  smOptionButtonHover,
  smOptionButtonDisabled
);
```

C++

```
enum TSkinMenu {
  smBackground,
  smBanner,
  smNormal,
  smSelected,
  smHover,
  smDisabled,
  smCheck,
  smCheckDisabled,
```

```
    smCheckIcon,
    smCheckDisabledIcon,
    smRadio,
    smRadioDisabled,
    smSeparator,
    smArrowLeftIcon,
    smArrowLeftDisabledIcon,
    smArrowRightIcon,
    smArrowRightDisabledIcon,
    smResize,
    smResizeVertical,
    smResizeBoth,
    smRichContentHover,
    smRichContentDisabled,
    smScrollButton,
    smScrollButtonHover,
    smScrollButtonUpIcon,
    smScrollButtonDownIcon,
    smOptionButton,
    smOptionButtonHover,
    smOptionButtonDisabled
  };
```

File

RibbonStyleActnCtrls

Description

TSkinMenu enumerates the individual skinned items of a Menu.

Value	Meaning
smBackground	Background
smBanner	Banner
smNormal	Normal
smSelected	Selected
smSelectedStart	Selected start
smSelectedEnd	Selected end
smHover	Hover
smHoverStart	Hover start
smHoverEnd	Hover end
smHoverSelected	Selected hover
smDisabled	Disabled
smCheck	Check
smCheckDisabled	Disabled check
smCheckIcon	Check icon
smCheckDisabledIcon	Disabled check icon
smRadio	Radio
smRadioDisabled	Disabled radio
smSeparator	Separator
smArrowLeftIcon	Left arrow icon
smArrowRightIcon	Right arrow icon

smResize	Resize
smResizeVertical	Vertical resize
smResizeBoth	Resize both
smRichContentHover	Rich content hover
smRichContentDisabled	Disabled rich content
smScrollButton	Scroll button
smScrollButtonHover	Scroll button hover
smScrollButtonUpIcon	Up scroll button icon
smScrollButtonDownIcon	Down scroll button icon
smOptionButton	Option button
smOptionButtonHover	Option button hover
smOptionButtonDisabled	Disabled option button

See Also

TRibbonElement (see page 7383)

81.2.9 RibbonStyleActnCtrls.TSkinQuickAccessToolbar Enumeration

Individual skinned items of Quick Access Toolbar.

Pascal

```
TSkinQuickAccessToolbar = (
  sqBackground,
  sqStart,
  sqStartRTL,
  sqMiddle,
  sqEnd,
  sqEndRTL,
  sqButton,
  sqButtonHover,
  sqButtonPressed,
  sqButtonCustomizeIcon,
  sqButtonMoreIcon,
  sqGroup,
  sqGroupHover,
  sqGroupPressed
);
```

C++

```
enum TSkinQuickAccessToolbar {
  sqBackground,
  sqStart,
  sqStartRTL,
  sqMiddle,
  sqEnd,
  sqEndRTL,
  sqButton,
  sqButtonHover,
  sqButtonPressed,
```

```
    sqButtonCustomizeIcon,
    sqButtonMoreIcon,
    sqGroup,
    sqGroupHover,
    sqGroupPressed
  };
```

File

RibbonStyleActnCtrls

Description

TSkinQuickAccessToolbar enumerates the individual skinned items of the Quick Access Toolbar.

Value	Meaning
sqBackground	Background
sqStart	Start
sqMiddle	Middle
sqEnd	End
sqButton	Button
sqButtonHover	Button hover
sqButtonPressed	Pressed button
sqButtonCustomizeIcon	Customize button icon
sqButtonMoreIcon	More button icon
sqGroup	Group
sqGroupHover	Group hover
sqGroupPressed	Pressed group

See Also

TRibbonElement (see page 7383)

81.2.10 RibbonStyleActnCtrls.TSkinRadioButton Enumeration

Pascal

```
  TSkinRadioButton = (
    sbNormal,
    sbDisabled,
    sbHover,
    sbPressed,
    sbSelected,
    sbHoverSelected,
    sbDisabledSelected
  );
```

C++

```
  enum TSkinRadioButton {
    sbNormal,
    sbDisabled,
    sbHover,
```

```
    sbPressed,
    sbSelected,
    sbHoverSelected,
    sbDisabledSelected
  };
```

File

RibbonStyleActnCtrls

Description

This is record RibbonStyleActnCtrls.TSkinRadioButton.

81.2.11 RibbonStyleActnCtrls.TSkinRibbon Enumeration

Individual skinned items of Ribbon (see page 6713).

Pascal

```
  TSkinRibbon = (
    srBackground,
    srBackgroundDisabled,
    srHeader,
    srBody,
    srPage,
    srHelp
  );
```

C++

```
  enum TSkinRibbon {
    srBackground,
    srBackgroundDisabled,
    srHeader,
    srBody,
    srPage,
    srHelp
  };
```

File

RibbonStyleActnCtrls

Description

TSkinRibbon enumerates the individual skinned items of the Ribbon (see page 6713).

Value	Meaning
srBackground	Background
srBackgroundDisabled	Disabled background
srHeader	Header
srBody	Body
srPageStart	Page start
srPage	Page
srPageEnd	Page end
srHelp	Help

See Also

TRibbonElement (see page 7383)

81.2.12 RibbonStyleActnCtrls.TSkinRibbonGroup Enumeration

Individual skinned items of Group.

Pascal

```
TSkinRibbonGroup = (
  srgBackground,
  srgBackgroundHover,
  srgBackgroundDisabled,
  srgCaption,
  srgCaptionHover,
  srgCaptionDisabled,
  srgDialogButton,
  srgDialogButtonHover,
  srgDialogButtonPressed,
  srgDialogButtonIcon,
  srgDialogButtonIconHover,
  srgDialogButtonIconPressed,
  srgMinimizeButton,
  srgMinimizeButtonHover,
  srgMinimizeButtonPressed,
  srgMinimizeButtonIcon,
  srgMinimizeButtonPressedIcon
);
```

C++

```
enum TSkinRibbonGroup {
  srgBackground,
  srgBackgroundHover,
  srgBackgroundDisabled,
  srgCaption,
  srgCaptionHover,
  srgCaptionDisabled,
  srgDialogButton,
  srgDialogButtonHover,
  srgDialogButtonPressed,
  srgDialogButtonIcon,
  srgDialogButtonIconHover,
  srgDialogButtonIconPressed,
  srgMinimizeButton,
  srgMinimizeButtonHover,
  srgMinimizeButtonPressed,
  srgMinimizeButtonIcon,
  srgMinimizeButtonPressedIcon
};
```

File

RibbonStyleActnCtrls

Description

TSkinRibbonGroup enumerates the individual skinned items of a Group.

Value	Meaning
srgBackground	Background
srgBackgroundStart	Background start
srgBackgroundEnd	Background end
srgBackgroundHover	Background hover
srgBackgroundHoverStart	Background start hover
srgBackgroundHoverEnd	Background end hover
srgBackgroundDisabled	Disabled background
srgBackgroundDisabledStart	Disabled background start
srgBackgroundDisabledEnd	Disabled background end
srgCaption	Caption
srgCaptionHover	Caption hover
srgCaptionDisabled	Disabled caption
srgDialogButton	Dialog button
srgDialogButtonHover	Dialog button hover
srgDialogButtonPressed	Pressed dialog button
srgDialogButtonIcon	Dialog button icon
srgDialogButtonIconHover	Dialog button icon hover
srgDialogButtonIconPressed	Pressed dialog button icon
srgMinimizeButton	Minimize button
srgMinimizeButtonStart	Minimize button start
srgMinimizeButtonEnd	Minimize button end
srgMinimizeButtonHover	Minimize button hover
srgMinimizeButtonHoverStart	Minimize button hover start
srgMinimizeButtonHoverEnd	Minimize button hover end
srgMinimizeButtonPressed	Pressed minimize button
srgMinimizeButtonPressedStart	Pressed minimize button start
srgMinimizeButtonPressedEnd	Pressed minimize button end
srgMinimizeButtonIcon	Minimize button icon
srgMinimizeButtonPressedIcon	Pressed minimize button icon
srgScrollLeft	Left scroll
srgScrollLeftHover	Left scroll hover
srgScrollRight	Scroll right
srgScrollRightHover	Scroll right hover

See Also

TRibbonElement (see page 7383)

81.2.13 RibbonStyleActnCtrls.TSkinScrollBar Enumeration

Individual skinned items of ScrollBar.

Pascal

```
TSkinScrollBar = (
  ssbBar,
  ssbBarHover,
  ssbBarPressed,
  ssbBarIcon,
  ssbButton,
  ssbButtonActive,
  ssbButtonDisabled,
  ssbButtonHover,
  ssbButtonPressed,
  ssbButtonUpIcon,
  ssbButtonDownIcon
);
```

C++

```
enum TSkinScrollBar {
  ssbBar,
  ssbBarHover,
  ssbBarPressed,
  ssbBarIcon,
  ssbButton,
  ssbButtonActive,
  ssbButtonDisabled,
  ssbButtonHover,
  ssbButtonPressed,
  ssbButtonUpIcon,
  ssbButtonDownIcon
};
```

File

RibbonStyleActnCtrls

Description

TSkinScrollBar enumerates the individual skinned items of a ScrollBar.

Value	Meaning
ssbBar	Bar
ssbBarHover	Bar hover
ssbBarPressed	Pressed bar
ssbBarIcon	Bar icon
ssbButton	Button
ssbButtonActive	Active button
ssbButtonDisabled	Disabled button
ssbButtonHover	Button hover
ssbButtonPressed	Pressed button
ssbButtonUpIcon	Button up icon
ssbButtonDownIcon	Button down icon

See Also

TRibbonElement (see page 7383)

81.2.14 RibbonStyleActnCtrls.TSkinSpinEdit Enumeration

Pascal

```
TSkinSpinEdit = (
  ssButton,
  ssButtonDisabled,
  ssButtonHover,
  ssButtonInactive,
  ssButtonPressed,
  ssButtonUp,
  ssButtonDown,
  ssButtonUpIcon,
  ssButtonDownIcon
);
```

C++

```
enum TSkinSpinEdit {
  ssButton,
  ssButtonDisabled,
  ssButtonHover,
  ssButtonInactive,
  ssButtonPressed,
  ssButtonUp,
  ssButtonDown,
  ssButtonUpIcon,
  ssButtonDownIcon
};
```

File

RibbonStyleActnCtrls

Description

This is record RibbonStyleActnCtrls.TSkinSpinEdit.

81.2.15 RibbonStyleActnCtrls.TSkinStatusBar Enumeration

Individual skinned items of Status Bar.

Pascal

```
TSkinStatusBar = (
  ssbrBackground,
  ssbrNotification,
  ssbrControl,
  ssbrGripper
);
```

C++

```
enum TSkinStatusBar {
  ssbrBackground,
  ssbrNotification,
```

```
    ssbrControl,
    ssbrGripper
  };
```

File

RibbonStyleActnCtrls

Description

TSkinStatusBar enumerates the individual skinned items of a Status Bar.

Value	Meaning
ssbrBackground	Background
ssbrNotification	Notification
ssbrControl	Control
ssbrGripper	Gripper

See Also

TRibbonElement (see page 7383)

81.2.16 RibbonStyleActnCtrls.TSkinTab Enumeration

Individual skinned items of Tab.

Pascal

```
  TSkinTab = (
    stBackground,
    stNormal,
    stHover,
    stHoverUnselected,
    stHoverSelected,
    stKeyTipsActive,
    stScrollLeft,
    stScrollLeftHover,
    stScrollLeftPressed,
    stScrollRight,
    stScrollRightHover,
    stScrollRightPressed,
    stSeparator,
    stScrollGroupLeft,
    stScrollGroupLeftHover,
    stScrollGroupRight,
    stScrollGroupRightHover
  );
```

C++

```
  enum TSkinTab {
    stBackground,
    stNormal,
    stHover,
    stHoverUnselected,
    stHoverSelected,
    stKeyTipsActive,
    stScrollLeft,
    stScrollLeftHover,
    stScrollLeftPressed,
    stScrollRight,
```

```
    stScrollRightHover,
    stScrollRightPressed,
    stSeparator,
    stScrollGroupLeft,
    stScrollGroupLeftHover,
    stScrollGroupRight,
    stScrollGroupRightHover
  };
```

File

RibbonStyleActnCtrls

Description

TSkinTab enumerates the individual skinned items of a Tab.

Value	Meaning
stBackground	Background
stHoverUnselected	Unselected hover
stHoverUnselectedStart	Unselected start hover
stHoverUnselectedEnd	Unselected end hover
stHover	Hover
stHoverStart	Start hover
stHoverEnd	End hover
stNormal	Normal
stNormalStart	Start normal
stNormalEnd	End normal
stHoverSelected	Selected hover
stScrollLeft	Left scroll
stScrollLeftPressed	Pressed Left scroll
stScrollLeftHover	Left scroll hover
stScrollRight	Right scroll
stScrollRightPressed	Pressed right scroll
stScrollRightHover	Right scroll hover
stKeyTipsActive	KeyTips active
stKeyTipsActiveStart	KeyTips active start
stKeyTipsActiveEnd	KeyTips active end
stSeparator	Separator

See Also

TRibbonElement (see page 7383)

81.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCustomRibbonSkinClass (see page 7406)	This is type RibbonStyleActnCtrls.TCustomRibbonSkinClass.

81.3.1 RibbonStyleActnCtrls.TCustomRibbonSkinClass Type

Pascal

```
TCustomRibbonSkinClass = class of TCustomRibbonSkin;
```

C++

```
TCustomRibbonSkin TCustomRibbonSkinClass;
```

File

RibbonStyleActnCtrls

Description

This is type RibbonStyleActnCtrls.TCustomRibbonSkinClass.

82 Rio Namespace

This is namespace Rio.

Topics

Name	Description
Classes (see page 7408)	The following table lists classes in this documentation.
Interfaces (see page 7420)	The following table lists interfaces in this documentation.
Types (see page 7422)	The following table lists types in this documentation.
Constants (see page 7423)	The following table lists constants in this documentation.

Classes

	Name	Description
	TRIO (see page 7408)	TRIO is the base class for components that generate calls to remote interfaced objects.

Constants

Name	Description
StubSize (see page 7423)	This is constant Rio.StubSize.

Interfaces

	Name	Description
	IRIOAccess (see page 7420)	IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

Types

Name	Description
TAfterExecuteEvent (see page 7422)	This is type Rio.TAfterExecuteEvent.
TBeforeExecuteEvent (see page 7422)	This is type Rio.TBeforeExecuteEvent.

82.1 Classes

The following table lists classes in this documentation.

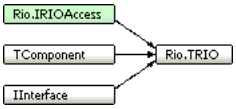
Classes

	Name	Description
	TRIO (see page 7408)	TRIO is the base class for components that generate calls to remote interfaced objects.

82.1.1 TRIO Class

TRIO is the base class for components that generate calls to remote interfaced objects.

Class Hierarchy



Pascal

```
TRIO = class(TComponent, IInterface, IRIOAccess);
```

C++

```
class TRIO : public TComponent, public IInterface, public IRIOAccess;
```

File

Rio

Description

Use TRIO as a base class for components that generate statically-linked calls to invokable interfaces. When an application casts a TRIO descendant to a registered invokable interface, it dynamically generates an in-memory method table, providing an implementation to that invokable interface. TRIO uses two helper objects when generating this implementation. These helper objects implement the IOPConvert (see page 6152) and IWebNode interfaces. IOPConvert (see page 6152) handles the marshaling and unmarshaling of calls to methods on the generated method table, and IWebNode receives the marshaled message call, executes it (typically by sending it to a remote server application), and returns the results of executing the call. These interfaces do not assume any particular encoding or transport protocol. Each TRIO descendant uses its own protocol-specific helper objects to implement these interfaces.

Before TRIO can generate a method table for an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry (see page 4471) function to obtain access to the registry, and call its RegisterInterface method.

Do not create instances of TRIO. To add a component that generates calls to a registered invokable interface, use a TRIO descendant such as THTTPRIO.

See Also

THTTPRio

InvRegistry

TInvokableClassRegistry (see page 4379)

IWebNode

IOPConvert (see page 6152)

IRIOAccess (see page 7420)

82.1.1.1 TRIO Members

The following tables list the members exposed by TRIO.

Protected Fields

	Name	Description
	FConverter (see page 7411)	This is FConverter, a member of class TRIO.
	FIID (see page 7411)	This is FIID, a member of class TRIO.
	FWebNode (see page 7411)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 7411)	This is IntfMD, a member of class TRIO.

Protected Methods

	Name	Description
V	DoAfterExecute (see page 7413)	Generates an OnAfterExecute (see page 7417) event.
V	DoBeforeExecute (see page 7413)	Generates an OnBeforeExecute (see page 7418) event.
V	GetResponseStream (see page 7414)	Returns a newly created stream that receives the results of executing a method call.
V	QueryInterface (see page 7415)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
V	AfterConstruction (see page 7412)	Updates the reference count after the last constructor has executed.
V	BeforeDestruction (see page 7412)	Prevents the destruction of a TRIO (see page 7408) instance if the reference count is greater than zero.
V	Create (see page 7410)	Creates an instance of TRIO (see page 7408).
V	Destroy (see page 7410)	Destroys the instance of a TRIO (see page 7408) object.
	GetRIO (see page 7421)	Returns the value of the RIO (see page 7421) property.
V	NewInstance (see page 7414)	Allocates memory for each instance of a remote interfaced object.

Public Properties

	Name	Description
	Converter (see page 7415)	Handles the marshaling and unmarshaling of calls to an invokable interface.
R	RefCount (see page 7416)	Maintains a reference count on the use of the TRIO (see page 7408) object's interface.
R	RIO (see page 7421)	Provides access to the remote interfaced object that implements this interface.
R	SOAPHeaders (see page 7416)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	WebNode (see page 7417)	Handles the execution of method calls on the invokable interface.

Published Events

	Name	Description
	OnAfterExecute (see page 7417)	Occurs when TRIO (see page 7408) receives the results from executing a method of the invokable interface.
	OnBeforeExecute (see page 7418)	Occurs before TRIO (see page 7408) sends an encoded method call for execution.
	OnGetAttachment (see page 7419)	Occurs when TRIO (see page 7408) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 7419)	Occurs when TRIO (see page 7408) sends each attachment that is the value of an argument of the method call on the invokable interface.

82.1.1.2 TRIO.Create Constructor

Creates an instance of TRIO (see page 7408).

Pascal

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

C++

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

Description

Call Create to instantiate a TRIO (see page 7408) descendant at runtime. TRIO (see page 7408) descendants added to forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TRIO (see page 7408) instance. It becomes the value of the Owner property. If the TRIO (see page 7408) instance is created with AOwner set to nil (Delphi) or NULL (C++), then it is automatically freed when its reference count drops to zero.

See Also

RefCount (see page 7416)

Owner

82.1.1.3 TRIO.Destroy Destructor

Destroys the instance of a TRIO (see page 7408) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRIO();
```

Description

There is no need to explicitly free a TRIO (see page 7408) instance.

If the TRIO (see page 7408) object is created with an Owner, then that Owner handles the freeing of the TRIO (see page 7408) instance. If the TRIO (see page 7408) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 7410)

Owner

RefCount (see page 7416)

82.1.1.4 TRIO Fields

The fields of the TRIO class are listed here.

Protected Fields

	Name	Description
	FConverter (see page 7411)	This is FConverter, a member of class TRIO.
	FIID (see page 7411)	This is FIID, a member of class TRIO.
	FWebNode (see page 7411)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 7411)	This is IntfMD, a member of class TRIO.

82.1.1.4.1 TRIO.FConverter Field

Pascal

```
FConverter: IOPConvert;
```

C++

```
IOPConvert FConverter;
```

Description

This is FConverter, a member of class TRIO.

82.1.1.4.2 TRIO.FIID Field

Pascal

```
FIID: TGUID;
```

C++

```
TGUID FIID;
```

Description

This is FIID, a member of class TRIO.

82.1.1.4.3 TRIO.FWebNode Field

Pascal

```
FWebNode: IWebNode;
```

C++

```
IWebNode FWebNode;
```

Description

This is FWebNode, a member of class TRIO.

82.1.1.4.4 TRIO.IntfMD Field

Pascal

```
IntfMD: TIntfMetaData;
```

C++

```
TIntfMetaData IntfMD;
```

Description

This is IntfMD, a member of class TRIO.

82.1.1.5 TRIO Methods

The methods of the TRIO class are listed here.

Protected Methods

	Name	Description
	DoAfterExecute (see page 7413)	Generates an OnAfterExecute (see page 7417) event.
	DoBeforeExecute (see page 7413)	Generates an OnBeforeExecute (see page 7418) event.
	GetResponseStream (see page 7414)	Returns a newly created stream that receives the results of executing a method call.
	QueryInterface (see page 7415)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 7412)	Updates the reference count after the last constructor has executed.
	BeforeDestruction (see page 7412)	Prevents the destruction of a TRIO (see page 7408) instance if the reference count is greater than zero.
	NewInstance (see page 7414)	Allocates memory for each instance of a remote interfaced object.

82.1.1.5.1 TRIO.AfterConstruction Method

Updates the reference count after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the TRIO (see page 7408) object's last constructor has executed. Do not call it explicitly in your applications.

TRIO (see page 7408) overrides the AfterConstruction method to update the RefCount (see page 7416) property. TRIO (see page 7408) artificially increments the reference count when it is first created to prevent changes to the reference count from causing the TRIO (see page 7408) instance to be freed before it is fully instantiated. AfterConstruction returns the reference count back to an appropriate level, now that the constructor is finished.

See Also

RefCount (see page 7416)

Create (see page 7410)

NewInstance (see page 7414)

82.1.1.5.2 TRIO.BeforeDestruction Method

Prevents the destruction of a TRIO (see page 7408) instance if the reference count is greater than zero.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called automatically before the TRIO (see page 7408) object's first destructor executes. Do not call it explicitly in your applications.

TRIO (see page 7408) overrides the BeforeDestruction method to check the RefCount (see page 7416) property. If RefCount (see page 7416) is greater than zero, then there is still an outstanding reference to the TRIO (see page 7408) object's interface,. and BeforeDestruction raises an exception to prevent the destruction of this object.

See Also

RefCount (see page 7416)

82.1.1.5.3 TRIO.DoAfterExecute Method

Generates an OnAfterExecute (see page 7417) event.

Pascal

```
procedure DoAfterExecute(const MethodName: string; Response: TStream); virtual;
```

C++

```
virtual __fastcall DoAfterExecute(const UnicodeString MethodName, TStream Response);
```

Description

TRIO (see page 7408) calls DoAfterExecute internally after it receives the results of a method call but before it unmarshals those results. This method simply generates an OnAfterExecute (see page 7417) event, passing the MethodName and Response parameters to the event handler. Override this method to perform class-specific actions before or after the OnAfterExecute (see page 7417) event.

MethodName is the name of the method that was just executed on the server.

Response is a stream from that contains the results that the server returned after executing the specified method.

See Also

OnAfterExecute (see page 7417)

82.1.1.5.4 TRIO.DoBeforeExecute Method

Generates an OnBeforeExecute (see page 7418) event.

Pascal

```
procedure DoBeforeExecute(const MethodName: string; Request: TStream); virtual;
```

C++

```
virtual __fastcall DoBeforeExecute(const UnicodeString MethodName, TStream Request);
```

Description

TRIO (see page 7408) calls DoBeforeExecute internally before it sends a SOAP packet to the server for execution. This method simply generates an OnBeforeExecute (see page 7418) event, passing the MethodName parameter and the string in the Request parameter to the event handler. Override this method to perform class-specific actions before or after the OnBeforeExecute (see page 7418) event.

MethodName is the name of the method that the encoded SOAP request calls.

Request is a stream from which the encoded SOAP request packet can be read.

See Also

OnBeforeExecute (see page 7418)

82.1.1.5.5 TRIO.GetResponseStream Method

Returns a newly created stream that receives the results of executing a method call.

Pascal

```
function GetResponseStream(BindingType: TWebServiceBindingType): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetResponseStream(TWebServiceBindingType BindingType);
```

Description

TRIO (see page 7408) calls GetResponseStream internally when it executes a method call. It passes the returned stream as the Response parameter of the Execute method on the interface that the WebNode (see page 7417) property specifies. After this stream receives the encoded response of a method call, the RIO (see page 7421) component decodes the response and frees the stream.

BindingType indicates the expected coding of the response that the stream will hold. GetResponseStream can use this information to determine what type of stream to create.

See Also

Execute

WebNode (see page 7417)

82.1.1.5.6 TRIO.NewInstance Method

Allocates memory for each instance of a remote interfaced object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TRIO (see page 7408) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a RefCount (see page 7416) of zero. The AfterConstruction (see page 7412) method decrements the value of RefCount (see page 7416) after all constructors have executed.

See Also

System::TObject::InstanceSize

System::TObject::NewInstance

AfterConstruction (see page 7412)

82.1.1.5.7 TRIO.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts TRIO (see page 7408) to an invokable interface using the as operator.

TRIO (see page 7408) can only return a single type of interface pointer. The first time an application calls QueryInterface, TRIO (see page 7408) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this TRIO (see page 7408) instance. If the TRIO (see page 7408) object has already been typecast to an interface, then IID must identify that interface. If the TRIO (see page 7408) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless TRIO (see page 7408) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE otherwise.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry (see page 4471)

RegisterInterface (see page 4404)

RefCount (see page 7416)

82.1.1.6 TRIO Properties

The properties of the TRIO class are listed here.

Public Properties

	Name	Description
	Converter (see page 7415)	Handles the marshaling and unmarshaling of calls to an invokable interface.
R	RefCount (see page 7416)	Maintains a reference count on the use of the TRIO (see page 7408) object's interface.
R	SOAPHeaders (see page 7416)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	WebNode (see page 7417)	Handles the execution of method calls on the invokable interface.

82.1.1.6.1 TRIO.Converter Property

Handles the marshaling and unmarshaling of calls to an invokable interface.

Pascal

```
property Converter: IOPConvert;
```

C++

```
__property IOPConvert Converter;
```

Description

TRIO (see page 7408) uses Converter internally to marshal and unmarshal calls to an invokable interface. That is, when the TRIO (see page 7408) object has been typecast to an invokable interface, it implements each method call by using Converter to marshal the call into an encoding protocol, and to unmarshal any return values after that call is executed.

Each TRIO (see page 7408) descendant uses its own protocol-specific helper object as the value of the Converter property.

See Also

WebNode (see page 7417)

TOPToSoapDomConvert (see page 6167)

82.1.1.6.2 TRIO.RefCount Property

Maintains a reference count on the use of the TRIO (see page 7408) object's interface.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount keeps track of the usage on the TRIO (see page 7408) object's interface. RefCount is incremented by the interface's _AddRef method (Delphi) or AddRef method (C++) and decremented by its _Release method (Delphi) or Release method (C++).

If the TRIO (see page 7408) object is created without an Owner, then when RefCount drops to zero, the TRIO (see page 7408) instance is automatically freed.

See Also

QueryInterface (see page 7415)

Create (see page 7410)

_AddRef

_Release

82.1.1.6.3 TRIO.SOAPHeaders Property

Provides access to the headers of SOAP messages that the remote interface object sends or receives.

Pascal

```
property SOAPHeaders: TSOAPHeaders;
```

C++

```
__property TSOAPHeaders SOAPHeaders;
```

Description

SOAPHeaders is a helper object that lets TRIO (see page 7408) support the ISOAPHeaders (see page 4464) interface. It has methods to add headers to an outgoing request message and to retrieve the headers from the resulting response message. You can use the SOAPHeaders property directly, or access it through an ISOAPHeaders (see page 4464) interface, which is

obtained from the invokable interface using the as operator (Delphi) or the Supports method (C++).

Note: Although TRIO (see page 7408) uses the SOAPHeaders property to implement an ISOAPHeaders (see page 4464) interface, you can't obtain an ISOAPHeaders (see page 4464) interface directly from TRIO (see page 7408). Rather, you must obtain this interface from an invokable interface.

See Also

ISOAPHeaders (see page 4464)

82.1.1.6.4 TRIO.WebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property WebNode: IWebNode;
```

C++

```
__property IWebNode WebNode;
```

Description

TRIO (see page 7408) uses WebNode internally to execute method calls after they have been marshaled by Converter (see page 7415). That is, when the TRIO (see page 7408) object has been typecast to an invokable interface, it implements each method call by using Converter (see page 7415) to marshal the call into an encoding protocol, and then uses WebNode to execute the encoded call.

Each TRIO (see page 7408) descendant uses its own protocol-specific helper object as the value of the WebNode property.

See Also

Converter (see page 7415)

THTTPReqResp

OnAfterExecute (see page 7417)

OnBeforeExecute (see page 7418)

82.1.1.7 TRIO Events

The events of the TRIO class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7417)	Occurs when TRIO (see page 7408) receives the results from executing a method of the invokable interface.
	OnBeforeExecute (see page 7418)	Occurs before TRIO (see page 7408) sends an encoded method call for execution.
	OnGetAttachment (see page 7419)	Occurs when TRIO (see page 7408) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 7419)	Occurs when TRIO (see page 7408) sends each attachment that is the value of an argument of the method call on the invokable interface.

82.1.1.7.1 TRIO.OnAfterExecute Event

Occurs when TRIO (see page 7408) receives the results from executing a method of the invokable interface.

Pascal

```
property OnAfterExecute: TAfterExecuteEvent;
```

C++

```
__property TAfterExecuteEvent OnAfterExecute;
```

Description

Write an OnAfterExecute event handler to examine or alter the encoded results from executing a method call. OnAfterExecute occurs after TRIO (see page 7408) calls the Execute method of the interface that the WebNode (see page 7417) property specifies, but before the interface that the Converter (see page 7415) property specifies unmarshals those results.

You can use the SOAPResponse stream of the event handler to alter the encoded results from the server before they are unmarshaled. This lets you compensate for known idiosyncrasies of a server or of the Converter (see page 7415) that is the value of the Converter (see page 7415) property. For example, if the Web Service provider publishes interoperability information, you can use this event to adjust SOAP packets to compensate for differences that the Web Service provider reports.

See Also

OnBeforeExecute (see page 7418)

Execute

WebNode (see page 7417)

Converter (see page 7415)

OnGetAttachment (see page 7419)

82.1.1.7.2 TRIO.OnBeforeExecute Event

Occurs before TRIO (see page 7408) sends an encoded method call for execution.

Pascal

```
property OnBeforeExecute: TBeforeExecuteEvent;
```

C++

```
__property TBeforeExecuteEvent OnBeforeExecute;
```

Description

Write an OnBeforeExecute event handler to examine or alter an encoded method call before it is sent to the server. OnBeforeExecute occurs after the interface that the Converter (see page 7415) property specifies marshals a method call into a SOAP request packet, but before TRIO (see page 7408) calls the Execute method of the interface that the WebNode (see page 7417) property specifies.

You can use the SoapRequest parameter to read or alter the SOAP request before it is passed to the server. For example, if the Web Service provider publishes interoperability information, you can use this parameter to adjust SOAP packets to compensate for differences that the Web Service provider reports.

See Also

OnAfterExecute (see page 7417)

Execute

WebNode (see page 7417)

Converter (see page 7415)

OnSendAttachment (see page 7419)

82.1.1.7.3 TRIO.OnGetAttachment Event

Occurs when TRIO (see page 7408) reads each attachment included in the results from executing a method of the invokable interface.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

Write an OnGetAttachment event handler to examine the encoded attachments included in the results from executing a method call or to redirect them so that they are saved to a specific file. OnGetAttachment occurs once for each attachment received in the results from a call to the Execute method of the interface that the WebNode (see page 7417) property specifies. All OnGetAttachment events occur before the OnAfterExecute (see page 7417) event.

See Also

OnAfterExecute (see page 7417)

Execute

WebNode (see page 7417)

82.1.1.7.4 TRIO.OnSendAttachment Event

Occurs when TRIO (see page 7408) sends each attachment that is the value of an argument of the method call on the invokable interface.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

Write an OnSendAttachment event handler to examine or alter the attachments that are to be included in a request message that represents a method call. OnSendAttachment occurs once for each attachment sent with the request message to the Web Service provider. All OnSendAttachment events occur after the OnBeforeExecute (see page 7418) event.

See Also

OnBeforeExecute (see page 7418)

Execute

WebNode (see page 7417)

82.2 Interfaces

The following table lists interfaces in this documentation.

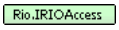
Interfaces

	Name	Description
	IRIOAccess (see page 7420)	IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

82.2.1 IRIOAccess Interface

IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

Class Hierarchy



Pascal

```
IRIOAccess = interface;
```

C++

```
__interface IRIOAccess;
```

File

Rio

Description

Remote interfaced objects (TRIO (see page 7408) and its descendants) implement IRIOAccess so that you can access the TRIO (see page 7408) instance given an invokable interface that it implements. This interface defines a single property, RIO (see page 7421), which returns the TRIO (see page 7408) instance that implements it.

Note: C++ method declarations that use IRIOAccess use the _di_IRIOAccess type instead. This type is a DelphiInterface wrapper around the IRIOAccess interface:

Note: typedef System::DelphiInterface< IRIOAccess > _di_IRIOAccess;

See Also

TRIO (see page 7408)

82.2.1.1 IRIOAccess Members

The following tables list the members exposed by IRIOAccess.

Public Methods

	Name	Description
	GetRIO (see page 7421)	Returns the value of the RIO (see page 7421) property.

Public Properties

	Name	Description
	RIO (see page 7421)	Provides access to the remote interfaced object that implements this interface.

82.2.1.2 IRIOAccess Methods

The methods of the IRIOAccess class are listed here.

Public Methods

	Name	Description
	GetRIO (see page 7421)	Returns the value of the RIO (see page 7421) property.

82.2.1.2.1 IRIOAccess.GetRIO Method

Returns the value of the RIO (see page 7421) property.

Pascal

```
function GetRIO: TRIO;
```

C++

```
__fastcall TRIO GetRIO();
```

Description

GetRIO is the read implementation for the RIO (see page 7421) property.

See Also

RIO (see page 7421)

82.2.1.3 IRIOAccess Properties

The properties of the IRIOAccess class are listed here.

Public Properties

	Name	Description
	RIO (see page 7421)	Provides access to the remote interfaced object that implements this interface.

82.2.1.3.1 IRIOAccess.RIO Property

Provides access to the remote interfaced object that implements this interface.

Pascal

```
property RIO: TRIO;
```

C++

```
__property TRIO RIO;
```

Description

Call GetRIO (see page 7421) to access the underlying remote interfaced object. This method is useful when the client of a Web Service uses the automatically-generated function to obtain an invokable interface rather than explicitly creating a TRIO (see page 7408) instance.

82.3 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterExecuteEvent (see page 7422)	This is type Rio.TAfterExecuteEvent.
TBeforeExecuteEvent (see page 7422)	This is type Rio.TBeforeExecuteEvent.

82.3.1 Rio.TAfterExecuteEvent Type

Pascal

```
TAfterExecuteEvent = procedure (const MethodName: string; SOAPResponse: TStream) of object;
```

C++

```
(const MethodName: string; SOAPResponse: TStream) ( TAfterExecuteEvent)();
```

File

Rio

Description

This is type Rio.TAfterExecuteEvent.

82.3.2 Rio.TBeforeExecuteEvent Type

Pascal

```
TBeforeExecuteEvent = procedure (const MethodName: string; SOAPRequest: TStream) of object;
```

C++

```
(const MethodName: string; SOAPRequest: TStream) ( TBeforeExecuteEvent)();
```

File

Rio

Description

This is type Rio.TBeforeExecuteEvent.

82.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
StubSize (see page 7423)	This is constant Rio.StubSize.

82.4.1 Rio.StubSize Constant

Pascal

```
StubSize = 19;
```

C++

```
StubSize = 19;
```

File

Rio

Description

This is constant Rio.StubSize.

83 RTLConsts Namespace

This is namespace RTLConsts.

Topics

Name	Description
Constants (see page 7431)	The following table lists constants in this documentation.

Constants

Name	Description
hNoContext (see page 7436)	This is constant RTLConsts.hNoContext.
hNoContextFound (see page 7436)	This is constant RTLConsts.hNoContextFound.
hNoFilterViewer (see page 7436)	This is constant RTLConsts.hNoFilterViewer.
hNoSystem (see page 7437)	This is constant RTLConsts.hNoSystem.
hNoTableOfContents (see page 7437)	This is constant RTLConsts.hNoTableOfContents.
hNothingFound (see page 7437)	This is constant RTLConsts.hNothingFound.
hNoTopics (see page 7438)	This is constant RTLConsts.hNoTopics.
SAcreFeetDescription (see page 7438)	This is constant RTLConsts.SAcreFeetDescription.
SAcreInchesDescription (see page 7438)	This is constant RTLConsts.SAcreInchesDescription.
SAcresDescription (see page 7439)	This is constant RTLConsts.SAcresDescription.
SAncestorNotFound (see page 7439)	This is constant RTLConsts.SAncestorNotFound.
SAngstromsDescription (see page 7439)	This is constant RTLConsts.SAngstromsDescription.
SAnsiUTF8Expected (see page 7440)	This is constant RTLConsts.SAnsiUTF8Expected.
SAreaDescription (see page 7440)	This is constant RTLConsts.SAreaDescription.
SAresDescription (see page 7440)	This is constant RTLConsts.SAresDescription.
sArgument_InvalidHighSurrogate (see page 7441)	This is constant RTLConsts.sArgument_InvalidHighSurrogate.
sArgument_InvalidLowSurrogate (see page 7441)	This is constant RTLConsts.sArgument_InvalidLowSurrogate.
sArgumentOutOfRange_Index (see page 7441)	This is constant RTLConsts.sArgumentOutOfRange_Index.
sArgumentOutOfRange_InvalidHighSurrogate (see page 7442)	This is constant RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate.
sArgumentOutOfRange_InvalidLowSurrogate (see page 7442)	This is constant RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate.
sArgumentOutOfRange_InvalidUTF32 (see page 7443)	This is constant RTLConsts.sArgumentOutOfRange_InvalidUTF32.
sArgumentOutOfRange_StringIndex (see page 7443)	This is constant RTLConsts.sArgumentOutOfRange_StringIndex.
SAssignError (see page 7443)	This is constant RTLConsts.SAssignError.
SAstronomicalUnitsDescription (see page 7444)	This is constant RTLConsts.SAstronomicalUnitsDescription.
sAsyncSocketError (see page 7444)	This is constant RTLConsts.sAsyncSocketError.
SBitsIndexError (see page 7444)	This is constant RTLConsts.SBitsIndexError.
SBucketListLocked (see page 7445)	This is constant RTLConsts.SBucketListLocked.

sCannotCallAcquireOnConditionVar (see page 7445)	This is constant RTLConsts.sCannotCallAcquireOnConditionVar.
sCannotCreateSocket (see page 7445)	This is constant RTLConsts.sCannotCreateSocket.
sCannotListenOnOpen (see page 7446)	This is constant RTLConsts.sCannotListenOnOpen.
sCantChangeWhileActive (see page 7446)	This is constant RTLConsts.sCantChangeWhileActive.
SCantWriteResourceStreamError (see page 7446)	This is constant RTLConsts.SCantWriteResourceStreamError.
SCelsiusDescription (see page 7447)	This is constant RTLConsts.SCelsiusDescription.
SCentaresDescription (see page 7447)	This is constant RTLConsts.SCentaresDescription.
SCentigramsDescription (see page 7447)	This is constant RTLConsts.SCentigramsDescription.
SCentiLitersDescription (see page 7448)	This is constant RTLConsts.SCentiLitersDescription.
SCentimetersDescription (see page 7448)	This is constant RTLConsts.SCentimetersDescription.
SCenturiesDescription (see page 7448)	This is constant RTLConsts.SCenturiesDescription.
SChainsDescription (see page 7449)	This is constant RTLConsts.SChainsDescription.
SCharExpected (see page 7449)	This is constant RTLConsts.SCharExpected.
SCheckSynchronizeError (see page 7449)	This is constant RTLConsts.SCheckSynchronizeError.
SClassNotFound (see page 7450)	This is constant RTLConsts.SClassNotFound.
SCmplxCouldNotParseImaginary (see page 7450)	This is constant RTLConsts.SCmplxCouldNotParseImaginary.
SCmplxCouldNotParsePlus (see page 7450)	This is constant RTLConsts.SCmplxCouldNotParsePlus.
SCmplxCouldNotParseReal (see page 7451)	This is constant RTLConsts.SCmplxCouldNotParseReal.
SCmplxCouldNotParseSymbol (see page 7451)	This is constant RTLConsts.SCmplxCouldNotParseSymbol.
SCmplxErrorSuffix (see page 7451)	This is constant RTLConsts.SCmplxErrorSuffix.
SCmplxUnexpectedChars (see page 7452)	This is constant RTLConsts.SCmplxUnexpectedChars.
SCmplxUnexpectedEOS (see page 7452)	This is constant RTLConsts.SCmplxUnexpectedEOS.
SComponentNameTooLong (see page 7452)	This is constant RTLConsts.SComponentNameTooLong.
SConvDuplicateFamily (see page 7453)	This is constant RTLConsts.SConvDuplicateFamily.
SConvDuplicateType (see page 7453)	This is constant RTLConsts.SConvDuplicateType.
SConvFactorZero (see page 7453)	This is constant RTLConsts.SConvFactorZero.
SConvIllegalFamily (see page 7454)	This is constant RTLConsts.SConvIllegalFamily.
SConvIllegalType (see page 7454)	This is constant RTLConsts.SConvIllegalType.
SConvIncompatibleTypes2 (see page 7454)	This is constant RTLConsts.SConvIncompatibleTypes2.
SConvIncompatibleTypes3 (see page 7455)	This is constant RTLConsts.SConvIncompatibleTypes3.
SConvIncompatibleTypes4 (see page 7455)	This is constant RTLConsts.SConvIncompatibleTypes4.
SConvStrParseError (see page 7455)	This is constant RTLConsts.SConvStrParseError.
SConvUnknownDescription (see page 7456)	This is constant RTLConsts.SConvUnknownDescription.
SConvUnknownDescriptionWithPrefix (see page 7456)	This is constant RTLConsts.SConvUnknownDescriptionWithPrefix.
SConvUnknownFamily (see page 7456)	This is constant RTLConsts.SConvUnknownFamily.
SConvUnknownType (see page 7457)	This is constant RTLConsts.SConvUnknownType.
SCordFeetDescription (see page 7457)	This is constant RTLConsts.SCordFeetDescription.
SCordsDescription (see page 7457)	This is constant RTLConsts.SCordsDescription.
SCubicCentimetersDescription (see page 7458)	This is constant RTLConsts.SCubicCentimetersDescription.
SCubicDecametersDescription (see page 7458)	This is constant RTLConsts.SCubicDecametersDescription.
SCubicDecimetersDescription (see page 7458)	This is constant RTLConsts.SCubicDecimetersDescription.
SCubicFeetDescription (see page 7459)	This is constant RTLConsts.SCubicFeetDescription.
SCubicHectometersDescription (see page 7459)	This is constant RTLConsts.SCubicHectometersDescription.
SCubicInchesDescription (see page 7459)	This is constant RTLConsts.SCubicInchesDescription.
SCubicKilometersDescription (see page 7460)	This is constant RTLConsts.SCubicKilometersDescription.
SCubicMetersDescription (see page 7460)	This is constant RTLConsts.SCubicMetersDescription.
SCubicMilesDescription (see page 7460)	This is constant RTLConsts.SCubicMilesDescription.
SCubicMillimetersDescription (see page 7461)	This is constant RTLConsts.SCubicMillimetersDescription.
SCubicYardsDescription (see page 7461)	This is constant RTLConsts.SCubicYardsDescription.
SCubitsDescription (see page 7461)	This is constant RTLConsts.SCubitsDescription.
SDateTimeDescription (see page 7462)	This is constant RTLConsts.SDateTimeDescription.
SDaysDescription (see page 7462)	This is constant RTLConsts.SDaysDescription.
SDecadesDescription (see page 7462)	This is constant RTLConsts.SDecadesDescription.
SDecagramsDescription (see page 7463)	This is constant RTLConsts.SDecagramsDescription.

SDecaLitersDescription (see page 7463)	This is constant RTLConsts.SDecaLitersDescription.
SDecametersDescription (see page 7463)	This is constant RTLConsts.SDecametersDescription.
SDecasteresDescription (see page 7464)	This is constant RTLConsts.SDecasteresDescription.
SDecigramsDescription (see page 7464)	This is constant RTLConsts.SDecigramsDescription.
SDeciLitersDescription (see page 7464)	This is constant RTLConsts.SDeciLitersDescription.
SDecimetersDescription (see page 7465)	This is constant RTLConsts.SDecimetersDescription.
SDecisteresDescription (see page 7465)	This is constant RTLConsts.SDecisteresDescription.
SDelimiterQuoteCharError (see page 7465)	This is constant RTLConsts.SDelimiterQuoteCharError.
SDistanceDescription (see page 7466)	This is constant RTLConsts.SDistanceDescription.
SDramsDescription (see page 7466)	This is constant RTLConsts.SDramsDescription.
SDryBucketsDescription (see page 7466)	This is constant RTLConsts.SDryBucketsDescription.
SDryBushelsDescription (see page 7467)	This is constant RTLConsts.SDryBushelsDescription.
SDryGallonsDescription (see page 7467)	This is constant RTLConsts.SDryGallonsDescription.
SDryPecksDescription (see page 7467)	This is constant RTLConsts.SDryPecksDescription.
SDryPintsDescription (see page 7468)	This is constant RTLConsts.SDryPintsDescription.
SDryQuartsDescription (see page 7468)	This is constant RTLConsts.SDryQuartsDescription.
SDuplicateClass (see page 7468)	This is constant RTLConsts.SDuplicateClass.
SDuplicateItem (see page 7469)	This is constant RTLConsts.SDuplicateItem.
SDuplicateName (see page 7469)	This is constant RTLConsts.SDuplicateName.
SDuplicateString (see page 7469)	This is constant RTLConsts.SDuplicateString.
SFahrenheitDescription (see page 7470)	This is constant RTLConsts.SFahrenheitDescription.
SFailedToCallConstructor (see page 7470)	This is constant RTLConsts.SFailedToCallConstructor.
SFathomsDescription (see page 7470)	This is constant RTLConsts.SFathomsDescription.
SFCreateError (see page 7471)	This is constant RTLConsts.SFCreateError.
SFCreateErrorEx (see page 7471)	This is constant RTLConsts.SFCreateErrorEx.
SFeetDescription (see page 7471)	This is constant RTLConsts.SFeetDescription.
SFixedColTooBig (see page 7472)	This is constant RTLConsts.SFixedColTooBig.
SFixedRowTooBig (see page 7472)	This is constant RTLConsts.SFixedRowTooBig.
SFluidCupsDescription (see page 7472)	This is constant RTLConsts.SFluidCupsDescription.
SFluidGallonsDescription (see page 7473)	This is constant RTLConsts.SFluidGallonsDescription.
SFluidGillsDescription (see page 7473)	This is constant RTLConsts.SFluidGillsDescription.
SFluidOuncesDescription (see page 7473)	This is constant RTLConsts.SFluidOuncesDescription.
SFluidPintsDescription (see page 7474)	This is constant RTLConsts.SFluidPintsDescription.
SFluidQuartsDescription (see page 7474)	This is constant RTLConsts.SFluidQuartsDescription.
SFluidTablespoonsDescription (see page 7474)	This is constant RTLConsts.SFluidTablespoonsDescription.
SFluidTeaspoonsDescription (see page 7475)	This is constant RTLConsts.SFluidTeaspoonsDescription.
SFOpenError (see page 7475)	This is constant RTLConsts.SFOpenError.
SFOpenErrorEx (see page 7475)	This is constant RTLConsts.SFOpenErrorEx.
SFortnightsDescription (see page 7476)	This is constant RTLConsts.SFortnightsDescription.
SFurlongsDescription (see page 7476)	This is constant RTLConsts.SFurlongsDescription.
SGigametersDescription (see page 7476)	This is constant RTLConsts.SGigametersDescription.
SGrainsDescription (see page 7477)	This is constant RTLConsts.SGrainsDescription.
SGramsDescription (see page 7477)	This is constant RTLConsts.SGramsDescription.
SGridTooLarge (see page 7477)	This is constant RTLConsts.SGridTooLarge.
SHandsDescription (see page 7478)	This is constant RTLConsts.SHandsDescription.
SHectaresDescription (see page 7478)	This is constant RTLConsts.SHectaresDescription.
SHectogramsDescription (see page 7478)	This is constant RTLConsts.SHectogramsDescription.
SHectoLitersDescription (see page 7479)	This is constant RTLConsts.SHectoLitersDescription.
SHectometersDescription (see page 7479)	This is constant RTLConsts.SHectometersDescription.
SHoursDescription (see page 7479)	This is constant RTLConsts.SHoursDescription.
SIdentifierExpected (see page 7480)	This is constant RTLConsts.SIdentifierExpected.
SInchesDescription (see page 7480)	This is constant RTLConsts.SInchesDescription.
SIndexOutOfRange (see page 7480)	This is constant RTLConsts.SIndexOutOfRange.
SIniFileWriteError (see page 7481)	This is constant RTLConsts.SIniFileWriteError.

SInvalidActionCreation (see page 7481)	This is constant RTLConsts.SInvalidActionCreation.
SInvalidActionEnumeration (see page 7481)	This is constant RTLConsts.SInvalidActionEnumeration.
SInvalidActionRegistration (see page 7482)	This is constant RTLConsts.SInvalidActionRegistration.
SInvalidActionUnregistration (see page 7482)	This is constant RTLConsts.SInvalidActionUnregistration.
SInvalidBinary (see page 7482)	This is constant RTLConsts.SInvalidBinary.
SInvalidDate (see page 7483)	This is constant RTLConsts.SInvalidDate.
SInvalidDateDay (see page 7483)	This is constant RTLConsts.SInvalidDateDay.
SInvalidDateMonthWeek (see page 7483)	This is constant RTLConsts.SInvalidDateMonthWeek.
SInvalidDateTime (see page 7484)	This is constant RTLConsts.SInvalidDateTime.
SInvalidDateWeek (see page 7484)	This is constant RTLConsts.SInvalidDateWeek.
SInvalidDayOfWeekInMonth (see page 7484)	This is constant RTLConsts.SInvalidDayOfWeekInMonth.
SInvalidFileName (see page 7485)	This is constant RTLConsts.SInvalidFileName.
SInvalidImage (see page 7485)	This is constant RTLConsts.SInvalidImage.
SInvalidInteger (see page 7485)	This is constant RTLConsts.SInvalidInteger.
SInvalidJulianDate (see page 7486)	This is constant RTLConsts.SInvalidJulianDate.
SInvalidMask (see page 7486)	This is constant RTLConsts.SInvalidMask.
SInvalidName (see page 7486)	This is constant RTLConsts.SInvalidName.
SInvalidProperty (see page 7487)	This is constant RTLConsts.SInvalidProperty.
SInvalidPropertyElement (see page 7487)	This is constant RTLConsts.SInvalidPropertyElement.
SInvalidPropertyPath (see page 7487)	This is constant RTLConsts.SInvalidPropertyPath.
SInvalidPropertyType (see page 7488)	This is constant RTLConsts.SInvalidPropertyType.
SInvalidPropertyValue (see page 7488)	This is constant RTLConsts.SInvalidPropertyValue.
SInvalidRegType (see page 7488)	This is constant RTLConsts.SInvalidRegType.
SInvalidString (see page 7489)	This is constant RTLConsts.SInvalidString.
SInvalidStringGridOp (see page 7489)	This is constant RTLConsts.SInvalidStringGridOp.
SInvalidTime (see page 7489)	This is constant RTLConsts.SInvalidTime.
SItemNotFound (see page 7490)	This is constant RTLConsts.SItemNotFound.
SJulianDateDescription (see page 7490)	This is constant RTLConsts.SJulianDateDescription.
SKelvinDescription (see page 7490)	This is constant RTLConsts.SKelvinDescription.
SKilogramsDescription (see page 7491)	This is constant RTLConsts.SKilogramsDescription.
SKiloLitersDescription (see page 7491)	This is constant RTLConsts.SKiloLitersDescription.
SKilometersDescription (see page 7491)	This is constant RTLConsts.SKilometersDescription.
SLightYearsDescription (see page 7492)	This is constant RTLConsts.SLightYearsDescription.
SLineTooLong (see page 7492)	This is constant RTLConsts.SLineTooLong.
SLinksDescription (see page 7492)	This is constant RTLConsts.SLinksDescription.
SListCapacityError (see page 7493)	This is constant RTLConsts.SListCapacityError.
SListCountError (see page 7493)	This is constant RTLConsts.SListCountError.
SListIndexError (see page 7493)	This is constant RTLConsts.SListIndexError.
SLitersDescription (see page 7494)	This is constant RTLConsts.SLitersDescription.
SLongTonsDescription (see page 7494)	This is constant RTLConsts.SLongTonsDescription.
SMaskEditErr (see page 7494)	This is constant RTLConsts.SMaskEditErr.
SMaskErr (see page 7495)	This is constant RTLConsts.SMaskErr.
SMassDescription (see page 7495)	This is constant RTLConsts.SMassDescription.
SMegametersDescription (see page 7495)	This is constant RTLConsts.SMegametersDescription.
SMemoryStreamError (see page 7496)	This is constant RTLConsts.SMemoryStreamError.
SMetersDescription (see page 7496)	This is constant RTLConsts.SMetersDescription.
SMetricTonsDescription (see page 7496)	This is constant RTLConsts.SMetricTonsDescription.
SMicrogramsDescription (see page 7497)	This is constant RTLConsts.SMicrogramsDescription.
SMicromicronsDescription (see page 7497)	This is constant RTLConsts.SMicromicronsDescription.
SMicronsDescription (see page 7497)	This is constant RTLConsts.SMicronsDescription.
SMilesDescription (see page 7498)	This is constant RTLConsts.SMilesDescription.
SMillenniaDescription (see page 7498)	This is constant RTLConsts.SMillenniaDescription.
SMilligramsDescription (see page 7498)	This is constant RTLConsts.SMilligramsDescription.
SMilliLitersDescription (see page 7499)	This is constant RTLConsts.SMilliLitersDescription.

SMillimetersDescription (see page 7499)	This is constant RTLConsts.SMillimetersDescription.
SMillimicronsDescription (see page 7499)	This is constant RTLConsts.SMillimicronsDescription.
SMilliSecondsDescription (see page 7500)	This is constant RTLConsts.SMilliSecondsDescription.
SMinutesDescription (see page 7500)	This is constant RTLConsts.SMinutesDescription.
SMissingDateTimeField (see page 7500)	This is constant RTLConsts.SMissingDateTimeField.
SModifiedJulianDateDescription (see page 7501)	This is constant RTLConsts.SModifiedJulianDateDescription.
SMonthsDescription (see page 7501)	This is constant RTLConsts.SMonthsDescription.
SNanogramsDescription (see page 7501)	This is constant RTLConsts.SNanogramsDescription.
SNauticalMilesDescription (see page 7502)	This is constant RTLConsts.SNauticalMilesDescription.
sNoAddress (see page 7502)	This is constant RTLConsts.sNoAddress.
SNoComSupport (see page 7502)	This is constant RTLConsts.SNoComSupport.
sNoConstruct (see page 7503)	This is constant RTLConsts.sNoConstruct.
SNotPrinting (see page 7503)	This is constant RTLConsts.SNotPrinting.
SNumberExpected (see page 7503)	This is constant RTLConsts.SNumberExpected.
SOuncesDescription (see page 7504)	This is constant RTLConsts.SOuncesDescription.
SPacesDescription (see page 7504)	This is constant RTLConsts.SPacesDescription.
SParsecsDescription (see page 7504)	This is constant RTLConsts.SParsecsDescription.
SParseError (see page 7505)	This is constant RTLConsts.SParseError.
SPicasDescription (see page 7505)	This is constant RTLConsts.SPicasDescription.
SPointsDescription (see page 7505)	This is constant RTLConsts.SPointsDescription.
SPoundsDescription (see page 7506)	This is constant RTLConsts.SPoundsDescription.
SPrinting (see page 7506)	This is constant RTLConsts.SPrinting.
SPropertyException (see page 7506)	This is constant RTLConsts.SPropertyException.
SRankineDescription (see page 7507)	This is constant RTLConsts.SRankineDescription.
SReadError (see page 7507)	This is constant RTLConsts.SReadError.
SReadOnlyProperty (see page 7507)	This is constant RTLConsts.SReadOnlyProperty.
SReaumurDescription (see page 7508)	This is constant RTLConsts.SReaumurDescription.
SRegCreateFailed (see page 7508)	This is constant RTLConsts.SRegCreateFailed.
SRegGetDataFailed (see page 7508)	This is constant RTLConsts.SRegGetDataFailed.
SRegisterError (see page 7509)	This is constant RTLConsts.SRegisterError.
SRegSetDataFailed (see page 7509)	This is constant RTLConsts.SRegSetDataFailed.
SResNotFound (see page 7509)	This is constant RTLConsts.SResNotFound.
SRodsDescription (see page 7510)	This is constant RTLConsts.SRodsDescription.
SSecondsDescription (see page 7510)	This is constant RTLConsts.SSecondsDescription.
SSeekNotImplemented (see page 7510)	This is constant RTLConsts.SSeekNotImplemented.
sSocketAlreadyOpen (see page 7511)	This is constant RTLConsts.sSocketAlreadyOpen.
sSocketIOError (see page 7511)	This is constant RTLConsts.sSocketIOError.
sSocketMustBeBlocking (see page 7511)	This is constant RTLConsts.sSocketMustBeBlocking.
sSocketRead (see page 7512)	This is constant RTLConsts.sSocketRead.
sSocketWrite (see page 7512)	This is constant RTLConsts.sSocketWrite.
SSortedListError (see page 7512)	This is constant RTLConsts.SSortedListError.
SSquareCentimetersDescription (see page 7513)	This is constant RTLConsts.SSquareCentimetersDescription.
SSquareDecametersDescription (see page 7513)	This is constant RTLConsts.SSquareDecametersDescription.
SSquareDecimetersDescription (see page 7513)	This is constant RTLConsts.SSquareDecimetersDescription.
SSquareFeetDescription (see page 7514)	This is constant RTLConsts.SSquareFeetDescription.
SSquareHectometersDescription (see page 7514)	This is constant RTLConsts.SSquareHectometersDescription.
SSquareInchesDescription (see page 7514)	This is constant RTLConsts.SSquareInchesDescription.
SSquareKilometersDescription (see page 7515)	This is constant RTLConsts.SSquareKilometersDescription.
SSquareMetersDescription (see page 7515)	This is constant RTLConsts.SSquareMetersDescription.
SSquareMilesDescription (see page 7515)	This is constant RTLConsts.SSquareMilesDescription.
SSquareMillimetersDescription (see page 7516)	This is constant RTLConsts.SSquareMillimetersDescription.
SSquareRodsDescription (see page 7516)	This is constant RTLConsts.SSquareRodsDescription.
SSquareYardsDescription (see page 7516)	This is constant RTLConsts.SSquareYardsDescription.
SSteresDescription (see page 7517)	This is constant RTLConsts.SSteresDescription.

SStonesDescription (see page 7517)	This is constant RTLConsts.SStonesDescription.
SStreamSetSize (see page 7517)	This is constant RTLConsts.SStreamSetSize.
SStringExpected (see page 7518)	This is constant RTLConsts.SStringExpected.
SSymbolExpected (see page 7518)	This is constant RTLConsts.SSymbolExpected.
STemperatureDescription (see page 7518)	This is constant RTLConsts.STemperatureDescription.
SThreadCreateError (see page 7519)	This is constant RTLConsts.SThreadCreateError.
SThreadError (see page 7519)	This is constant RTLConsts.SThreadError.
SThreadExternalTerminate (see page 7519)	This is constant RTLConsts.SThreadExternalTerminate.
SThreadExternalWait (see page 7520)	This is constant RTLConsts.SThreadExternalWait.
STimeDescription (see page 7520)	This is constant RTLConsts.STimeDescription.
STimeEncodeError (see page 7520)	This is constant RTLConsts.STimeEncodeError.
STonsDescription (see page 7521)	This is constant RTLConsts.STonsDescription.
STooManyDeleted (see page 7521)	This is constant RTLConsts.STooManyDeleted.
SUKBucketsDescription (see page 7521)	This is constant RTLConsts.SUKBucketsDescription.
SUKBushelsDescription (see page 7522)	This is constant RTLConsts.SUKBushelsDescription.
SUKGallonsDescription (see page 7522)	This is constant RTLConsts.SUKGallonsDescription.
SUKGillsDescription (see page 7522)	This is constant RTLConsts.SUKGillsDescription.
SUKOuncesDescription (see page 7523)	This is constant RTLConsts.SUKOuncesDescription.
SUKPecksDescription (see page 7523)	This is constant RTLConsts.SUKPecksDescription.
SUKPintsDescription (see page 7523)	This is constant RTLConsts.SUKPintsDescription.
SUKPottlesDescription (see page 7524)	This is constant RTLConsts.SUKPottlesDescription.
SUKQuartsDescription (see page 7524)	This is constant RTLConsts.SUKQuartsDescription.
SUnknownGroup (see page 7524)	This is constant RTLConsts.SUnknownGroup.
SUnknownProperty (see page 7525)	This is constant RTLConsts.SUnknownProperty.
SVolumeDescription (see page 7525)	This is constant RTLConsts.SVolumeDescription.
SWeeksDescription (see page 7525)	This is constant RTLConsts.SWeeksDescription.
sWindowsSocketError (see page 7526)	This is constant RTLConsts.sWindowsSocketError.
SWriteError (see page 7526)	This is constant RTLConsts.SWriteError.
SYardsDescription (see page 7526)	This is constant RTLConsts.SYardsDescription.
SYearsDescription (see page 7527)	This is constant RTLConsts.SYearsDescription.

83.1 Constants

The following table lists constants in this documentation.

Constants

Name	Description
hNoContext (see page 7436)	This is constant RTLConsts.hNoContext.
hNoContextFound (see page 7436)	This is constant RTLConsts.hNoContextFound.
hNoFilterViewer (see page 7436)	This is constant RTLConsts.hNoFilterViewer.
hNoSystem (see page 7437)	This is constant RTLConsts.hNoSystem.
hNoTableOfContents (see page 7437)	This is constant RTLConsts.hNoTableOfContents.
hNothingFound (see page 7437)	This is constant RTLConsts.hNothingFound.
hNoTopics (see page 7438)	This is constant RTLConsts.hNoTopics.
SAcreFeetDescription (see page 7438)	This is constant RTLConsts.SAcreFeetDescription.
SAcreInchesDescription (see page 7438)	This is constant RTLConsts.SAcreInchesDescription.
SAcresDescription (see page 7439)	This is constant RTLConsts.SAcresDescription.
SAncestorNotFound (see page 7439)	This is constant RTLConsts.SAncestorNotFound.
SAngstromsDescription (see page 7439)	This is constant RTLConsts.SAngstromsDescription.
SAnsiUTF8Expected (see page 7440)	This is constant RTLConsts.SAnsiUTF8Expected.
SAreaDescription (see page 7440)	This is constant RTLConsts.SAreaDescription.
SAresDescription (see page 7440)	This is constant RTLConsts.SAresDescription.
sArgument_InvalidHighSurrogate (see page 7441)	This is constant RTLConsts.sArgument_InvalidHighSurrogate.
sArgument_InvalidLowSurrogate (see page 7441)	This is constant RTLConsts.sArgument_InvalidLowSurrogate.
sArgumentOutOfRange_Index (see page 7441)	This is constant RTLConsts.sArgumentOutOfRange_Index.
sArgumentOutOfRange_InvalidHighSurrogate (see page 7442)	This is constant RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate.
sArgumentOutOfRange_InvalidLowSurrogate (see page 7442)	This is constant RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate.
sArgumentOutOfRange_InvalidUTF32 (see page 7443)	This is constant RTLConsts.sArgumentOutOfRange_InvalidUTF32.
sArgumentOutOfRange_StringIndex (see page 7443)	This is constant RTLConsts.sArgumentOutOfRange_StringIndex.
SAssignError (see page 7443)	This is constant RTLConsts.SAssignError.
SAstronomicalUnitsDescription (see page 7444)	This is constant RTLConsts.SAstronomicalUnitsDescription.
sAsyncSocketError (see page 7444)	This is constant RTLConsts.sAsyncSocketError.
SBitsIndexError (see page 7444)	This is constant RTLConsts.SBitsIndexError.
SBucketListLocked (see page 7445)	This is constant RTLConsts.SBucketListLocked.
sCannotCallAcquireOnConditionVar (see page 7445)	This is constant RTLConsts.sCannotCallAcquireOnConditionVar.
sCannotCreateSocket (see page 7445)	This is constant RTLConsts.sCannotCreateSocket.
sCannotListenOnOpen (see page 7446)	This is constant RTLConsts.sCannotListenOnOpen.
sCantChangeWhileActive (see page 7446)	This is constant RTLConsts.sCantChangeWhileActive.
SCantWriteResourceStreamError (see page 7446)	This is constant RTLConsts.SCantWriteResourceStreamError.
SCelsiusDescription (see page 7447)	This is constant RTLConsts.SCelsiusDescription.
SCentaresDescription (see page 7447)	This is constant RTLConsts.SCentaresDescription.
SCentigramsDescription (see page 7447)	This is constant RTLConsts.SCentigramsDescription.
SCentiLitersDescription (see page 7448)	This is constant RTLConsts.SCentiLitersDescription.
SCentimetersDescription (see page 7448)	This is constant RTLConsts.SCentimetersDescription.
SCenturiesDescription (see page 7448)	This is constant RTLConsts.SCenturiesDescription.
SChainsDescription (see page 7449)	This is constant RTLConsts.SChainsDescription.
SCharExpected (see page 7449)	This is constant RTLConsts.SCharExpected.
SCheckSynchronizeError (see page 7449)	This is constant RTLConsts.SCheckSynchronizeError.
SClassNotFound (see page 7450)	This is constant RTLConsts.SClassNotFound.
SCmplxCouldNotParseImaginary (see page 7450)	This is constant RTLConsts.SCmplxCouldNotParseImaginary.
SCmplxCouldNotParsePlus (see page 7450)	This is constant RTLConsts.SCmplxCouldNotParsePlus.

SCmplxCouldNotParseReal (see page 7451)	This is constant RTLConsts.SCmplxCouldNotParseReal.
SCmplxCouldNotParseSymbol (see page 7451)	This is constant RTLConsts.SCmplxCouldNotParseSymbol.
SCmplxErrorSuffix (see page 7451)	This is constant RTLConsts.SCmplxErrorSuffix.
SCmplxUnexpectedChars (see page 7452)	This is constant RTLConsts.SCmplxUnexpectedChars.
SCmplxUnexpectedEOS (see page 7452)	This is constant RTLConsts.SCmplxUnexpectedEOS.
SComponentNameTooLong (see page 7452)	This is constant RTLConsts.SComponentNameTooLong.
SConvDuplicateFamily (see page 7453)	This is constant RTLConsts.SConvDuplicateFamily.
SConvDuplicateType (see page 7453)	This is constant RTLConsts.SConvDuplicateType.
SConvFactorZero (see page 7453)	This is constant RTLConsts.SConvFactorZero.
SConvIllegalFamily (see page 7454)	This is constant RTLConsts.SConvIllegalFamily.
SConvIllegalType (see page 7454)	This is constant RTLConsts.SConvIllegalType.
SConvIncompatibleTypes2 (see page 7454)	This is constant RTLConsts.SConvIncompatibleTypes2.
SConvIncompatibleTypes3 (see page 7455)	This is constant RTLConsts.SConvIncompatibleTypes3.
SConvIncompatibleTypes4 (see page 7455)	This is constant RTLConsts.SConvIncompatibleTypes4.
SConvStrParseError (see page 7455)	This is constant RTLConsts.SConvStrParseError.
SConvUnknownDescription (see page 7456)	This is constant RTLConsts.SConvUnknownDescription.
SConvUnknownDescriptionWithPrefix (see page 7456)	This is constant RTLConsts.SConvUnknownDescriptionWithPrefix.
SConvUnknownFamily (see page 7456)	This is constant RTLConsts.SConvUnknownFamily.
SConvUnknownType (see page 7457)	This is constant RTLConsts.SConvUnknownType.
SCordFeetDescription (see page 7457)	This is constant RTLConsts.SCordFeetDescription.
SCordsDescription (see page 7457)	This is constant RTLConsts.SCordsDescription.
SCubicCentimetersDescription (see page 7458)	This is constant RTLConsts.SCubicCentimetersDescription.
SCubicDecametersDescription (see page 7458)	This is constant RTLConsts.SCubicDecametersDescription.
SCubicDecimetersDescription (see page 7458)	This is constant RTLConsts.SCubicDecimetersDescription.
SCubicFeetDescription (see page 7459)	This is constant RTLConsts.SCubicFeetDescription.
SCubicHectometersDescription (see page 7459)	This is constant RTLConsts.SCubicHectometersDescription.
SCubicInchesDescription (see page 7459)	This is constant RTLConsts.SCubicInchesDescription.
SCubicKilometersDescription (see page 7460)	This is constant RTLConsts.SCubicKilometersDescription.
SCubicMetersDescription (see page 7460)	This is constant RTLConsts.SCubicMetersDescription.
SCubicMilesDescription (see page 7460)	This is constant RTLConsts.SCubicMilesDescription.
SCubicMillimetersDescription (see page 7461)	This is constant RTLConsts.SCubicMillimetersDescription.
SCubicYardsDescription (see page 7461)	This is constant RTLConsts.SCubicYardsDescription.
SCubitsDescription (see page 7461)	This is constant RTLConsts.SCubitsDescription.
SDateTimeDescription (see page 7462)	This is constant RTLConsts.SDateTimeDescription.
SDaysDescription (see page 7462)	This is constant RTLConsts.SDaysDescription.
SDecadesDescription (see page 7462)	This is constant RTLConsts.SDecadesDescription.
SDecagramsDescription (see page 7463)	This is constant RTLConsts.SDecagramsDescription.
SDecaLitersDescription (see page 7463)	This is constant RTLConsts.SDecaLitersDescription.
SDecametersDescription (see page 7463)	This is constant RTLConsts.SDecametersDescription.
SDecasteresDescription (see page 7464)	This is constant RTLConsts.SDecasteresDescription.
SDecigramsDescription (see page 7464)	This is constant RTLConsts.SDecigramsDescription.
SDeciLitersDescription (see page 7464)	This is constant RTLConsts.SDeciLitersDescription.
SDecimetersDescription (see page 7465)	This is constant RTLConsts.SDecimetersDescription.
SDecisteresDescription (see page 7465)	This is constant RTLConsts.SDecisteresDescription.
SDelimiterQuoteCharError (see page 7465)	This is constant RTLConsts.SDelimiterQuoteCharError.
SDistanceDescription (see page 7466)	This is constant RTLConsts.SDistanceDescription.
SDramsDescription (see page 7466)	This is constant RTLConsts.SDramsDescription.
SDryBucketsDescription (see page 7466)	This is constant RTLConsts.SDryBucketsDescription.
SDryBushelsDescription (see page 7467)	This is constant RTLConsts.SDryBushelsDescription.
SDryGallonsDescription (see page 7467)	This is constant RTLConsts.SDryGallonsDescription.
SDryPecksDescription (see page 7467)	This is constant RTLConsts.SDryPecksDescription.
SDryPintsDescription (see page 7468)	This is constant RTLConsts.SDryPintsDescription.
SDryQuartsDescription (see page 7468)	This is constant RTLConsts.SDryQuartsDescription.
SDuplicateClass (see page 7468)	This is constant RTLConsts.SDuplicateClass.

SDuplicateItem (see page 7469)	This is constant RTLConsts.SDuplicateItem.
SDuplicateName (see page 7469)	This is constant RTLConsts.SDuplicateName.
SDuplicateString (see page 7469)	This is constant RTLConsts.SDuplicateString.
SFahrenheitDescription (see page 7470)	This is constant RTLConsts.SFahrenheitDescription.
SFailedToCallConstructor (see page 7470)	This is constant RTLConsts.SFailedToCallConstructor.
SFathomsDescription (see page 7470)	This is constant RTLConsts.SFathomsDescription.
SFCreateError (see page 7471)	This is constant RTLConsts.SFCreateError.
SFCreateErrorEx (see page 7471)	This is constant RTLConsts.SFCreateErrorEx.
SFeetDescription (see page 7471)	This is constant RTLConsts.SFeetDescription.
SFixedColTooBig (see page 7472)	This is constant RTLConsts.SFixedColTooBig.
SFixedRowTooBig (see page 7472)	This is constant RTLConsts.SFixedRowTooBig.
SFluidCupsDescription (see page 7472)	This is constant RTLConsts.SFluidCupsDescription.
SFluidGallonsDescription (see page 7473)	This is constant RTLConsts.SFluidGallonsDescription.
SFluidGillsDescription (see page 7473)	This is constant RTLConsts.SFluidGillsDescription.
SFluidOuncesDescription (see page 7473)	This is constant RTLConsts.SFluidOuncesDescription.
SFluidPintsDescription (see page 7474)	This is constant RTLConsts.SFluidPintsDescription.
SFluidQuartsDescription (see page 7474)	This is constant RTLConsts.SFluidQuartsDescription.
SFluidTablespoonsDescription (see page 7474)	This is constant RTLConsts.SFluidTablespoonsDescription.
SFluidTeaspoonsDescription (see page 7475)	This is constant RTLConsts.SFluidTeaspoonsDescription.
SFOpenError (see page 7475)	This is constant RTLConsts.SFOpenError.
SFOpenErrorEx (see page 7475)	This is constant RTLConsts.SFOpenErrorEx.
SFortnightsDescription (see page 7476)	This is constant RTLConsts.SFortnightsDescription.
SFurlongsDescription (see page 7476)	This is constant RTLConsts.SFurlongsDescription.
SGigametersDescription (see page 7476)	This is constant RTLConsts.SGigametersDescription.
SGrainsDescription (see page 7477)	This is constant RTLConsts.SGrainsDescription.
SGramsDescription (see page 7477)	This is constant RTLConsts.SGramsDescription.
SGridTooLarge (see page 7477)	This is constant RTLConsts.SGridTooLarge.
SHandsDescription (see page 7478)	This is constant RTLConsts.SHandsDescription.
SHectaresDescription (see page 7478)	This is constant RTLConsts.SHectaresDescription.
SHectogramsDescription (see page 7478)	This is constant RTLConsts.SHectogramsDescription.
SHectoLitersDescription (see page 7479)	This is constant RTLConsts.SHectoLitersDescription.
SHectometersDescription (see page 7479)	This is constant RTLConsts.SHectometersDescription.
SHoursDescription (see page 7479)	This is constant RTLConsts.SHoursDescription.
SIdentifierExpected (see page 7480)	This is constant RTLConsts.SIdentifierExpected.
SInchesDescription (see page 7480)	This is constant RTLConsts.SInchesDescription.
SIndexOutOfRange (see page 7480)	This is constant RTLConsts.SIndexOutOfRange.
SIniFileWriteError (see page 7481)	This is constant RTLConsts.SIniFileWriteError.
SInvalidActionCreation (see page 7481)	This is constant RTLConsts.SInvalidActionCreation.
SInvalidActionEnumeration (see page 7481)	This is constant RTLConsts.SInvalidActionEnumeration.
SInvalidActionRegistration (see page 7482)	This is constant RTLConsts.SInvalidActionRegistration.
SInvalidActionUnregistration (see page 7482)	This is constant RTLConsts.SInvalidActionUnregistration.
SInvalidBinary (see page 7482)	This is constant RTLConsts.SInvalidBinary.
SInvalidDate (see page 7483)	This is constant RTLConsts.SInvalidDate.
SInvalidDateDay (see page 7483)	This is constant RTLConsts.SInvalidDateDay.
SInvalidDateMonthWeek (see page 7483)	This is constant RTLConsts.SInvalidDateMonthWeek.
SInvalidDateTime (see page 7484)	This is constant RTLConsts.SInvalidDateTime.
SInvalidDateWeek (see page 7484)	This is constant RTLConsts.SInvalidDateWeek.
SInvalidDayOfWeekInMonth (see page 7484)	This is constant RTLConsts.SInvalidDayOfWeekInMonth.
SInvalidFileName (see page 7485)	This is constant RTLConsts.SInvalidFileName.
SInvalidImage (see page 7485)	This is constant RTLConsts.SInvalidImage.
SInvalidInteger (see page 7485)	This is constant RTLConsts.SInvalidInteger.
SInvalidJulianDate (see page 7486)	This is constant RTLConsts.SInvalidJulianDate.
SInvalidMask (see page 7486)	This is constant RTLConsts.SInvalidMask.
SInvalidName (see page 7486)	This is constant RTLConsts.SInvalidName.

SInvalidProperty (see page 7487)	This is constant RTLConsts.SInvalidProperty.
SInvalidPropertyElement (see page 7487)	This is constant RTLConsts.SInvalidPropertyElement.
SInvalidPropertyPath (see page 7487)	This is constant RTLConsts.SInvalidPropertyPath.
SInvalidPropertyType (see page 7488)	This is constant RTLConsts.SInvalidPropertyType.
SInvalidPropertyValue (see page 7488)	This is constant RTLConsts.SInvalidPropertyValue.
SInvalidRegType (see page 7488)	This is constant RTLConsts.SInvalidRegType.
SInvalidString (see page 7489)	This is constant RTLConsts.SInvalidString.
SInvalidStringGridOp (see page 7489)	This is constant RTLConsts.SInvalidStringGridOp.
SInvalidTime (see page 7489)	This is constant RTLConsts.SInvalidTime.
SItemNotFound (see page 7490)	This is constant RTLConsts.SItemNotFound.
SJulianDateDescription (see page 7490)	This is constant RTLConsts.SJulianDateDescription.
SKelvinDescription (see page 7490)	This is constant RTLConsts.SKelvinDescription.
SKilogramsDescription (see page 7491)	This is constant RTLConsts.SKilogramsDescription.
SKiloLitersDescription (see page 7491)	This is constant RTLConsts.SKiloLitersDescription.
SKilometersDescription (see page 7491)	This is constant RTLConsts.SKilometersDescription.
SLightYearsDescription (see page 7492)	This is constant RTLConsts.SLightYearsDescription.
SLineTooLong (see page 7492)	This is constant RTLConsts.SLineTooLong.
SLinksDescription (see page 7492)	This is constant RTLConsts.SLinksDescription.
SListCapacityError (see page 7493)	This is constant RTLConsts.SListCapacityError.
SListCountError (see page 7493)	This is constant RTLConsts.SListCountError.
SListIndexError (see page 7493)	This is constant RTLConsts.SListIndexError.
SLitersDescription (see page 7494)	This is constant RTLConsts.SLitersDescription.
SLongTonsDescription (see page 7494)	This is constant RTLConsts.SLongTonsDescription.
SMaskEditErr (see page 7494)	This is constant RTLConsts.SMaskEditErr.
SMaskErr (see page 7495)	This is constant RTLConsts.SMaskErr.
SMassDescription (see page 7495)	This is constant RTLConsts.SMassDescription.
SMegametersDescription (see page 7495)	This is constant RTLConsts.SMegametersDescription.
SMemoryStreamError (see page 7496)	This is constant RTLConsts.SMemoryStreamError.
SMetersDescription (see page 7496)	This is constant RTLConsts.SMetersDescription.
SMetricTonsDescription (see page 7496)	This is constant RTLConsts.SMetricTonsDescription.
SMicrogramsDescription (see page 7497)	This is constant RTLConsts.SMicrogramsDescription.
SMicromicronsDescription (see page 7497)	This is constant RTLConsts.SMicromicronsDescription.
SMicronsDescription (see page 7497)	This is constant RTLConsts.SMicronsDescription.
SMilesDescription (see page 7498)	This is constant RTLConsts.SMilesDescription.
SMillenniaDescription (see page 7498)	This is constant RTLConsts.SMillenniaDescription.
SMilligramsDescription (see page 7498)	This is constant RTLConsts.SMilligramsDescription.
SMilliLitersDescription (see page 7499)	This is constant RTLConsts.SMilliLitersDescription.
SMillimetersDescription (see page 7499)	This is constant RTLConsts.SMillimetersDescription.
SMillimicronsDescription (see page 7499)	This is constant RTLConsts.SMillimicronsDescription.
SMilliSecondsDescription (see page 7500)	This is constant RTLConsts.SMilliSecondsDescription.
SMinutesDescription (see page 7500)	This is constant RTLConsts.SMinutesDescription.
SMissingDateTimeField (see page 7500)	This is constant RTLConsts.SMissingDateTimeField.
SModifiedJulianDateDescription (see page 7501)	This is constant RTLConsts.SModifiedJulianDateDescription.
SMonthsDescription (see page 7501)	This is constant RTLConsts.SMonthsDescription.
SNanogramsDescription (see page 7501)	This is constant RTLConsts.SNanogramsDescription.
SNauticalMilesDescription (see page 7502)	This is constant RTLConsts.SNauticalMilesDescription.
sNoAddress (see page 7502)	This is constant RTLConsts.sNoAddress.
SNoComSupport (see page 7502)	This is constant RTLConsts.SNoComSupport.
sNoConstruct (see page 7503)	This is constant RTLConsts.sNoConstruct.
SNotPrinting (see page 7503)	This is constant RTLConsts.SNotPrinting.
SNumberExpected (see page 7503)	This is constant RTLConsts.SNumberExpected.
SOuncesDescription (see page 7504)	This is constant RTLConsts.SOuncesDescription.
SPacesDescription (see page 7504)	This is constant RTLConsts.SPacesDescription.
SParsecsDescription (see page 7504)	This is constant RTLConsts.SParsecsDescription.

SParseError (see page 7505)	This is constant RTLConsts.SParseError.
SPicasDescription (see page 7505)	This is constant RTLConsts.SPicasDescription.
SPointsDescription (see page 7505)	This is constant RTLConsts.SPointsDescription.
SPoundsDescription (see page 7506)	This is constant RTLConsts.SPoundsDescription.
SPrinting (see page 7506)	This is constant RTLConsts.SPrinting.
SPropertyException (see page 7506)	This is constant RTLConsts.SPropertyException.
SRankineDescription (see page 7507)	This is constant RTLConsts.SRankineDescription.
SReadError (see page 7507)	This is constant RTLConsts.SReadError.
SReadOnlyProperty (see page 7507)	This is constant RTLConsts.SReadOnlyProperty.
SReaumurDescription (see page 7508)	This is constant RTLConsts.SReaumurDescription.
SRegCreateFailed (see page 7508)	This is constant RTLConsts.SRegCreateFailed.
SRegGetDataFailed (see page 7508)	This is constant RTLConsts.SRegGetDataFailed.
SRegisterError (see page 7509)	This is constant RTLConsts.SRegisterError.
SRegSetDataFailed (see page 7509)	This is constant RTLConsts.SRegSetDataFailed.
SResNotFound (see page 7509)	This is constant RTLConsts.SResNotFound.
SRodsDescription (see page 7510)	This is constant RTLConsts.SRodsDescription.
SSecondsDescription (see page 7510)	This is constant RTLConsts.SSecondsDescription.
SSeekNotImplemented (see page 7510)	This is constant RTLConsts.SSeekNotImplemented.
sSocketAlreadyOpen (see page 7511)	This is constant RTLConsts.sSocketAlreadyOpen.
sSocketIOError (see page 7511)	This is constant RTLConsts.sSocketIOError.
sSocketMustBeBlocking (see page 7511)	This is constant RTLConsts.sSocketMustBeBlocking.
sSocketRead (see page 7512)	This is constant RTLConsts.sSocketRead.
sSocketWrite (see page 7512)	This is constant RTLConsts.sSocketWrite.
SSortedListError (see page 7512)	This is constant RTLConsts.SSortedListError.
SSquareCentimetersDescription (see page 7513)	This is constant RTLConsts.SSquareCentimetersDescription.
SSquareDecametersDescription (see page 7513)	This is constant RTLConsts.SSquareDecametersDescription.
SSquareDecimetersDescription (see page 7513)	This is constant RTLConsts.SSquareDecimetersDescription.
SSquareFeetDescription (see page 7514)	This is constant RTLConsts.SSquareFeetDescription.
SSquareHectometersDescription (see page 7514)	This is constant RTLConsts.SSquareHectometersDescription.
SSquareInchesDescription (see page 7514)	This is constant RTLConsts.SSquareInchesDescription.
SSquareKilometersDescription (see page 7515)	This is constant RTLConsts.SSquareKilometersDescription.
SSquareMetersDescription (see page 7515)	This is constant RTLConsts.SSquareMetersDescription.
SSquareMilesDescription (see page 7515)	This is constant RTLConsts.SSquareMilesDescription.
SSquareMillimetersDescription (see page 7516)	This is constant RTLConsts.SSquareMillimetersDescription.
SSquareRodsDescription (see page 7516)	This is constant RTLConsts.SSquareRodsDescription.
SSquareYardsDescription (see page 7516)	This is constant RTLConsts.SSquareYardsDescription.
SSteresDescription (see page 7517)	This is constant RTLConsts.SSteresDescription.
SStonesDescription (see page 7517)	This is constant RTLConsts.SStonesDescription.
SStreamSetSize (see page 7517)	This is constant RTLConsts.SStreamSetSize.
SStringExpected (see page 7518)	This is constant RTLConsts.SStringExpected.
SSymbolExpected (see page 7518)	This is constant RTLConsts.SSymbolExpected.
STemperatureDescription (see page 7518)	This is constant RTLConsts.STemperatureDescription.
SThreadCreateError (see page 7519)	This is constant RTLConsts.SThreadCreateError.
SThreadError (see page 7519)	This is constant RTLConsts.SThreadError.
SThreadExternalTerminate (see page 7519)	This is constant RTLConsts.SThreadExternalTerminate.
SThreadExternalWait (see page 7520)	This is constant RTLConsts.SThreadExternalWait.
STimeDescription (see page 7520)	This is constant RTLConsts.STimeDescription.
STimeEncodeError (see page 7520)	This is constant RTLConsts.STimeEncodeError.
STonsDescription (see page 7521)	This is constant RTLConsts.STonsDescription.
STooManyDeleted (see page 7521)	This is constant RTLConsts.STooManyDeleted.
SUKBucketsDescription (see page 7521)	This is constant RTLConsts.SUKBucketsDescription.
SUKBushelsDescription (see page 7522)	This is constant RTLConsts.SUKBushelsDescription.
SUKGallonsDescription (see page 7522)	This is constant RTLConsts.SUKGallonsDescription.
SUKGillsDescription (see page 7522)	This is constant RTLConsts.SUKGillsDescription.

SUKOuncesDescription (see page 7523)	This is constant RTLConsts.SUKOuncesDescription.
SUKPecksDescription (see page 7523)	This is constant RTLConsts.SUKPecksDescription.
SUKPintsDescription (see page 7523)	This is constant RTLConsts.SUKPintsDescription.
SUKPottlesDescription (see page 7524)	This is constant RTLConsts.SUKPottlesDescription.
SUKQuartsDescription (see page 7524)	This is constant RTLConsts.SUKQuartsDescription.
SUnknownGroup (see page 7524)	This is constant RTLConsts.SUnknownGroup.
SUnknownProperty (see page 7525)	This is constant RTLConsts.SUnknownProperty.
SVolumeDescription (see page 7525)	This is constant RTLConsts.SVolumeDescription.
SWeeksDescription (see page 7525)	This is constant RTLConsts.SWeeksDescription.
sWindowsSocketError (see page 7526)	This is constant RTLConsts.sWindowsSocketError.
SWriteError (see page 7526)	This is constant RTLConsts.SWriteError.
SYardsDescription (see page 7526)	This is constant RTLConsts.SYardsDescription.
SYearsDescription (see page 7527)	This is constant RTLConsts.SYearsDescription.

83.1.1 RTLConsts.hNoContext Constant

Pascal

```
hNoContext = 'No context-sensitive help installed.';
```

C++

```
hNoContext = 'No context-sensitive help installed.';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoContext.

83.1.2 RTLConsts.hNoContextFound Constant

Pascal

```
hNoContextFound = 'No help found for context %d.';
```

C++

```
hNoContextFound = 'No help found for context %d.';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoContextFound.

83.1.3 RTLConsts.hNoFilterViewer Constant

Pascal

```
hNoFilterViewer = 'No help viewer that supports filters';
```

C++

```
hNoFilterViewer = 'No help viewer that supports filters';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoFilterViewer.

83.1.4 RTLConsts.hNoSystem Constant

Pascal

```
hNoSystem = 'No Help Manager installed.';
```

C++

```
hNoSystem = 'No Help Manager installed.';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoSystem.

83.1.5 RTLConsts.hNoTableOfContents Constant

Pascal

```
hNoTableOfContents = 'No Table of Contents found.';
```

C++

```
hNoTableOfContents = 'No Table of Contents found.';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoTableOfContents.

83.1.6 RTLConsts.hNothingFound Constant

Pascal

```
hNothingFound = 'No help found for "%s"';
```

C++

```
hNothingFound = 'No help found for "%s"';
```

File

RTLConsts

Description

This is constant RTLConsts.hNothingFound.

83.1.7 RTLConsts.hNoTopics Constant

Pascal

```
hNoTopics = 'No topic-based help installed.';
```

C++

```
hNoTopics = 'No topic-based help installed.';
```

File

RTLConsts

Description

This is constant RTLConsts.hNoTopics.

83.1.8 RTLConsts.SAcreFeetDescription Constant

Pascal

```
SAcreFeetDescription = 'AcreFeet';
```

C++

```
SAcreFeetDescription = 'AcreFeet';
```

File

RTLConsts

Description

This is constant RTLConsts.SAcreFeetDescription.

83.1.9 RTLConsts.SAcreInchesDescription Constant

Pascal

```
SAcreInchesDescription = 'AcreInches';
```

C++

```
SAcreInchesDescription = 'AcreInches';
```

File

RTLConsts

Description

This is constant RTLConsts.SAcreInchesDescription.

83.1.10 RTLConsts.SAcresDescription Constant

Pascal

```
SAcresDescription = 'Acres';
```

C++

```
SAcresDescription = 'Acres';
```

File

RTLConsts

Description

This is constant RTLConsts.SAcresDescription.

83.1.11 RTLConsts.SAncestorNotFound Constant

Pascal

```
SAncestorNotFound = 'Ancestor for ''%s'' not found';
```

C++

```
SAncestorNotFound = 'Ancestor for ''%s'' not found';
```

File

RTLConsts

Description

This is constant RTLConsts.SAncestorNotFound.

83.1.12 RTLConsts.SAngstromsDescription Constant

Pascal

```
SAngstromsDescription = 'Angstroms';
```

C++

```
SAngstromsDescription = 'Angstroms';
```

File

RTLConsts

Description

This is constant RTLConsts.SAngstromsDescription.

83.1.13 RTLConsts.SAnsiUTF8Expected Constant

Pascal

```
SAnsiUTF8Expected = 'ANSI or UTF8 encoding expected';
```

C++

```
SAnsiUTF8Expected = 'ANSI or UTF8 encoding expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SAnsiUTF8Expected.

83.1.14 RTLConsts.SAreaDescription Constant

Pascal

```
SAreaDescription = 'Area';
```

C++

```
SAreaDescription = 'Area';
```

File

RTLConsts

Description

This is constant RTLConsts.SAreaDescription.

83.1.15 RTLConsts.SAresDescription Constant

Pascal

```
SAresDescription = 'Ares';
```

C++

```
SAresDescription = 'Ares';
```

File

RTLConsts

Description

This is constant RTLConsts.SAresDescription.

83.1.16 RTLConsts.sArgument_InvalidHighSurrogate Constant

Pascal

```
sArgument_InvalidHighSurrogate = 'High surrogate char without a following low surrogate char
at index: %d. Check that the string is encoded properly';
```

C++

```
sArgument_InvalidHighSurrogate = 'High surrogate char without a following low surrogate char
at index: %d. Check that the string is encoded properly';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgument_InvalidHighSurrogate.

83.1.17 RTLConsts.sArgument_InvalidLowSurrogate Constant

Pascal

```
sArgument_InvalidLowSurrogate = 'Low surrogate char without a preceding high surrogate char at
index: %d. Check that the string is encoded properly';
```

C++

```
sArgument_InvalidLowSurrogate = 'Low surrogate char without a preceding high surrogate char at
index: %d. Check that the string is encoded properly';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgument_InvalidLowSurrogate.

83.1.18 RTLConsts.sArgumentOutOfRange_Index Constant

Pascal

```
sArgumentOutOfRange_Index = 'Index out of range (%d).  Must be >= 0 and < %d';
```

C++

```
sArgumentOutOfRange_Index = 'Index out of range (%d).  Must be >= 0 and < %d';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgumentOutOfRange_Index.

83.1.19 RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate Constant

Pascal

```
sArgumentOutOfRange_InvalidHighSurrogate = 'A valid high surrogate character is >= $D800 and
<= $DBFF';
```

C++

```
sArgumentOutOfRange_InvalidHighSurrogate = 'A valid high surrogate character is >= $D800 and
<= $DBFF';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate.

83.1.20 RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate Constant

Pascal

```
sArgumentOutOfRange_InvalidLowSurrogate = 'A valid low surrogate character is >= $DC00 and <=
$DFFF';
```

C++

```
sArgumentOutOfRange_InvalidLowSurrogate = 'A valid low surrogate character is >= $DC00 and <=
$DFFF';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate.

83.1.21 RTLConsts.sArgumentOutOfRange_InvalidUTF32 Constant

Pascal

```
sArgumentOutOfRange_InvalidUTF32 = 'Invalid UTF32 character value.  Must be >= 0 and <=
$10FFF, excluding surrogate pair ranges';
```

C++

```
sArgumentOutOfRange_InvalidUTF32 = 'Invalid UTF32 character value.  Must be >= 0 and <=
$10FFF, excluding surrogate pair ranges';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgumentOutOfRange_InvalidUTF32.

83.1.22 RTLConsts.sArgumentOutOfRange_StringIndex Constant

Pascal

```
sArgumentOutOfRange_StringIndex = 'String index out of range (%d).  Must be >= 1 and <= %d';
```

C++

```
sArgumentOutOfRange_StringIndex = 'String index out of range (%d).  Must be >= 1 and <= %d';
```

File

RTLConsts

Description

This is constant RTLConsts.sArgumentOutOfRange_StringIndex.

83.1.23 RTLConsts.SAssignError Constant

Pascal

```
SAssignError = 'Cannot assign a %s to a %s';
```

C++

```
SAssignError = 'Cannot assign a %s to a %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SAssignError.

83.1.24 RTLConsts.SAstronomicalUnitsDescription Constant

Pascal

```
SAstronomicalUnitsDescription = 'AstronomicalUnits';
```

C++

```
SAstronomicalUnitsDescription = 'AstronomicalUnits';
```

File

RTLConsts

Description

This is constant RTLConsts.SAstronomicalUnitsDescription.

83.1.25 RTLConsts.sAsyncSocketError Constant

Pascal

```
sAsyncSocketError = 'Asynchronous socket error %d';
```

C++

```
sAsyncSocketError = 'Asynchronous socket error %d';
```

File

RTLConsts

Description

This is constant RTLConsts.sAsyncSocketError.

83.1.26 RTLConsts.SBitsIndexError Constant

Pascal

```
SBitsIndexError = 'Bits index out of range';
```

C++

```
SBitsIndexError = 'Bits index out of range';
```

File

RTLConsts

Description

This is constant RTLConsts.SBitsIndexError.

83.1.27 RTLConsts.SBucketListLocked Constant

Pascal

```
SBucketListLocked = 'List is locked during an active ForEach';
```

C++

```
SBucketListLocked = 'List is locked during an active ForEach';
```

File

RTLConsts

Description

This is constant RTLConsts.SBucketListLocked.

83.1.28 RTLConsts.sCannotCallAcquireOnConditionVar Constant

Pascal

```
sCannotCallAcquireOnConditionVar = 'Cannot call Acquire on TConditionVariable.  Must call
WaitFor with an external TMutex';
```

C++

```
sCannotCallAcquireOnConditionVar = 'Cannot call Acquire on TConditionVariable.  Must call
WaitFor with an external TMutex';
```

File

RTLConsts

Description

This is constant RTLConsts.sCannotCallAcquireOnConditionVar.

83.1.29 RTLConsts.sCannotCreateSocket Constant

Pascal

```
sCannotCreateSocket = 'Can''t create new socket';
```

C++

```
sCannotCreateSocket = 'Can''t create new socket';
```

File

RTLConsts

Description

This is constant RTLConsts.sCannotCreateSocket.

83.1.30 RTLConsts.sCannotListenOnOpen Constant

Pascal

```
sCannotListenOnOpen = 'Can''t listen on an open socket';
```

C++

```
sCannotListenOnOpen = 'Can''t listen on an open socket';
```

File

RTLConsts

Description

This is constant RTLConsts.sCannotListenOnOpen.

83.1.31 RTLConsts.sCantChangeWhileActive Constant

Pascal

```
sCantChangeWhileActive = 'Can''t change value while socket is active';
```

C++

```
sCantChangeWhileActive = 'Can''t change value while socket is active';
```

File

RTLConsts

Description

This is constant RTLConsts.sCantChangeWhileActive.

83.1.32 RTLConsts.SCantWriteResourceStreamError Constant

Pascal

```
SCantWriteResourceStreamError = 'Can''t write to a read-only resource stream';
```

C++

```
SCantWriteResourceStreamError = 'Can''t write to a read-only resource stream';
```

File

RTLConsts

Description

This is constant RTLConsts.SCantWriteResourceStreamError.

83.1.33 RTLConsts.SCelsiusDescription Constant

Pascal

```
SCelsiusDescription = 'Celsius';
```

C++

```
SCelsiusDescription = 'Celsius';
```

File

RTLConsts

Description

This is constant RTLConsts.SCelsiusDescription.

83.1.34 RTLConsts.SCentaresDescription Constant

Pascal

```
SCentaresDescription = 'Centares';
```

C++

```
SCentaresDescription = 'Centares';
```

File

RTLConsts

Description

This is constant RTLConsts.SCentaresDescription.

83.1.35 RTLConsts.SCentigramsDescription Constant

Pascal

```
SCentigramsDescription = 'Centigrams';
```

C++

```
SCentigramsDescription = 'Centigrams';
```

File

RTLConsts

Description

This is constant RTLConsts.SCentigramsDescription.

83.1.36 RTLConsts.SCentiLitersDescription Constant

Pascal

```
SCentiLitersDescription = 'CentiLiters';
```

C++

```
SCentiLitersDescription = 'CentiLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCentiLitersDescription.

83.1.37 RTLConsts.SCentimetersDescription Constant

Pascal

```
SCentimetersDescription = 'Centimeters';
```

C++

```
SCentimetersDescription = 'Centimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCentimetersDescription.

83.1.38 RTLConsts.SCenturiesDescription Constant

Pascal

```
SCenturiesDescription = 'Centuries';
```

C++

```
SCenturiesDescription = 'Centuries';
```

File

RTLConsts

Description

This is constant RTLConsts.SCenturiesDescription.

83.1.39 RTLConsts.SChainsDescription Constant

Pascal

```
SChainsDescription = 'Chains';
```

C++

```
SChainsDescription = 'Chains';
```

File

RTLConsts

Description

This is constant RTLConsts.SChainsDescription.

83.1.40 RTLConsts.SCharExpected Constant

Pascal

```
SCharExpected = '''''%s'''' expected';
```

C++

```
SCharExpected = '''''%s'''' expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SCharExpected.

83.1.41 RTLConsts.SCheckSynchronizeError Constant

Pascal

```
SCheckSynchronizeError = 'CheckSynchronize called from thread $%x, which is NOT the main
thread';
```

C++

```
SCheckSynchronizeError = 'CheckSynchronize called from thread $%x, which is NOT the main
thread';
```

File

RTLConsts

Description

This is constant RTLConsts.SCheckSynchronizeError.

83.1.42 RTLConsts.SClassNotFound Constant

Pascal

```
SClassNotFound = 'Class %s not found';
```

C++

```
SClassNotFound = 'Class %s not found';
```

File

RTLConsts

Description

This is constant RTLConsts.SClassNotFound.

83.1.43 RTLConsts.SCmplxCouldNotParseImaginary Constant

Pascal

```
SCmplxCouldNotParseImaginary = 'Could not parse imaginary portion';
```

C++

```
SCmplxCouldNotParseImaginary = 'Could not parse imaginary portion';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxCouldNotParseImaginary.

83.1.44 RTLConsts.SCmplxCouldNotParsePlus Constant

Pascal

```
SCmplxCouldNotParsePlus = 'Could not parse required ''+'' (or ''-'') symbol';
```

C++

```
SCmplxCouldNotParsePlus = 'Could not parse required ''+'' (or ''-'') symbol';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxCouldNotParsePlus.

83.1.45 RTLConsts.SCmplxCouldNotParseReal Constant

Pascal

```
SCmplxCouldNotParseReal = 'Could not parse real portion';
```

C++

```
SCmplxCouldNotParseReal = 'Could not parse real portion';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxCouldNotParseReal.

83.1.46 RTLConsts.SCmplxCouldNotParseSymbol Constant

Pascal

```
SCmplxCouldNotParseSymbol = 'Could not parse required ''%s'' symbol';
```

C++

```
SCmplxCouldNotParseSymbol = 'Could not parse required ''%s'' symbol';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxCouldNotParseSymbol.

83.1.47 RTLConsts.SCmplxErrorSuffix Constant

Pascal

```
SCmplxErrorSuffix = '%s [%s<?>%s]';
```

C++

```
SCmplxErrorSuffix = '%s [%s<?>%s]';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxErrorSuffix.

83.1.48 RTLConsts.SCmplxUnexpectedChars Constant

Pascal

```
SCmplxUnexpectedChars = 'Unexpected characters';
```

C++

```
SCmplxUnexpectedChars = 'Unexpected characters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxUnexpectedChars.

83.1.49 RTLConsts.SCmplxUnexpectedEOS Constant

Pascal

```
SCmplxUnexpectedEOS = 'Unexpected end of string [%s]';
```

C++

```
SCmplxUnexpectedEOS = 'Unexpected end of string [%s]';
```

File

RTLConsts

Description

This is constant RTLConsts.SCmplxUnexpectedEOS.

83.1.50 RTLConsts.SComponentNameTooLong Constant

Pascal

```
SComponentNameTooLong = 'Component name ''%s'' exceeds 64 character limit';
```

C++

```
SComponentNameTooLong = 'Component name ''%s'' exceeds 64 character limit';
```

File

RTLConsts

Description

This is constant RTLConsts.SComponentNameTooLong.

83.1.51 RTLConsts.SConvDuplicateFamily Constant

Pascal

```
SConvDuplicateFamily = 'Conversion family (%s) already registered';
```

C++

```
SConvDuplicateFamily = 'Conversion family (%s) already registered';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvDuplicateFamily.

83.1.52 RTLConsts.SConvDuplicateType Constant

Pascal

```
SConvDuplicateType = 'Conversion type (%s) already registered in %s';
```

C++

```
SConvDuplicateType = 'Conversion type (%s) already registered in %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvDuplicateType.

83.1.53 RTLConsts.SConvFactorZero Constant

Pascal

```
SConvFactorZero = '%s has a factor of zero';
```

C++

```
SConvFactorZero = '%s has a factor of zero';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvFactorZero.

83.1.54 RTLConsts.SConvIllegalFamily Constant

Pascal

```
SConvIllegalFamily = 'Illegal family';
```

C++

```
SConvIllegalFamily = 'Illegal family';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvIllegalFamily.

83.1.55 RTLConsts.SConvIllegalType Constant

Pascal

```
SConvIllegalType = 'Illegal type';
```

C++

```
SConvIllegalType = 'Illegal type';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvIllegalType.

83.1.56 RTLConsts.SConvIncompatibleTypes2 Constant

Pascal

```
SConvIncompatibleTypes2 = 'Incompatible conversion types [%s, %s]';
```

C++

```
SConvIncompatibleTypes2 = 'Incompatible conversion types [%s, %s]';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvIncompatibleTypes2.

83.1.57 RTLConsts.SConvIncompatibleTypes3 Constant

Pascal

```
SConvIncompatibleTypes3 = 'Incompatible conversion types [%s, %s, %s]';
```

C++

```
SConvIncompatibleTypes3 = 'Incompatible conversion types [%s, %s, %s]';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvIncompatibleTypes3.

83.1.58 RTLConsts.SConvIncompatibleTypes4 Constant

Pascal

```
SConvIncompatibleTypes4 = 'Incompatible conversion types [%s - %s, %s - %s]';
```

C++

```
SConvIncompatibleTypes4 = 'Incompatible conversion types [%s - %s, %s - %s]';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvIncompatibleTypes4.

83.1.59 RTLConsts.SConvStrParseError Constant

Pascal

```
SConvStrParseError = 'Could not parse %s';
```

C++

```
SConvStrParseError = 'Could not parse %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvStrParseError.

83.1.60 RTLConsts.SConvUnknownDescription Constant

Pascal

```
SConvUnknownDescription = '[$%.8x]' deprecated;
```

C++

```
SConvUnknownDescription = '[$%.8x]' deprecated;
```

File

RTLConsts

Description

This is constant RTLConsts.SConvUnknownDescription.

83.1.61 RTLConsts.SConvUnknownDescriptionWithPrefix Constant

Pascal

```
SConvUnknownDescriptionWithPrefix = '[%s%.8x]';
```

C++

```
SConvUnknownDescriptionWithPrefix = '[%s%.8x]';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvUnknownDescriptionWithPrefix.

83.1.62 RTLConsts.SConvUnknownFamily Constant

Pascal

```
SConvUnknownFamily = 'Unknown conversion family %s';
```

C++

```
SConvUnknownFamily = 'Unknown conversion family %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvUnknownFamily.

83.1.63 RTLConsts.SConvUnknownType Constant

Pascal

```
SConvUnknownType = 'Unknown conversion type %s';
```

C++

```
SConvUnknownType = 'Unknown conversion type %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SConvUnknownType.

83.1.64 RTLConsts.SCordFeetDescription Constant

Pascal

```
SCordFeetDescription = 'CordFeet';
```

C++

```
SCordFeetDescription = 'CordFeet';
```

File

RTLConsts

Description

This is constant RTLConsts.SCordFeetDescription.

83.1.65 RTLConsts.SCordsDescription Constant

Pascal

```
SCordsDescription = 'Cords';
```

C++

```
SCordsDescription = 'Cords';
```

File

RTLConsts

Description

This is constant RTLConsts.SCordsDescription.

83.1.66 RTLConsts.SCubicCentimetersDescription Constant

Pascal

```
SCubicCentimetersDescription = 'CubicCentimeters';
```

C++

```
SCubicCentimetersDescription = 'CubicCentimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicCentimetersDescription.

83.1.67 RTLConsts.SCubicDecametersDescription Constant

Pascal

```
SCubicDecametersDescription = 'CubicDecameters';
```

C++

```
SCubicDecametersDescription = 'CubicDecameters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicDecametersDescription.

83.1.68 RTLConsts.SCubicDecimetersDescription Constant

Pascal

```
SCubicDecimetersDescription = 'CubicDecimeters';
```

C++

```
SCubicDecimetersDescription = 'CubicDecimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicDecimetersDescription.

83.1.69 RTLConsts.SCubicFeetDescription Constant

Pascal

```
SCubicFeetDescription = 'CubicFeet';
```

C++

```
SCubicFeetDescription = 'CubicFeet';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicFeetDescription.

83.1.70 RTLConsts.SCubicHectometersDescription Constant

Pascal

```
SCubicHectometersDescription = 'CubicHectometers';
```

C++

```
SCubicHectometersDescription = 'CubicHectometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicHectometersDescription.

83.1.71 RTLConsts.SCubicInchesDescription Constant

Pascal

```
SCubicInchesDescription = 'CubicInches';
```

C++

```
SCubicInchesDescription = 'CubicInches';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicInchesDescription.

83.1.72 RTLConsts.SCubicKilometersDescription Constant

Pascal

```
SCubicKilometersDescription = 'CubicKilometers';
```

C++

```
SCubicKilometersDescription = 'CubicKilometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicKilometersDescription.

83.1.73 RTLConsts.SCubicMetersDescription Constant

Pascal

```
SCubicMetersDescription = 'CubicMeters';
```

C++

```
SCubicMetersDescription = 'CubicMeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicMetersDescription.

83.1.74 RTLConsts.SCubicMilesDescription Constant

Pascal

```
SCubicMilesDescription = 'CubicMiles';
```

C++

```
SCubicMilesDescription = 'CubicMiles';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicMilesDescription.

83.1.75 RTLConsts.SCubicMillimetersDescription Constant

Pascal

```
SCubicMillimetersDescription = 'CubicMillimeters';
```

C++

```
SCubicMillimetersDescription = 'CubicMillimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicMillimetersDescription.

83.1.76 RTLConsts.SCubicYardsDescription Constant

Pascal

```
SCubicYardsDescription = 'CubicYards';
```

C++

```
SCubicYardsDescription = 'CubicYards';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubicYardsDescription.

83.1.77 RTLConsts.SCubitsDescription Constant

Pascal

```
SCubitsDescription = 'Cubits';
```

C++

```
SCubitsDescription = 'Cubits';
```

File

RTLConsts

Description

This is constant RTLConsts.SCubitsDescription.

83.1.78 RTLConsts.SDateTimeDescription Constant

Pascal

```
SDateTimeDescription = 'DateTime';
```

C++

```
SDateTimeDescription = 'DateTime';
```

File

RTLConsts

Description

This is constant RTLConsts.SDateTimeDescription.

83.1.79 RTLConsts.SDaysDescription Constant

Pascal

```
SDaysDescription = 'Days';
```

C++

```
SDaysDescription = 'Days';
```

File

RTLConsts

Description

This is constant RTLConsts.SDaysDescription.

83.1.80 RTLConsts.SDecadesDescription Constant

Pascal

```
SDecadesDescription = 'Decades';
```

C++

```
SDecadesDescription = 'Decades';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecadesDescription.

83.1.81 RTLConsts.SDecagramsDescription Constant

Pascal

```
SDecagramsDescription = 'Decagrams';
```

C++

```
SDecagramsDescription = 'Decagrams';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecagramsDescription.

83.1.82 RTLConsts.SDecaLitersDescription Constant

Pascal

```
SDecaLitersDescription = 'DecaLiters';
```

C++

```
SDecaLitersDescription = 'DecaLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecaLitersDescription.

83.1.83 RTLConsts.SDecametersDescription Constant

Pascal

```
SDecametersDescription = 'Decameters';
```

C++

```
SDecametersDescription = 'Decameters';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecametersDescription.

83.1.84 RTLConsts.SDecasteresDescription Constant

Pascal

```
SDecasteresDescription = 'Decasteres';
```

C++

```
SDecasteresDescription = 'Decasteres';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecasteresDescription.

83.1.85 RTLConsts.SDecigramsDescription Constant

Pascal

```
SDecigramsDescription = 'Decigrams';
```

C++

```
SDecigramsDescription = 'Decigrams';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecigramsDescription.

83.1.86 RTLConsts.SDeciLitersDescription Constant

Pascal

```
SDeciLitersDescription = 'DeciLiters';
```

C++

```
SDeciLitersDescription = 'DeciLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SDeciLitersDescription.

83.1.87 RTLConsts.SDecimetersDescription Constant

Pascal

```
SDecimetersDescription = 'Decimeters';
```

C++

```
SDecimetersDescription = 'Decimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecimetersDescription.

83.1.88 RTLConsts.SDecisteresDescription Constant

Pascal

```
SDecisteresDescription = 'Decisteres';
```

C++

```
SDecisteresDescription = 'Decisteres';
```

File

RTLConsts

Description

This is constant RTLConsts.SDecisteresDescription.

83.1.89 RTLConsts.SDelimiterQuoteCharError Constant

Pascal

```
SDelimiterQuoteCharError = 'Delimiter and QuoteChar properties cannot have the same value';
```

C++

```
SDelimiterQuoteCharError = 'Delimiter and QuoteChar properties cannot have the same value';
```

File

RTLConsts

Description

This is constant RTLConsts.SDelimiterQuoteCharError.

83.1.90 RTLConsts.SDistanceDescription Constant

Pascal

```
SDistanceDescription = 'Distance';
```

C++

```
SDistanceDescription = 'Distance';
```

File

RTLConsts

Description

This is constant RTLConsts.SDistanceDescription.

83.1.91 RTLConsts.SDramsDescription Constant

Pascal

```
SDramsDescription = 'Drams';
```

C++

```
SDramsDescription = 'Drams';
```

File

RTLConsts

Description

This is constant RTLConsts.SDramsDescription.

83.1.92 RTLConsts.SDryBucketsDescription Constant

Pascal

```
SDryBucketsDescription = 'DryBuckets';
```

C++

```
SDryBucketsDescription = 'DryBuckets';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryBucketsDescription.

83.1.93 RTLConsts.SDryBushelsDescription Constant

Pascal

```
SDryBushelsDescription = 'DryBushels';
```

C++

```
SDryBushelsDescription = 'DryBushels';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryBushelsDescription.

83.1.94 RTLConsts.SDryGallonsDescription Constant

Pascal

```
SDryGallonsDescription = 'DryGallons';
```

C++

```
SDryGallonsDescription = 'DryGallons';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryGallonsDescription.

83.1.95 RTLConsts.SDryPecksDescription Constant

Pascal

```
SDryPecksDescription = 'DryPecks';
```

C++

```
SDryPecksDescription = 'DryPecks';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryPecksDescription.

83.1.96 RTLConsts.SDryPintsDescription Constant

Pascal

```
SDryPintsDescription = 'DryPints';
```

C++

```
SDryPintsDescription = 'DryPints';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryPintsDescription.

83.1.97 RTLConsts.SDryQuartsDescription Constant

Pascal

```
SDryQuartsDescription = 'DryQuarts';
```

C++

```
SDryQuartsDescription = 'DryQuarts';
```

File

RTLConsts

Description

This is constant RTLConsts.SDryQuartsDescription.

83.1.98 RTLConsts.SDuplicateClass Constant

Pascal

```
SDuplicateClass = 'A class named %s already exists';
```

C++

```
SDuplicateClass = 'A class named %s already exists';
```

File

RTLConsts

Description

This is constant RTLConsts.SDuplicateClass.

83.1.99 RTLConsts.SDuplicateItem Constant

Pascal

```
SDuplicateItem = 'List does not allow duplicates ($0%x)';
```

C++

```
SDuplicateItem = 'List does not allow duplicates ($0%x)';
```

File

RTLConsts

Description

This is constant RTLConsts.SDuplicateItem.

83.1.100 RTLConsts.SDuplicateName Constant

Pascal

```
SDuplicateName = 'A component named %s already exists';
```

C++

```
SDuplicateName = 'A component named %s already exists';
```

File

RTLConsts

Description

This is constant RTLConsts.SDuplicateName.

83.1.101 RTLConsts.SDuplicateString Constant

Pascal

```
SDuplicateString = 'String list does not allow duplicates';
```

C++

```
SDuplicateString = 'String list does not allow duplicates';
```

File

RTLConsts

Description

This is constant RTLConsts.SDuplicateString.

83.1.102 RTLConsts.SFahrenheitDescription Constant

Pascal

```
SFahrenheitDescription = 'Fahrenheit';
```

C++

```
SFahrenheitDescription = 'Fahrenheit';
```

File

RTLConsts

Description

This is constant RTLConsts.SFahrenheitDescription.

83.1.103 RTLConsts.SFailedToCallConstructor Constant

Pascal

```
SFailedToCallConstructor = 'TStrings descendant %s failed to call inherited constructor';
```

C++

```
SFailedToCallConstructor = 'TStrings descendant %s failed to call inherited constructor';
```

File

RTLConsts

Description

This is constant RTLConsts.SFailedToCallConstructor.

83.1.104 RTLConsts.SFathomsDescription Constant

Pascal

```
SFathomsDescription = 'Fathoms';
```

C++

```
SFathomsDescription = 'Fathoms';
```

File

RTLConsts

Description

This is constant RTLConsts.SFathomsDescription.

83.1.105 RTLConsts.SFCreateError Constant

Pascal

```
SFCreateError = 'Cannot create file %s';
```

C++

```
SFCreateError = 'Cannot create file %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SFCreateError.

83.1.106 RTLConsts.SFCreateErrorEx Constant

Pascal

```
SFCreateErrorEx = 'Cannot create file "%s". %s';
```

C++

```
SFCreateErrorEx = 'Cannot create file "%s". %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SFCreateErrorEx.

83.1.107 RTLConsts.SFeetDescription Constant

Pascal

```
SFeetDescription = 'Feet';
```

C++

```
SFeetDescription = 'Feet';
```

File

RTLConsts

Description

This is constant RTLConsts.SFeetDescription.

83.1.108 RTLConsts.SFixedColTooBig Constant

Pascal

```
SFixedColTooBig = 'Fixed column count must be less than column count';
```

C++

```
SFixedColTooBig = 'Fixed column count must be less than column count';
```

File

RTLConsts

Description

This is constant RTLConsts.SFixedColTooBig.

83.1.109 RTLConsts.SFixedRowTooBig Constant

Pascal

```
SFixedRowTooBig = 'Fixed row count must be less than row count';
```

C++

```
SFixedRowTooBig = 'Fixed row count must be less than row count';
```

File

RTLConsts

Description

This is constant RTLConsts.SFixedRowTooBig.

83.1.110 RTLConsts.SFluidCupsDescription Constant

Pascal

```
SFluidCupsDescription = 'FluidCups';
```

C++

```
SFluidCupsDescription = 'FluidCups';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidCupsDescription.

83.1.111 RTLConsts.SFluidGallonsDescription Constant

Pascal

```
SFluidGallonsDescription = 'FluidGallons';
```

C++

```
SFluidGallonsDescription = 'FluidGallons';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidGallonsDescription.

83.1.112 RTLConsts.SFluidGillsDescription Constant

Pascal

```
SFluidGillsDescription = 'FluidGills';
```

C++

```
SFluidGillsDescription = 'FluidGills';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidGillsDescription.

83.1.113 RTLConsts.SFluidOuncesDescription Constant

Pascal

```
SFluidOuncesDescription = 'FluidOunces';
```

C++

```
SFluidOuncesDescription = 'FluidOunces';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidOuncesDescription.

83.1.114 RTLConsts.SFluidPintsDescription Constant

Pascal

```
SFluidPintsDescription = 'FluidPints';
```

C++

```
SFluidPintsDescription = 'FluidPints';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidPintsDescription.

83.1.115 RTLConsts.SFluidQuartsDescription Constant

Pascal

```
SFluidQuartsDescription = 'FluidQuarts';
```

C++

```
SFluidQuartsDescription = 'FluidQuarts';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidQuartsDescription.

83.1.116 RTLConsts.SFluidTablespoonsDescription Constant

Pascal

```
SFluidTablespoonsDescription = 'FluidTablespoons';
```

C++

```
SFluidTablespoonsDescription = 'FluidTablespoons';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidTablespoonsDescription.

83.1.117 RTLConsts.SFluidTeaspoonsDescription Constant

Pascal

```
SFluidTeaspoonsDescription = 'FluidTeaspoons';
```

C++

```
SFluidTeaspoonsDescription = 'FluidTeaspoons';
```

File

RTLConsts

Description

This is constant RTLConsts.SFluidTeaspoonsDescription.

83.1.118 RTLConsts.SFOpenError Constant

Pascal

```
SFOpenError = 'Cannot open file %s';
```

C++

```
SFOpenError = 'Cannot open file %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SFOpenError.

83.1.119 RTLConsts.SFOpenErrorEx Constant

Pascal

```
SFOpenErrorEx = 'Cannot open file "%s". %s';
```

C++

```
SFOpenErrorEx = 'Cannot open file "%s". %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SFOpenErrorEx.

83.1.120 RTLConsts.SFortnightsDescription Constant

Pascal

```
SFortnightsDescription = 'Fortnights';
```

C++

```
SFortnightsDescription = 'Fortnights';
```

File

RTLConsts

Description

This is constant RTLConsts.SFortnightsDescription.

83.1.121 RTLConsts.SFurlongsDescription Constant

Pascal

```
SFurlongsDescription = 'Furlongs';
```

C++

```
SFurlongsDescription = 'Furlongs';
```

File

RTLConsts

Description

This is constant RTLConsts.SFurlongsDescription.

83.1.122 RTLConsts.SGigametersDescription Constant

Pascal

```
SGigametersDescription = 'Gigameters';
```

C++

```
SGigametersDescription = 'Gigameters';
```

File

RTLConsts

Description

This is constant RTLConsts.SGigametersDescription.

83.1.123 RTLConsts.SGrainsDescription Constant

Pascal

```
SGrainsDescription = 'Grains';
```

C++

```
SGrainsDescription = 'Grains';
```

File

RTLConsts

Description

This is constant RTLConsts.SGrainsDescription.

83.1.124 RTLConsts.SGramsDescription Constant

Pascal

```
SGramsDescription = 'Grams';
```

C++

```
SGramsDescription = 'Grams';
```

File

RTLConsts

Description

This is constant RTLConsts.SGramsDescription.

83.1.125 RTLConsts.SGridTooLarge Constant

Pascal

```
SGridTooLarge = 'Grid too large for operation';
```

C++

```
SGridTooLarge = 'Grid too large for operation';
```

File

RTLConsts

Description

This is constant RTLConsts.SGridTooLarge.

83.1.126 RTLConsts.SHandsDescription Constant

Pascal

```
SHandsDescription = 'Hands';
```

C++

```
SHandsDescription = 'Hands';
```

File

RTLConsts

Description

This is constant RTLConsts.SHandsDescription.

83.1.127 RTLConsts.SHectaresDescription Constant

Pascal

```
SHectaresDescription = 'Hectares';
```

C++

```
SHectaresDescription = 'Hectares';
```

File

RTLConsts

Description

This is constant RTLConsts.SHectaresDescription.

83.1.128 RTLConsts.SHectogramsDescription Constant

Pascal

```
SHectogramsDescription = 'Hectograms';
```

C++

```
SHectogramsDescription = 'Hectograms';
```

File

RTLConsts

Description

This is constant RTLConsts.SHectogramsDescription.

83.1.129 RTLConsts.SHectoLitersDescription Constant

Pascal

```
SHectoLitersDescription = 'HectoLiters';
```

C++

```
SHectoLitersDescription = 'HectoLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SHectoLitersDescription.

83.1.130 RTLConsts.SHectometersDescription Constant

Pascal

```
SHectometersDescription = 'Hectometers';
```

C++

```
SHectometersDescription = 'Hectometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SHectometersDescription.

83.1.131 RTLConsts.SHoursDescription Constant

Pascal

```
SHoursDescription = 'Hours';
```

C++

```
SHoursDescription = 'Hours';
```

File

RTLConsts

Description

This is constant RTLConsts.SHoursDescription.

83.1.132 RTLConsts.SIdentifierExpected Constant

Pascal

```
SIdentifierExpected = 'Identifier expected';
```

C++

```
SIdentifierExpected = 'Identifier expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SIdentifierExpected.

83.1.133 RTLConsts.SInchesDescription Constant

Pascal

```
SInchesDescription = 'Inches';
```

C++

```
SInchesDescription = 'Inches';
```

File

RTLConsts

Description

This is constant RTLConsts.SInchesDescription.

83.1.134 RTLConsts.SIndexOutOfRange Constant

Pascal

```
SIndexOutOfRange = 'Grid index out of range';
```

C++

```
SIndexOutOfRange = 'Grid index out of range';
```

File

RTLConsts

Description

This is constant RTLConsts.SIndexOutOfRange.

83.1.135 RTLConsts.SIniFileWriteError Constant

Pascal

```
SIniFileWriteError = 'Unable to write to %s';
```

C++

```
SIniFileWriteError = 'Unable to write to %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SIniFileWriteError.

83.1.136 RTLConsts.SInvalidActionCreation Constant

Pascal

```
SInvalidActionCreation = 'Invalid action creation';
```

C++

```
SInvalidActionCreation = 'Invalid action creation';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidActionCreation.

83.1.137 RTLConsts.SInvalidActionEnumeration Constant

Pascal

```
SInvalidActionEnumeration = 'Invalid action enumeration';
```

C++

```
SInvalidActionEnumeration = 'Invalid action enumeration';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidActionEnumeration.

83.1.138 RTLConsts.SInvalidActionRegistration Constant

Pascal

```
SInvalidActionRegistration = 'Invalid action registration';
```

C++

```
SInvalidActionRegistration = 'Invalid action registration';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidActionRegistration.

83.1.139 RTLConsts.SInvalidActionUnregistration Constant

Pascal

```
SInvalidActionUnregistration = 'Invalid action unregistration';
```

C++

```
SInvalidActionUnregistration = 'Invalid action unregistration';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidActionUnregistration.

83.1.140 RTLConsts.SInvalidBinary Constant

Pascal

```
SInvalidBinary = 'Invalid binary value';
```

C++

```
SInvalidBinary = 'Invalid binary value';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidBinary.

83.1.141 RTLConsts.SInvalidDate Constant

Pascal

```
SInvalidDate = '''''%s'''' is not a valid date' deprecated 'Use SysConsts.SInvalidDate';
```

C++

```
SInvalidDate = '''''%s'''' is not a valid date' deprecated 'Use SysConsts.SInvalidDate';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDate.

83.1.142 RTLConsts.SInvalidDateDay Constant

Pascal

```
SInvalidDateDay = '(%d, %d) is not a valid DateDay pair';
```

C++

```
SInvalidDateDay = '(%d, %d) is not a valid DateDay pair';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDateDay.

83.1.143 RTLConsts.SInvalidDateMonthWeek Constant

Pascal

```
SInvalidDateMonthWeek = '(%d, %d, %d, %d) is not a valid DateMonthWeek quad';
```

C++

```
SInvalidDateMonthWeek = '(%d, %d, %d, %d) is not a valid DateMonthWeek quad';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDateMonthWeek.

83.1.144 RTLConsts.SInvalidDateTime Constant

Pascal

```
SInvalidDateTime = '''''%s'''' is not a valid date and time' deprecated 'Use
SysConsts.SInvalidDateTime';
```

C++

```
SInvalidDateTime = '''''%s'''' is not a valid date and time' deprecated 'Use
SysConsts.SInvalidDateTime';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDateTime.

83.1.145 RTLConsts.SInvalidDateWeek Constant

Pascal

```
SInvalidDateWeek = '(%d, %d, %d) is not a valid DateWeek triplet';
```

C++

```
SInvalidDateWeek = '(%d, %d, %d) is not a valid DateWeek triplet';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDateWeek.

83.1.146 RTLConsts.SInvalidDayOfWeekInMonth Constant

Pascal

```
SInvalidDayOfWeekInMonth = '(%d, %d, %d, %d) is not a valid DayOfWeekInMonth quad';
```

C++

```
SInvalidDayOfWeekInMonth = '(%d, %d, %d, %d) is not a valid DayOfWeekInMonth quad';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidDayOfWeekInMonth.

83.1.147 RTLConsts.SInvalidFileName Constant

Pascal

```
SInvalidFileName = 'Invalid file name - %s';
```

C++

```
SInvalidFileName = 'Invalid file name - %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidFileName.

83.1.148 RTLConsts.SInvalidImage Constant

Pascal

```
SInvalidImage = 'Invalid stream format';
```

C++

```
SInvalidImage = 'Invalid stream format';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidImage.

83.1.149 RTLConsts.SInvalidInteger Constant

Pascal

```
SInvalidInteger = '''''%s'''' is not a valid integer value' deprecated 'Use
SysConsts.SInvalidInteger';
```

C++

```
SInvalidInteger = '''''%s'''' is not a valid integer value' deprecated 'Use
SysConsts.SInvalidInteger';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidInteger.

83.1.150 RTLConsts.SInvalidJulianDate Constant

Pascal

```
SInvalidJulianDate = '%f Julian cannot be represented as a DateTime';
```

C++

```
SInvalidJulianDate = '%f Julian cannot be represented as a DateTime';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidJulianDate.

83.1.151 RTLConsts.SInvalidMask Constant

Pascal

```
SInvalidMask = '''%s'' is an invalid mask at (%d)';
```

C++

```
SInvalidMask = '''%s'' is an invalid mask at (%d)';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidMask.

83.1.152 RTLConsts.SInvalidName Constant

Pascal

```
SInvalidName = '''''%s'''' is not a valid component name';
```

C++

```
SInvalidName = '''''%s'''' is not a valid component name';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidName.

83.1.153 RTLConsts.SInvalidProperty Constant

Pascal

```
SInvalidProperty = 'Invalid property value';
```

C++

```
SInvalidProperty = 'Invalid property value';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidProperty.

83.1.154 RTLConsts.SInvalidPropertyElement Constant

Pascal

```
SInvalidPropertyElement = 'Invalid property element: %s';
```

C++

```
SInvalidPropertyElement = 'Invalid property element: %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidPropertyElement.

83.1.155 RTLConsts.SInvalidPropertyPath Constant

Pascal

```
SInvalidPropertyPath = 'Invalid property path';
```

C++

```
SInvalidPropertyPath = 'Invalid property path';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidPropertyPath.

83.1.156 RTLConsts.SInvalidPropertyType Constant

Pascal

```
SInvalidPropertyType = 'Invalid property type: %s';
```

C++

```
SInvalidPropertyType = 'Invalid property type: %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidPropertyType.

83.1.157 RTLConsts.SInvalidPropertyValue Constant

Pascal

```
SInvalidPropertyValue = 'Invalid property value';
```

C++

```
SInvalidPropertyValue = 'Invalid property value';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidPropertyValue.

83.1.158 RTLConsts.SInvalidRegType Constant

Pascal

```
SInvalidRegType = 'Invalid data type for ''%s''';
```

C++

```
SInvalidRegType = 'Invalid data type for ''%s''';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidRegType.

83.1.159 RTLConsts.SInvalidString Constant

Pascal

```
SInvalidString = 'Invalid string constant';
```

C++

```
SInvalidString = 'Invalid string constant';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidString.

83.1.160 RTLConsts.SInvalidStringGridOp Constant

Pascal

```
SInvalidStringGridOp = 'Cannot insert or delete rows from grid';
```

C++

```
SInvalidStringGridOp = 'Cannot insert or delete rows from grid';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidStringGridOp.

83.1.161 RTLConsts.SInvalidTime Constant

Pascal

```
SInvalidTime = '''''%s'''' is not a valid time' deprecated 'Use SysConsts.SInvalidTime';
```

C++

```
SInvalidTime = '''''%s'''' is not a valid time' deprecated 'Use SysConsts.SInvalidTime';
```

File

RTLConsts

Description

This is constant RTLConsts.SInvalidTime.

83.1.162 RTLConsts.SItemNotFound Constant

Pascal

```
SItemNotFound = 'Item not found ($0%x)';
```

C++

```
SItemNotFound = 'Item not found ($0%x)';
```

File

RTLConsts

Description

This is constant RTLConsts.SItemNotFound.

83.1.163 RTLConsts.SJulianDateDescription Constant

Pascal

```
SJulianDateDescription = 'JulianDate';
```

C++

```
SJulianDateDescription = 'JulianDate';
```

File

RTLConsts

Description

This is constant RTLConsts.SJulianDateDescription.

83.1.164 RTLConsts.SKelvinDescription Constant

Pascal

```
SKelvinDescription = 'Kelvin';
```

C++

```
SKelvinDescription = 'Kelvin';
```

File

RTLConsts

Description

This is constant RTLConsts.SKelvinDescription.

83.1.165 RTLConsts.SKilogramsDescription Constant

Pascal

```
SKilogramsDescription = 'Kilograms';
```

C++

```
SKilogramsDescription = 'Kilograms';
```

File

RTLConsts

Description

This is constant RTLConsts.SKilogramsDescription.

83.1.166 RTLConsts.SKiloLitersDescription Constant

Pascal

```
SKiloLitersDescription = 'KiloLiters';
```

C++

```
SKiloLitersDescription = 'KiloLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SKiloLitersDescription.

83.1.167 RTLConsts.SKilometersDescription Constant

Pascal

```
SKilometersDescription = 'Kilometers';
```

C++

```
SKilometersDescription = 'Kilometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SKilometersDescription.

83.1.168 RTLConsts.SLightYearsDescription Constant

Pascal

```
SLightYearsDescription = 'LightYears';
```

C++

```
SLightYearsDescription = 'LightYears';
```

File

RTLConsts

Description

This is constant RTLConsts.SLightYearsDescription.

83.1.169 RTLConsts.SLineTooLong Constant

Pascal

```
SLineTooLong = 'Line too long';
```

C++

```
SLineTooLong = 'Line too long';
```

File

RTLConsts

Description

This is constant RTLConsts.SLineTooLong.

83.1.170 RTLConsts.SLinksDescription Constant

Pascal

```
SLinksDescription = 'Links';
```

C++

```
SLinksDescription = 'Links';
```

File

RTLConsts

Description

This is constant RTLConsts.SLinksDescription.

83.1.171 RTLConsts.SListCapacityError Constant

Pascal

```
SListCapacityError = 'List capacity out of bounds (%d)';
```

C++

```
SListCapacityError = 'List capacity out of bounds (%d)';
```

File

RTLConsts

Description

This is constant RTLConsts.SListCapacityError.

83.1.172 RTLConsts.SListCountError Constant

Pascal

```
SListCountError = 'List count out of bounds (%d)';
```

C++

```
SListCountError = 'List count out of bounds (%d)';
```

File

RTLConsts

Description

This is constant RTLConsts.SListCountError.

83.1.173 RTLConsts.SListIndexError Constant

Pascal

```
SListIndexError = 'List index out of bounds (%d)';
```

C++

```
SListIndexError = 'List index out of bounds (%d)';
```

File

RTLConsts

Description

This is constant RTLConsts.SListIndexError.

83.1.174 RTLConsts.SLitersDescription Constant

Pascal

```
SLitersDescription = 'Liters';
```

C++

```
SLitersDescription = 'Liters';
```

File

RTLConsts

Description

This is constant RTLConsts.SLitersDescription.

83.1.175 RTLConsts.SLongTonsDescription Constant

Pascal

```
SLongTonsDescription = 'LongTons';
```

C++

```
SLongTonsDescription = 'LongTons';
```

File

RTLConsts

Description

This is constant RTLConsts.SLongTonsDescription.

83.1.176 RTLConsts.SMaskEditErr Constant

Pascal

```
SMaskEditErr = 'Invalid input value.  Use escape key to abandon changes';
```

C++

```
SMaskEditErr = 'Invalid input value.  Use escape key to abandon changes';
```

File

RTLConsts

Description

This is constant RTLConsts.SMaskEditErr.

83.1.177 RTLConsts.SMaskErr Constant

Pascal

```
SMaskErr = 'Invalid input value';
```

C++

```
SMaskErr = 'Invalid input value';
```

File

RTLConsts

Description

This is constant RTLConsts.SMaskErr.

83.1.178 RTLConsts.SMassDescription Constant

Pascal

```
SMassDescription = 'Mass';
```

C++

```
SMassDescription = 'Mass';
```

File

RTLConsts

Description

This is constant RTLConsts.SMassDescription.

83.1.179 RTLConsts.SMegametersDescription Constant

Pascal

```
SMegametersDescription = 'Megameters';
```

C++

```
SMegametersDescription = 'Megameters';
```

File

RTLConsts

Description

This is constant RTLConsts.SMegametersDescription.

83.1.180 RTLConsts.SMemoryStreamError Constant

Pascal

```
SMemoryStreamError = 'Out of memory while expanding memory stream';
```

C++

```
SMemoryStreamError = 'Out of memory while expanding memory stream';
```

File

RTLConsts

Description

This is constant RTLConsts.SMemoryStreamError.

83.1.181 RTLConsts.SMetersDescription Constant

Pascal

```
SMetersDescription = 'Meters';
```

C++

```
SMetersDescription = 'Meters';
```

File

RTLConsts

Description

This is constant RTLConsts.SMetersDescription.

83.1.182 RTLConsts.SMetricTonsDescription Constant

Pascal

```
SMetricTonsDescription = 'MetricTons';
```

C++

```
SMetricTonsDescription = 'MetricTons';
```

File

RTLConsts

Description

This is constant RTLConsts.SMetricTonsDescription.

83.1.183 RTLConsts.SMicrogramsDescription Constant

Pascal

```
SMicrogramsDescription = 'Micrograms';
```

C++

```
SMicrogramsDescription = 'Micrograms';
```

File

RTLConsts

Description

This is constant RTLConsts.SMicrogramsDescription.

83.1.184 RTLConsts.SMicromicronsDescription Constant

Pascal

```
SMicromicronsDescription = 'Micromicrons';
```

C++

```
SMicromicronsDescription = 'Micromicrons';
```

File

RTLConsts

Description

This is constant RTLConsts.SMicromicronsDescription.

83.1.185 RTLConsts.SMicronsDescription Constant

Pascal

```
SMicronsDescription = 'Microns';
```

C++

```
SMicronsDescription = 'Microns';
```

File

RTLConsts

Description

This is constant RTLConsts.SMicronsDescription.

83.1.186 RTLConsts.SMilesDescription Constant

Pascal

```
SMilesDescription = 'Miles';
```

C++

```
SMilesDescription = 'Miles';
```

File

RTLConsts

Description

This is constant RTLConsts.SMilesDescription.

83.1.187 RTLConsts.SMillenniaDescription Constant

Pascal

```
SMillenniaDescription = 'Millennia';
```

C++

```
SMillenniaDescription = 'Millennia';
```

File

RTLConsts

Description

This is constant RTLConsts.SMillenniaDescription.

83.1.188 RTLConsts.SMilligramsDescription Constant

Pascal

```
SMilligramsDescription = 'Milligrams';
```

C++

```
SMilligramsDescription = 'Milligrams';
```

File

RTLConsts

Description

This is constant RTLConsts.SMilligramsDescription.

83.1.189 RTLConsts.SMilliLitersDescription Constant

Pascal

```
SMilliLitersDescription = 'MilliLiters';
```

C++

```
SMilliLitersDescription = 'MilliLiters';
```

File

RTLConsts

Description

This is constant RTLConsts.SMilliLitersDescription.

83.1.190 RTLConsts.SMillimetersDescription Constant

Pascal

```
SMillimetersDescription = 'Millimeters';
```

C++

```
SMillimetersDescription = 'Millimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SMillimetersDescription.

83.1.191 RTLConsts.SMillimicronsDescription Constant

Pascal

```
SMillimicronsDescription = 'Millimicrons';
```

C++

```
SMillimicronsDescription = 'Millimicrons';
```

File

RTLConsts

Description

This is constant RTLConsts.SMillimicronsDescription.

83.1.192 RTLConsts.SMilliSecondsDescription Constant

Pascal

```
SMilliSecondsDescription = 'MilliSeconds';
```

C++

```
SMilliSecondsDescription = 'MilliSeconds';
```

File

RTLConsts

Description

This is constant RTLConsts.SMilliSecondsDescription.

83.1.193 RTLConsts.SMinutesDescription Constant

Pascal

```
SMinutesDescription = 'Minutes';
```

C++

```
SMinutesDescription = 'Minutes';
```

File

RTLConsts

Description

This is constant RTLConsts.SMinutesDescription.

83.1.194 RTLConsts.SMissingDateTimeField Constant

Pascal

```
SMissingDateTimeField = '?';
```

C++

```
SMissingDateTimeField = '?';
```

File

RTLConsts

Description

This is constant RTLConsts.SMissingDateTimeField.

83.1.195 RTLConsts.SModifiedJulianDateDescription Constant

Pascal

```
SModifiedJulianDateDescription = 'ModifiedJulianDate';
```

C++

```
SModifiedJulianDateDescription = 'ModifiedJulianDate';
```

File

RTLConsts

Description

This is constant RTLConsts.SModifiedJulianDateDescription.

83.1.196 RTLConsts.SMonthsDescription Constant

Pascal

```
SMonthsDescription = 'Months';
```

C++

```
SMonthsDescription = 'Months';
```

File

RTLConsts

Description

This is constant RTLConsts.SMonthsDescription.

83.1.197 RTLConsts.SNanogramsDescription Constant

Pascal

```
SNanogramsDescription = 'Nanograms';
```

C++

```
SNanogramsDescription = 'Nanograms';
```

File

RTLConsts

Description

This is constant RTLConsts.SNanogramsDescription.

83.1.198 RTLConsts.SNauticalMilesDescription Constant

Pascal

```
SNauticalMilesDescription = 'NauticalMiles';
```

C++

```
SNauticalMilesDescription = 'NauticalMiles';
```

File

RTLConsts

Description

This is constant RTLConsts.SNauticalMilesDescription.

83.1.199 RTLConsts.sNoAddress Constant

Pascal

```
sNoAddress = 'No address specified';
```

C++

```
sNoAddress = 'No address specified';
```

File

RTLConsts

Description

This is constant RTLConsts.sNoAddress.

83.1.200 RTLConsts.SNoComSupport Constant

Pascal

```
SNoComSupport = '%s has not been registered as a COM class';
```

C++

```
SNoComSupport = '%s has not been registered as a COM class';
```

File

RTLConsts

Description

This is constant RTLConsts.SNoComSupport.

83.1.201 RTLConsts.sNoConstruct Constant

Pascal

```
sNoConstruct = 'Class %s is not intended to be constructed';
```

C++

```
sNoConstruct = 'Class %s is not intended to be constructed';
```

File

RTLConsts

Description

This is constant RTLConsts.sNoConstruct.

83.1.202 RTLConsts.SNotPrinting Constant

Pascal

```
SNotPrinting = 'Printer is not currently printing';
```

C++

```
SNotPrinting = 'Printer is not currently printing';
```

File

RTLConsts

Description

This is constant RTLConsts.SNotPrinting.

83.1.203 RTLConsts.SNumberExpected Constant

Pascal

```
SNumberExpected = 'Number expected';
```

C++

```
SNumberExpected = 'Number expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SNumberExpected.

83.1.204 RTLConsts.SOuncesDescription Constant

Pascal

```
SOuncesDescription = 'Ounces';
```

C++

```
SOuncesDescription = 'Ounces';
```

File

RTLConsts

Description

This is constant RTLConsts.SOuncesDescription.

83.1.205 RTLConsts.SPacesDescription Constant

Pascal

```
SPacesDescription = 'Paces';
```

C++

```
SPacesDescription = 'Paces';
```

File

RTLConsts

Description

This is constant RTLConsts.SPacesDescription.

83.1.206 RTLConsts.SParsecsDescription Constant

Pascal

```
SParsecsDescription = 'Parsecs';
```

C++

```
SParsecsDescription = 'Parsecs';
```

File

RTLConsts

Description

This is constant RTLConsts.SParsecsDescription.

83.1.207 RTLConsts.SParseError Constant

Pascal

```
SParseError = '%s on line %d';
```

C++

```
SParseError = '%s on line %d';
```

File

RTLConsts

Description

This is constant RTLConsts.SParseError.

83.1.208 RTLConsts.SPicasDescription Constant

Pascal

```
SPicasDescription = 'Picas';
```

C++

```
SPicasDescription = 'Picas';
```

File

RTLConsts

Description

This is constant RTLConsts.SPicasDescription.

83.1.209 RTLConsts.SPointsDescription Constant

Pascal

```
SPointsDescription = 'Points';
```

C++

```
SPointsDescription = 'Points';
```

File

RTLConsts

Description

This is constant RTLConsts.SPointsDescription.

83.1.210 RTLConsts.SPoundsDescription Constant

Pascal

```
SPoundsDescription = 'Pounds';
```

C++

```
SPoundsDescription = 'Pounds';
```

File

RTLConsts

Description

This is constant RTLConsts.SPoundsDescription.

83.1.211 RTLConsts.SPrinting Constant

Pascal

```
SPrinting = 'Printing in progress';
```

C++

```
SPrinting = 'Printing in progress';
```

File

RTLConsts

Description

This is constant RTLConsts.SPrinting.

83.1.212 RTLConsts.SPropertyException Constant

Pascal

```
SPropertyException = 'Error reading %s%s%s: %s';
```

C++

```
SPropertyException = 'Error reading %s%s%s: %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SPropertyException.

83.1.213 RTLConsts.SRankineDescription Constant

Pascal

```
SRankineDescription = 'Rankine';
```

C++

```
SRankineDescription = 'Rankine';
```

File

RTLConsts

Description

This is constant RTLConsts.SRankineDescription.

83.1.214 RTLConsts.SReadError Constant

Pascal

```
SReadError = 'Stream read error';
```

C++

```
SReadError = 'Stream read error';
```

File

RTLConsts

Description

This is constant RTLConsts.SReadError.

83.1.215 RTLConsts.SReadOnlyProperty Constant

Pascal

```
SReadOnlyProperty = 'Property is read-only';
```

C++

```
SReadOnlyProperty = 'Property is read-only';
```

File

RTLConsts

Description

This is constant RTLConsts.SReadOnlyProperty.

83.1.216 RTLConsts.SReaumurDescription Constant

Pascal

```
SReaumurDescription = 'Reaumur';
```

C++

```
SReaumurDescription = 'Reaumur';
```

File

RTLConsts

Description

This is constant RTLConsts.SReaumurDescription.

83.1.217 RTLConsts.SRegCreateFailed Constant

Pascal

```
SRegCreateFailed = 'Failed to create key %s';
```

C++

```
SRegCreateFailed = 'Failed to create key %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SRegCreateFailed.

83.1.218 RTLConsts.SRegGetDataFailed Constant

Pascal

```
SRegGetDataFailed = 'Failed to get data for ''%s''';
```

C++

```
SRegGetDataFailed = 'Failed to get data for ''%s''';
```

File

RTLConsts

Description

This is constant RTLConsts.SRegGetDataFailed.

83.1.219 RTLConsts.SRegisterError Constant

Pascal

```
SRegisterError = 'Invalid component registration';
```

C++

```
SRegisterError = 'Invalid component registration';
```

File

RTLConsts

Description

This is constant RTLConsts.SRegisterError.

83.1.220 RTLConsts.SRegSetDataFailed Constant

Pascal

```
SRegSetDataFailed = 'Failed to set data for ''%s''';
```

C++

```
SRegSetDataFailed = 'Failed to set data for ''%s''';
```

File

RTLConsts

Description

This is constant RTLConsts.SRegSetDataFailed.

83.1.221 RTLConsts.SResNotFound Constant

Pascal

```
SResNotFound = 'Resource %s not found';
```

C++

```
SResNotFound = 'Resource %s not found';
```

File

RTLConsts

Description

This is constant RTLConsts.SResNotFound.

83.1.222 RTLConsts.SRodsDescription Constant

Pascal

```
SRodsDescription = 'Rods';
```

C++

```
SRodsDescription = 'Rods';
```

File

RTLConsts

Description

This is constant RTLConsts.SRodsDescription.

83.1.223 RTLConsts.SSecondsDescription Constant

Pascal

```
SSecondsDescription = 'Seconds';
```

C++

```
SSecondsDescription = 'Seconds';
```

File

RTLConsts

Description

This is constant RTLConsts.SSecondsDescription.

83.1.224 RTLConsts.SSeekNotImplemented Constant

Pascal

```
SSeekNotImplemented = '%s.Seek not implemented';
```

C++

```
SSeekNotImplemented = '%s.Seek not implemented';
```

File

RTLConsts

Description

This is constant RTLConsts.SSeekNotImplemented.

83.1.225 RTLConsts.sSocketAlreadyOpen Constant

Pascal

```
sSocketAlreadyOpen = 'Socket already open';
```

C++

```
sSocketAlreadyOpen = 'Socket already open';
```

File

RTLConsts

Description

This is constant RTLConsts.sSocketAlreadyOpen.

83.1.226 RTLConsts.sSocketIOError Constant

Pascal

```
sSocketIOError = '%s error %d, %s';
```

C++

```
sSocketIOError = '%s error %d, %s';
```

File

RTLConsts

Description

This is constant RTLConsts.sSocketIOError.

83.1.227 RTLConsts.sSocketMustBeBlocking Constant

Pascal

```
sSocketMustBeBlocking = 'Socket must be in blocking mode';
```

C++

```
sSocketMustBeBlocking = 'Socket must be in blocking mode';
```

File

RTLConsts

Description

This is constant RTLConsts.sSocketMustBeBlocking.

83.1.228 RTLConsts.sSocketRead Constant

Pascal

```
sSocketRead = 'Read';
```

C++

```
sSocketRead = 'Read';
```

File

RTLConsts

Description

This is constant RTLConsts.sSocketRead.

83.1.229 RTLConsts.sSocketWrite Constant

Pascal

```
sSocketWrite = 'Write';
```

C++

```
sSocketWrite = 'Write';
```

File

RTLConsts

Description

This is constant RTLConsts.sSocketWrite.

83.1.230 RTLConsts.SSortedListError Constant

Pascal

```
SSortedListError = 'Operation not allowed on sorted list';
```

C++

```
SSortedListError = 'Operation not allowed on sorted list';
```

File

RTLConsts

Description

This is constant RTLConsts.SSortedListError.

83.1.231 RTLConsts.SSquareCentimetersDescription Constant

Pascal

```
SSquareCentimetersDescription = 'SquareCentimeters';
```

C++

```
SSquareCentimetersDescription = 'SquareCentimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareCentimetersDescription.

83.1.232 RTLConsts.SSquareDecametersDescription Constant

Pascal

```
SSquareDecametersDescription = 'SquareDecameters';
```

C++

```
SSquareDecametersDescription = 'SquareDecameters';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareDecametersDescription.

83.1.233 RTLConsts.SSquareDecimetersDescription Constant

Pascal

```
SSquareDecimetersDescription = 'SquareDecimeters';
```

C++

```
SSquareDecimetersDescription = 'SquareDecimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareDecimetersDescription.

83.1.234 RTLConsts.SSquareFeetDescription Constant

Pascal

```
SSquareFeetDescription = 'SquareFeet';
```

C++

```
SSquareFeetDescription = 'SquareFeet';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareFeetDescription.

83.1.235 RTLConsts.SSquareHectometersDescription Constant

Pascal

```
SSquareHectometersDescription = 'SquareHectometers';
```

C++

```
SSquareHectometersDescription = 'SquareHectometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareHectometersDescription.

83.1.236 RTLConsts.SSquareInchesDescription Constant

Pascal

```
SSquareInchesDescription = 'SquareInches';
```

C++

```
SSquareInchesDescription = 'SquareInches';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareInchesDescription.

83.1.237 RTLConsts.SSquareKilometersDescription Constant

Pascal

```
SSquareKilometersDescription = 'SquareKilometers';
```

C++

```
SSquareKilometersDescription = 'SquareKilometers';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareKilometersDescription.

83.1.238 RTLConsts.SSquareMetersDescription Constant

Pascal

```
SSquareMetersDescription = 'SquareMeters';
```

C++

```
SSquareMetersDescription = 'SquareMeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareMetersDescription.

83.1.239 RTLConsts.SSquareMilesDescription Constant

Pascal

```
SSquareMilesDescription = 'SquareMiles';
```

C++

```
SSquareMilesDescription = 'SquareMiles';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareMilesDescription.

83.1.240 RTLConsts.SSquareMillimetersDescription Constant

Pascal

```
SSquareMillimetersDescription = 'SquareMillimeters';
```

C++

```
SSquareMillimetersDescription = 'SquareMillimeters';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareMillimetersDescription.

83.1.241 RTLConsts.SSquareRodsDescription Constant

Pascal

```
SSquareRodsDescription = 'SquareRods';
```

C++

```
SSquareRodsDescription = 'SquareRods';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareRodsDescription.

83.1.242 RTLConsts.SSquareYardsDescription Constant

Pascal

```
SSquareYardsDescription = 'SquareYards';
```

C++

```
SSquareYardsDescription = 'SquareYards';
```

File

RTLConsts

Description

This is constant RTLConsts.SSquareYardsDescription.

83.1.243 RTLConsts.SSteresDescription Constant

Pascal

```
SSteresDescription = 'Steres';
```

C++

```
SSteresDescription = 'Steres';
```

File

RTLConsts

Description

This is constant RTLConsts.SSteresDescription.

83.1.244 RTLConsts.SStonesDescription Constant

Pascal

```
SStonesDescription = 'Stones';
```

C++

```
SStonesDescription = 'Stones';
```

File

RTLConsts

Description

This is constant RTLConsts.SStonesDescription.

83.1.245 RTLConsts.SStreamSetSize Constant

Pascal

```
SStreamSetSize = 'Error setting stream size';
```

C++

```
SStreamSetSize = 'Error setting stream size';
```

File

RTLConsts

Description

This is constant RTLConsts.SStreamSetSize.

83.1.246 RTLConsts.SStringExpected Constant

Pascal

```
SStringExpected = 'String expected';
```

C++

```
SStringExpected = 'String expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SStringExpected.

83.1.247 RTLConsts.SSymbolExpected Constant

Pascal

```
SSymbolExpected = '%s expected';
```

C++

```
SSymbolExpected = '%s expected';
```

File

RTLConsts

Description

This is constant RTLConsts.SSymbolExpected.

83.1.248 RTLConsts.STemperatureDescription Constant

Pascal

```
STemperatureDescription = 'Temperature';
```

C++

```
STemperatureDescription = 'Temperature';
```

File

RTLConsts

Description

This is constant RTLConsts.STemperatureDescription.

83.1.249 RTLConsts.SThreadCreateError Constant

Pascal

```
SThreadCreateError = 'Thread creation error: %s';
```

C++

```
SThreadCreateError = 'Thread creation error: %s';
```

File

RTLConsts

Description

This is constant RTLConsts.SThreadCreateError.

83.1.250 RTLConsts.SThreadError Constant

Pascal

```
SThreadError = 'Thread Error: %s (%d)';
```

C++

```
SThreadError = 'Thread Error: %s (%d)';
```

File

RTLConsts

Description

This is constant RTLConsts.SThreadError.

83.1.251 RTLConsts.SThreadExternalTerminate Constant

Pascal

```
SThreadExternalTerminate = 'Cannot terminate an externally created thread';
```

C++

```
SThreadExternalTerminate = 'Cannot terminate an externally created thread';
```

File

RTLConsts

Description

This is constant RTLConsts.SThreadExternalTerminate.

83.1.252 RTLConsts.SThreadExternalWait Constant

Pascal

```
SThreadExternalWait = 'Cannot wait for an externally created thread';
```

C++

```
SThreadExternalWait = 'Cannot wait for an externally created thread';
```

File

RTLConsts

Description

This is constant RTLConsts.SThreadExternalWait.

83.1.253 RTLConsts.STimeDescription Constant

Pascal

```
STimeDescription = 'Time';
```

C++

```
STimeDescription = 'Time';
```

File

RTLConsts

Description

This is constant RTLConsts.STimeDescription.

83.1.254 RTLConsts.STimeEncodeError Constant

Pascal

```
STimeEncodeError = 'Invalid argument to time encode' deprecated 'Use
SysConsts.STimeEncodeError';
```

C++

```
STimeEncodeError = 'Invalid argument to time encode' deprecated 'Use
SysConsts.STimeEncodeError';
```

File

RTLConsts

Description

This is constant RTLConsts.STimeEncodeError.

83.1.255 RTLConsts.STonsDescription Constant

Pascal

```
STonsDescription = 'Tons';
```

C++

```
STonsDescription = 'Tons';
```

File

RTLConsts

Description

This is constant RTLConsts.STonsDescription.

83.1.256 RTLConsts.STooManyDeleted Constant

Pascal

```
STooManyDeleted = 'Too many rows or columns deleted';
```

C++

```
STooManyDeleted = 'Too many rows or columns deleted';
```

File

RTLConsts

Description

This is constant RTLConsts.STooManyDeleted.

83.1.257 RTLConsts.SUKBucketsDescription Constant

Pascal

```
SUKBucketsDescription = 'UKBuckets';
```

C++

```
SUKBucketsDescription = 'UKBuckets';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKBucketsDescription.

83.1.258 RTLConsts.SUKBushelsDescription Constant

Pascal

```
SUKBushelsDescription = 'UKBushels';
```

C++

```
SUKBushelsDescription = 'UKBushels';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKBushelsDescription.

83.1.259 RTLConsts.SUKGallonsDescription Constant

Pascal

```
SUKGallonsDescription = 'UKGallons';
```

C++

```
SUKGallonsDescription = 'UKGallons';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKGallonsDescription.

83.1.260 RTLConsts.SUKGillsDescription Constant

Pascal

```
SUKGillsDescription = 'UKGill';
```

C++

```
SUKGillsDescription = 'UKGill';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKGillsDescription.

83.1.261 RTLConsts.SUKOuncesDescription Constant

Pascal

```
SUKOuncesDescription = 'UKOunces';
```

C++

```
SUKOuncesDescription = 'UKOunces';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKOuncesDescription.

83.1.262 RTLConsts.SUKPecksDescription Constant

Pascal

```
SUKPecksDescription = 'UKPecks';
```

C++

```
SUKPecksDescription = 'UKPecks';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKPecksDescription.

83.1.263 RTLConsts.SUKPintsDescription Constant

Pascal

```
SUKPintsDescription = 'UKPints';
```

C++

```
SUKPintsDescription = 'UKPints';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKPintsDescription.

83.1.264 RTLConsts.SUKPottlesDescription Constant

Pascal

```
SUKPottlesDescription = 'UKPottle';
```

C++

```
SUKPottlesDescription = 'UKPottle';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKPottlesDescription.

83.1.265 RTLConsts.SUKQuartsDescription Constant

Pascal

```
SUKQuartsDescription = 'UKQuarts';
```

C++

```
SUKQuartsDescription = 'UKQuarts';
```

File

RTLConsts

Description

This is constant RTLConsts.SUKQuartsDescription.

83.1.266 RTLConsts.SUnknownGroup Constant

Pascal

```
SUnknownGroup = '%s not in a class registration group';
```

C++

```
SUnknownGroup = '%s not in a class registration group';
```

File

RTLConsts

Description

This is constant RTLConsts.SUnknownGroup.

83.1.267 RTLConsts.SUnknownProperty Constant

Pascal

```
SUnknownProperty = 'Property %s does not exist';
```

C++

```
SUnknownProperty = 'Property %s does not exist';
```

File

RTLConsts

Description

This is constant RTLConsts.SUnknownProperty.

83.1.268 RTLConsts.SVolumeDescription Constant

Pascal

```
SVolumeDescription = 'Volume';
```

C++

```
SVolumeDescription = 'Volume';
```

File

RTLConsts

Description

This is constant RTLConsts.SVolumeDescription.

83.1.269 RTLConsts.SWeeksDescription Constant

Pascal

```
SWeeksDescription = 'Weeks';
```

C++

```
SWeeksDescription = 'Weeks';
```

File

RTLConsts

Description

This is constant RTLConsts.SWeeksDescription.

83.1.270 RTLConsts.sWindowsSocketError Constant

Pascal

```
sWindowsSocketError = 'Windows socket error: %s (%d), on API ''%s''';
```

C++

```
sWindowsSocketError = 'Windows socket error: %s (%d), on API ''%s''';
```

File

RTLConsts

Description

This is constant RTLConsts.sWindowsSocketError.

83.1.271 RTLConsts.SWriteError Constant

Pascal

```
SWriteError = 'Stream write error';
```

C++

```
SWriteError = 'Stream write error';
```

File

RTLConsts

Description

This is constant RTLConsts.SWriteError.

83.1.272 RTLConsts.SYardsDescription Constant

Pascal

```
SYardsDescription = 'Yards';
```

C++

```
SYardsDescription = 'Yards';
```

File

RTLConsts

Description

This is constant RTLConsts.SYardsDescription.

83.1.273 RTLConsts.SYearsDescription Constant

Pascal

```
SYearsDescription = 'Years';
```

C++

```
SYearsDescription = 'Years';
```

File

RTLConsts

Description

This is constant RTLConsts.SYearsDescription.

Index

A

aaDoubleClick enumeration member 6056
aaGetFocus enumeration member 6056
aaManual enumeration member 6056
acInternal enumeration member 5961
acMultiInstance enumeration member 5961
acSingleInstance enumeration member 5961
ActionAddUser enumeration member 4021
ActionDeleteUser enumeration member 4021
ActionDisplayUser enumeration member 4021
ActionModifyUser enumeration member 4021
afbNone enumeration member 2582
afbRaised enumeration member 2582
afbSingle enumeration member 2582
afbSunken enumeration member 2582
afDefined enumeration member 2860
afIgnored enumeration member 2860
afPremultiplied enumeration member 2860
asbLeft enumeration member 6938
asbNone enumeration member 6938
asbRight enumeration member 6938
atArrows enumeration member 2989
atSolid enumeration member 2989

B

batAppend enumeration member 3578
batAppendUpdate enumeration member 3578
batCopy enumeration member 3578
batDelete enumeration member 3578
batUpdate enumeration member 3578
bfError enumeration member 1538
bfInfo enumeration member 1538
bfNone enumeration member 1538
bfWarning enumeration member 1538
biHelp enumeration member 2583
biMaximize enumeration member 2583
biMinimize enumeration member 2583
biSystemMenu enumeration member 2583
bmDDB enumeration member 2861
bmDIB enumeration member 2861
bpLeft enumeration member 840
bpoFrame enumeration member 1539
bpoGrabber enumeration member 1539
bpoGradient enumeration member 1539
bpoRoundRect enumeration member 1539
bpRight enumeration member 840
bsBDiagonal enumeration member 2862
bsBottomLine enumeration member 1540
bsBox enumeration member 1540
bsClear enumeration member 2862
bsCross enumeration member 2862
bsDiagCross enumeration member 2862
bsDialog enumeration member 2586
bsFDiagonal enumeration member 2862
bsFrame enumeration member 1540
bsHorizontal enumeration member 2862
bsLeftLine enumeration member 1540
bsLowered enumeration member 1541
bsNone enumeration member 2586
bsRaised enumeration member 1541
bsRightLine enumeration member 1540
bsSingle enumeration member 2586
bsSizeable enumeration member 2586
bsSizeToolWin enumeration member 2586
bsSolid enumeration member 2862
bsSpacer enumeration member 1540
bsToolWindow enumeration member 2586
bsTopLine enumeration member 1540
bsVertical enumeration member 2862
btBack enumeration member 5819
btEject enumeration member 5819
btNext enumeration member 5819
btPause enumeration member 5819
btPlay enumeration member 5819
btPrev enumeration member 5819

btRecord enumeration member 5819
btStep enumeration member 5819
btStop enumeration member 5819
ByChunk enumeration member 4019
ByLine enumeration member 4019

C

caCache enumeration member 3444
caDestroy enumeration member 3444
caFree enumeration member 2584
caHide enumeration member 2584
caMinimize enumeration member 2584
caNone enumeration member 2584
capAbove enumeration member 5681
capBelow enumeration member 5681
capLeft enumeration member 5681
capRight enumeration member 5681
cbCustomColor enumeration member 1541
cbCustomColors enumeration member 1541
cbExtendedColors enumeration member 1541
cbIncludeDefault enumeration member 1541
cbIncludeNone enumeration member 1541
cbPrettyNames enumeration member 1541
cbStandardColors enumeration member 1541
cbSystemColors enumeration member 1541
ccCdecl enumeration member 4339, 5898
ccPascal enumeration member 4339, 5898
ccReg enumeration member 4339
ccRegister enumeration member 5898
ccSafeCall enumeration member 4339, 5898
ccStdCall enumeration member 4339, 5898
cd16Bit enumeration member 4227
cd24Bit enumeration member 4227
cd32Bit enumeration member 4227
cd4Bit enumeration member 4227
cd8Bit enumeration member 4227
cdDefault enumeration member 4227
cdDeviceDependent enumeration member 4227
ceLarge enumeration member 1542
ceMedium enumeration member 1542
ceNone enumeration member 1542
ceSmall enumeration member 1542
CheckDB enumeration member 4026
ckAttachToInterface enumeration member 6148
ckNewInstance enumeration member 6148
ckRemote enumeration member 6148
ckRunningInstance enumeration member 6148
ckRunningOrNew enumeration member 6148
cmDBMgmt enumeration member 3868
cmDBQuery enumeration member 3868
cmUsrMgmt enumeration member 3868
cnJDBC enumeration member 3869
cnODBC enumeration member 3869
cnOLEDB enumeration member 3869
coLeftToRight enumeration member 2863
CommitAction enumeration member 4023
CommitAdvise enumeration member 4024
CommitGlobal enumeration member 4024
CommitState enumeration member 4025
ConfigParameters enumeration member 4020
ConvertExtTables enumeration member 4018
coRightToLeft enumeration member 2863
crAlways enumeration member 3445
CreateNewDB enumeration member 4021
crOnDemand enumeration member 3445
csBrushValid enumeration member 2863
csFontValid enumeration member 2863
csHandleValid enumeration member 2863
csPenValid enumeration member 2863
ctCommands enumeration member 6936
ctDCOM enumeration member 5241
ctFromData enumeration member 6057
ctFromFile enumeration member 6057
ctLinkFromData enumeration member 6057
ctLinkToFile enumeration member 6057
ctNewObject enumeration member 6057
ctOLEnterprise enumeration member 5241
ctRecent enumeration member 6936

ctSockets enumeration member 5241
cusDeleted enumeration member 3689
cusInserted enumeration member 3689
cusModified enumeration member 3689
cusUninserted enumeration member 3689
cusUnmodified enumeration member 3689

D

Database enumeration member 4020
DataPages enumeration member 4023
DbLog enumeration member 4023
DeactivateIndexes enumeration member 4021
DenyAttachment enumeration member 4022
DenyTransaction enumeration member 4022
dkMethod enumeration member 5933
dkProperty enumeration member 5933
dkSubComponent enumeration member 5933
dmActiveForm enumeration member 2585
dmDesktop enumeration member 2585
dmMainForm enumeration member 2585
dmPrimary enumeration member 2585
dsFocus enumeration member 4228
dsGradient enumeration member 1539
dsNormal enumeration member 1539, 4228
dsSelected enumeration member 4228
dsTransparent enumeration member 4228
dtAutoSelect enumeration member 5821
dtAVIVideo enumeration member 5821
dtCDAudio enumeration member 5821
dtCDROM enumeration member 1883
dtDAT enumeration member 5821
dtDigitalVideo enumeration member 5821
dtFixed enumeration member 1883
dtFloppy enumeration member 1883
dtMMMovie enumeration member 5821
dtNetwork enumeration member 1883
dtNoDrive enumeration member 1883
dtOther enumeration member 5821
dtOverlay enumeration member 5821
dtRAM enumeration member 1883
dtScanner enumeration member 5821
dtSequencer enumeration member 5821
dtUnknown enumeration member 1883
dtVCR enumeration member 5821
dtVideodisc enumeration member 5821
dtWaveAudio enumeration member 5821

E

EBcdException class 1894
    about EBcdException class 1894
EBcdOverflowException class 1894
    about EBcdOverflowException class 1894
EBrokerException class 5908
    about EBrokerException class 5908
EDBEditError class 4592
    about EDBEditError class 4592
EDSWriter class 6443
    about EDSWriter class 6443
    Create 6444
    EDSWriter Members 6444
    EDSWriter properties 6444
    ErrorCode 6444
EGridPanelException class 163
    about EGridPanelException class 163
EHelpSystemException class 3214
    about EHelpSystemException class 3214
EIBClientError class 3543
    about EIBClientError class 3543
EIBError class 3543
    about EIBError class 3543
    Create 3544
    Create constructor 3544
    EIBError Members 3544
    EIBError properties 3545
    IBErrorCode 3545
    SQLCode 3545
EIBInstall class 3847
    about EIBInstall class 3847

Create 3848
Create constructor 3848
EIBInstall Members 3847
EIBInstall properties 3848
InstallerError 3848
InstallError 3849
EIBInstallerError class 3849
about EIBInstallerError class 3849
EIBInstallError class 3850
about EIBInstallError class 3850
EIBInterBaseError class 3545
about EIBInterBaseError class 3545
EIBInterBaseRoleError class 3546
about EIBInterBaseRoleError class 3546
EIniFileException class 4258
about EIniFileException class 4258
EInterfaceRTTIException class 4333
about EInterfaceRTTIException class 4333
EInvalidArgument class 4715
about EInvalidArgument class 4715
EInvalidGraphic class 2627
about EInvalidGraphic class 2627
EInvalidGraphicOperation class 2628
about EInvalidGraphicOperation class 2628
EInvalidGridOperation class 2999
about EInvalidGridOperation class 2999
emAddColumns enumeration member 1542
emAddRows enumeration member 1542
EMaskException class 4684
about EMaskException class 4684
EMCIDeviceError class 5749
about EMCIDeviceError class 5749
EMenuError class 4827
about EMenuError class 4827
emFixedSize enumeration member 1542
eoBLOBFilter enumeration member 3823
eoChecks enumeration member 3823
eoData enumeration member 3823
eoDatabase enumeration member 3823
eoDomain enumeration member 3823
eoException enumeration member 3823
eoForeign enumeration member 3823
eoFunction enumeration member 3823
eoGenerator enumeration member 3823
eoIndexes enumeration member 3823
EOleCtrlError class 6067
about EOleCtrlError class 6067
EOleError class 5939
about EOleError class 5939
EOleException class 5939
about EOleException class 5939
Create 5940
EOleException Members 5940
EOleException properties 5940
ErrorCode 5940
HelpFile 5941
Source 5941
EOleSysError class 5941
about EOleSysError class 5941
Create 5942
EOleSysError Members 5941
EOleSysError properties 5942
ErrorCode 5942
eoProcedure enumeration member 3823
eoRole enumeration member 3823
eoTable enumeration member 3823
eoTrigger enumeration member 3823
EOutlineError class 6257
about EOutlineError class 6257
eoView enumeration member 3823
EPrinter class 6406
about EPrinter class 6406
erAbort enumeration member 3869
erContinue enumeration member 3869
ERegistryException class 6600
about ERegistryException class 6600
ERemotableException class 4350
about ERemotableException class 4350

Create 4351
Create constructor 4351
ERemotableException Members 4351
ERemotableException properties 4352
FaultActor 4352
FaultCode 4352
FaultDetail 4353
ERibbonException class 6715
about ERibbonException class 6715
erRetry enumeration member 3869
esAddIn enumeration member 8
EScriptError class 3453
about EScriptError class 3453
Content 3455
Create 3454
Destroy 3454
Errors 3455
EScriptError Members 3453
EScriptError properties 3455
esEllipsis enumeration member 3199
esForm enumeration member 8
ESOAPDomConvertError class 6167
about ESOAPDomConvertError class 6167
esPickList enumeration member 3199
esProject enumeration member 8
esSimple enumeration member 3199
esStandard enumeration member 8
etAlterProc enumeration member 3824
etCheck enumeration member 3824
etData enumeration member 3824
etDomain enumeration member 3824
etForeign enumeration member 3824
etGrant enumeration member 3824
etIndex enumeration member 3824
etRole enumeration member 3824
etTable enumeration member 3824
etTrigger enumeration member 3824
ETypeRegistryException class 4353
about ETypeRegistryException class 4353
EWebBrokerException class 3254
about EWebBrokerException class 3254
exAPI enumeration member 3870
exDB enumeration member 3870
exDenormalized enumeration member 4769
exInvalidOp enumeration member 4769
exOverflow enumeration member 4769
exPrecision enumeration member 4769
ExptIntf 1
ExptIntf namespace 1
Classes 3
Constants 12
Functions 7
Structs, Records, Enums 8
Types 9
Variables 11
ExptIntf.ExpertEntryPoint 12
ExptIntf.ExpertEntryPoint constant 12
ExptIntf.isExperts 12
ExptIntf.isExperts constant 12
ExptIntf.LibraryExpertProc 11
ExptIntf.LibraryExpertProc variable 11
ExptIntf.RegisterLibraryExpert 7
ExptIntf.RegisterLibraryExpert function 7
ExptIntf.TExpertInitProc 9
ExptIntf.TExpertInitProc type 9
ExptIntf.TExpertRegisterProc 9
ExptIntf.TExpertRegisterProc type 9
ExptIntf.TExpertState 10
ExptIntf.TExpertState type 10
ExptIntf.TExpertStyle 8
ExptIntf.TExpertStyle enumeration 8
ExptIntf.TExpertTerminateProc 10
ExptIntf.TExpertTerminateProc type 10
ExptIntf.TIExpert 3
ExptIntf.TIExpert.Execute 4
ExptIntf.TIExpert.GetAuthor 4
ExptIntf.TIExpert.GetComment 4
ExptIntf.TIExpert.GetGlyph 4

ExptIntf.TIExpert.GetIDString 5
ExptIntf.TIExpert.GetMenuText 5
ExptIntf.TIExpert.GetName 5
ExptIntf.TIExpert.GetPage 5
ExptIntf.TIExpert.GetState 6
ExptIntf.TIExpert.GetStyle 6
ExptIntf.ToolServices 11
ExptIntf.ToolServices variable 11
ExptIntf.ValidExpertVersion 12
ExptIntf.ValidExpertVersion constant 12
ExtActns 15
ExtActns namespace 15
Classes 17
Structs, Records, Enums 157
Types 158
ExtActns.TCustomFileRun 17
ExtActns.TCustomFileRun.Browse 22
ExtActns.TCustomFileRun.BrowseDlg 23
ExtActns.TCustomFileRun.Create 19
ExtActns.TCustomFileRun.Directory 23
ExtActns.TCustomFileRun.ExecuteTarget 20
ExtActns.TCustomFileRun.FileName 23
ExtActns.TCustomFileRun.HandlesTarget 20
ExtActns.TCustomFileRun.HInst 24
ExtActns.TCustomFileRun.Notification 21
ExtActns.TCustomFileRun.Operation 25
ExtActns.TCustomFileRun.Parameters 25
ExtActns.TCustomFileRun.ParentControl 26
ExtActns.TCustomFileRun.SetupBrowseDialog 21
ExtActns.TCustomFileRun.ShowCmd 26
ExtActns.TCustomFileRun.UpdateTarget 21
ExtActns.TFileRun 27
ExtActns.TFileRun.Browse 29
ExtActns.TFileRun.BrowseDlg 29
ExtActns.TFileRun.Caption 30
ExtActns.TFileRun.Directory 30
ExtActns.TFileRun.Enabled 31
ExtActns.TFileRun.FileName 31
ExtActns.TFileRun.HelpContext 31
ExtActns.TFileRun.Hint 32
ExtActns.TFileRun.ImageIndex 32
ExtActns.TFileRun.OnHint 36
ExtActns.TFileRun.OnUpdate 36
ExtActns.TFileRun.Operation 33
ExtActns.TFileRun.Parameters 33
ExtActns.TFileRun.ParentControl 34
ExtActns.TFileRun.SecondaryShortCuts 34
ExtActns.TFileRun.ShortCut 34
ExtActns.TFileRun.ShowCmd 35
ExtActns.TFileRun.Visible 35
ExtActns.TListControlAction 37
ExtActns.TListControlAction.Create 38
ExtActns.TListControlAction.HandlesTarget 39
ExtActns.TListControlAction.IsEnabled 40
ExtActns.TListControlAction.ListControl 40
ExtActns.TListControlAction.Notification 40
ExtActns.TListControlClearSelection 41
ExtActns.TListControlClearSelection.Caption 44
ExtActns.TListControlClearSelection.Enabled 44
ExtActns.TListControlClearSelection.ExecuteTarget 42
ExtActns.TListControlClearSelection.HelpContext 44
ExtActns.TListControlClearSelection.HelpKeyword 45
ExtActns.TListControlClearSelection.HelpType 45
ExtActns.TListControlClearSelection.Hint 46
ExtActns.TListControlClearSelection.ImageIndex 46
ExtActns.TListControlClearSelection.ListControl 46
ExtActns.TListControlClearSelection.OnHint 48
ExtActns.TListControlClearSelection.OnUpdate 48
ExtActns.TListControlClearSelection.SecondaryShortCuts 47
ExtActns.TListControlClearSelection.ShortCut 47
ExtActns.TListControlClearSelection.UpdateTarget 43
ExtActns.TListControlClearSelection.Visible 47
ExtActns.TListControlCopySelection 50
ExtActns.TListControlCopySelection.Caption 53
ExtActns.TListControlCopySelection.Destination 53
ExtActns.TListControlCopySelection.Enabled 54
ExtActns.TListControlCopySelection.ExecuteTarget 51
ExtActns.TListControlCopySelection.HandlesTarget 52

ExtActns.TListControlCopySelection.HelpContext 54
ExtActns.TListControlCopySelection.HelpKeyword 55
ExtActns.TListControlCopySelection.HelpType 55
ExtActns.TListControlCopySelection.Hint 56
ExtActns.TListControlCopySelection.ImageIndex 56
ExtActns.TListControlCopySelection.ListControl 56
ExtActns.TListControlCopySelection.OnHint 58
ExtActns.TListControlCopySelection.OnUpdate 58
ExtActns.TListControlCopySelection.SecondaryShortCuts 57
ExtActns.TListControlCopySelection.ShortCut 57
ExtActns.TListControlCopySelection.UpdateTarget 52
ExtActns.TListControlCopySelection.Visible 57
ExtActns.TListControlDeleteSelection 60
ExtActns.TListControlDeleteSelection.Caption 62
ExtActns.TListControlDeleteSelection.Enabled 63
ExtActns.TListControlDeleteSelection.ExecuteTarget 61
ExtActns.TListControlDeleteSelection.HelpContext 63
ExtActns.TListControlDeleteSelection.HelpKeyword 64
ExtActns.TListControlDeleteSelection.HelpType 64
ExtActns.TListControlDeleteSelection.Hint 65
ExtActns.TListControlDeleteSelection.ImageIndex 65
ExtActns.TListControlDeleteSelection.ListControl 65
ExtActns.TListControlDeleteSelection.OnHint 67
ExtActns.TListControlDeleteSelection.OnUpdate 67
ExtActns.TListControlDeleteSelection.SecondaryShortCuts 66
ExtActns.TListControlDeleteSelection.ShortCut 66
ExtActns.TListControlDeleteSelection.UpdateTarget 62
ExtActns.TListControlDeleteSelection.Visible 66
ExtActns.TListControlMoveSelection 68
ExtActns.TListControlMoveSelection.ExecuteTarget 70
ExtActns.TListControlSelectAll 71
ExtActns.TListControlSelectAll.Caption 74
ExtActns.TListControlSelectAll.Enabled 74
ExtActns.TListControlSelectAll.ExecuteTarget 72
ExtActns.TListControlSelectAll.HelpContext 74
ExtActns.TListControlSelectAll.HelpKeyword 75
ExtActns.TListControlSelectAll.HelpType 75
ExtActns.TListControlSelectAll.Hint 76
ExtActns.TListControlSelectAll.ImageIndex 76
ExtActns.TListControlSelectAll.ListControl 76
ExtActns.TListControlSelectAll.OnHint 78
ExtActns.TListControlSelectAll.OnUpdate 78
ExtActns.TListControlSelectAll.SecondaryShortCuts 77
ExtActns.TListControlSelectAll.ShortCut 77
ExtActns.TListControlSelectAll.UpdateTarget 73
ExtActns.TListControlSelectAll.Visible 77
ExtActns.TNextTab 80
ExtActns.TNextTab.AfterTabChange 89
ExtActns.TNextTab.BeforeTabChange 89
ExtActns.TNextTab.Caption 83
ExtActns.TNextTab.Enabled 83
ExtActns.TNextTab.ExecuteTarget 81
ExtActns.TNextTab.HelpContext 84
ExtActns.TNextTab.HelpKeyword 84
ExtActns.TNextTab.HelpType 85
ExtActns.TNextTab.Hint 85
ExtActns.TNextTab.ImageIndex 85
ExtActns.TNextTab.LastTabCaption 86
ExtActns.TNextTab.OnFinish 90
ExtActns.TNextTab.OnHint 90
ExtActns.TNextTab.OnUpdate 90
ExtActns.TNextTab.OnValidateTab 92
ExtActns.TNextTab.ShortCut 86
ExtActns.TNextTab.SkipHiddenTabs 87
ExtActns.TNextTab.TabControl 87
ExtActns.TNextTab.UpdateTarget 82
ExtActns.TNextTab.Visible 88
ExtActns.TNextTab.Wrap 88
ExtActns.TOpenPicture 92
ExtActns.TOpenPicture.Caption 94
ExtActns.TOpenPicture.Dialog 94
ExtActns.TOpenPicture.Enabled 95
ExtActns.TOpenPicture.GetDialogClass 94
ExtActns.TOpenPicture.HelpContext 95
ExtActns.TOpenPicture.HelpKeyword 96
ExtActns.TOpenPicture.HelpType 96
ExtActns.TOpenPicture.Hint 97
ExtActns.TOpenPicture.ImageIndex 97

ExtActns.TOpenPicture.OnAccept 99
ExtActns.TOpenPicture.OnCancel 99
ExtActns.TOpenPicture.OnHint 99
ExtActns.TOpenPicture.OnUpdate 99
ExtActns.TOpenPicture.SecondaryShortCuts 97
ExtActns.TOpenPicture.ShortCut 98
ExtActns.TOpenPicture.Visible 98
ExtActns.TPreviousTab 101
ExtActns.TPreviousTab.AfterTabChange 109
ExtActns.TPreviousTab.BeforeTabChange 109
ExtActns.TPreviousTab.Caption 104
ExtActns.TPreviousTab.Enabled 104
ExtActns.TPreviousTab.ExecuteTarget 102
ExtActns.TPreviousTab.HelpContext 105
ExtActns.TPreviousTab.HelpKeyword 105
ExtActns.TPreviousTab.HelpType 106
ExtActns.TPreviousTab.Hint 106
ExtActns.TPreviousTab.ImageIndex 106
ExtActns.TPreviousTab.OnHint 110
ExtActns.TPreviousTab.OnUpdate 110
ExtActns.TPreviousTab.OnValidateTab 111
ExtActns.TPreviousTab.ShortCut 107
ExtActns.TPreviousTab.SkipHiddenTabs 107
ExtActns.TPreviousTab.TabControl 107
ExtActns.TPreviousTab.UpdateTarget 103
ExtActns.TPreviousTab.Visible 108
ExtActns.TPreviousTab.Wrap 108
ExtActns.TRichEditAction 112
ExtActns.TRichEditAction.Create 113
ExtActns.TRichEditAction.CurrText 114
ExtActns.TRichEditAction.HandlesTarget 114
ExtActns.TRichEditAction.SetFontStyle 114
ExtActns.TRichEditAlignCenter 115
ExtActns.TRichEditAlignCenter.ExecuteTarget 116
ExtActns.TRichEditAlignCenter.UpdateTarget 116
ExtActns.TRichEditAlignLeft 117
ExtActns.TRichEditAlignLeft.ExecuteTarget 118
ExtActns.TRichEditAlignLeft.UpdateTarget 118
ExtActns.TRichEditAlignRight 119
ExtActns.TRichEditAlignRight.ExecuteTarget 120
ExtActns.TRichEditAlignRight.UpdateTarget 120
ExtActns.TRichEditBold 121
ExtActns.TRichEditBold.ExecuteTarget 122
ExtActns.TRichEditBold.UpdateTarget 123
ExtActns.TRichEditBullets 123
ExtActns.TRichEditBullets.ExecuteTarget 124
ExtActns.TRichEditBullets.UpdateTarget 125
ExtActns.TRichEditItalic 125
ExtActns.TRichEditItalic.ExecuteTarget 127
ExtActns.TRichEditItalic.UpdateTarget 127
ExtActns.TRichEditStrikeOut 127
ExtActns.TRichEditStrikeOut.ExecuteTarget 129
ExtActns.TRichEditStrikeOut.UpdateTarget 129
ExtActns.TRichEditUnderline 130
ExtActns.TRichEditUnderline.ExecuteTarget 131
ExtActns.TRichEditUnderline.UpdateTarget 131
ExtActns.TSavePicture 132
ExtActns.TSavePicture.Caption 134
ExtActns.TSavePicture.Dialog 134
ExtActns.TSavePicture.Enabled 135
ExtActns.TSavePicture.GetDialogClass 133
ExtActns.TSavePicture.HelpContext 135
ExtActns.TSavePicture.HelpKeyword 136
ExtActns.TSavePicture.HelpType 136
ExtActns.TSavePicture.Hint 137
ExtActns.TSavePicture.ImageIndex 137
ExtActns.TSavePicture.OnAccept 139
ExtActns.TSavePicture.OnCancel 139
ExtActns.TSavePicture.OnHint 139
ExtActns.TSavePicture.OnUpdate 139
ExtActns.TSavePicture.SecondaryShortCuts 137
ExtActns.TSavePicture.ShortCut 138
ExtActns.TSavePicture.Visible 138
ExtActns.TSendMail 141
ExtActns.TSendMail.Caption 144
ExtActns.TSendMail.Destroy 142
ExtActns.TSendMail.Enabled 144
ExtActns.TSendMail.ExecuteTarget 142

ExtActns.TSendMail.HandlesTarget 143
ExtActns.TSendMail.HelpContext 144
ExtActns.TSendMail.HelpKeyword 145
ExtActns.TSendMail.HelpType 145
ExtActns.TSendMail.Hint 146
ExtActns.TSendMail.ImageIndex 146
ExtActns.TSendMail.OnHint 148
ExtActns.TSendMail.OnUpdate 148
ExtActns.TSendMail.SecondaryShortCuts 146
ExtActns.TSendMail.ShortCut 147
ExtActns.TSendMail.Text 147
ExtActns.TSendMail.UpdateTarget 143
ExtActns.TSendMail.Visible 148
ExtActns.TShowCmd 157
ExtActns.TShowCmd enumeration 157
ExtActns.TTabAction 150
ExtActns.TTabAction.AfterTabChange 155
ExtActns.TTabAction.BeforeTabChange 155
ExtActns.TTabAction.Create 151
ExtActns.TTabAction.HandlesTarget 152
ExtActns.TTabAction.Notification 152
ExtActns.TTabAction.OnValidateTab 156
ExtActns.TTabAction.SelectNextTab 152
ExtActns.TTabAction.SkipHiddenTabs 153
ExtActns.TTabAction.TabControl 154
ExtActns.TTabAction.UpdateTarget 153
ExtActns.TTabAction.Wrap 154
ExtActns.TValidateTabEvent 158
ExtActns.TValidateTabEvent type 158
ExtCtrls 159
ExtCtrls namespace 159
    Classes 162
    Constants 1556
    Functions 1536
    Structs, Records, Enums 1538
    Types 1547
ExtCtrls.DefaultCategoryPanelHeight 1556
ExtCtrls.DefaultCategoryPanelHeight constant 1556
ExtCtrls.EGridPanelException 163
ExtCtrls.Frame3D 1536
ExtCtrls.Frame3D function 1536
ExtCtrls.NaturalNumber 1547
ExtCtrls.NaturalNumber type 1547
ExtCtrls.NoColorSelected 1556
ExtCtrls.NoColorSelected constant 1556
ExtCtrls.NotebookHandlesNeeded 1537
ExtCtrls.NotebookHandlesNeeded function 1537
ExtCtrls.TBalloonFlags 1538
ExtCtrls.TBalloonFlags enumeration 1538
ExtCtrls.TBandDragEvent 1548
ExtCtrls.TBandDragEvent type 1548
ExtCtrls.TBandDrawingStyle 1539
ExtCtrls.TBandDrawingStyle enumeration 1539
ExtCtrls.TBandInfoEvent 1548
ExtCtrls.TBandInfoEvent type 1548
ExtCtrls.TBandMoveEvent 1548
ExtCtrls.TBandMoveEvent type 1548
ExtCtrls.TBandPaintEvent 1549
ExtCtrls.TBandPaintEvent type 1549
ExtCtrls.TBandPaintOption 1539
ExtCtrls.TBandPaintOption enumeration 1539
ExtCtrls.TBandPaintOptions 1549
ExtCtrls.TBandPaintOptions type 1549
ExtCtrls.TBeginBandMoveEvent 1549
ExtCtrls.TBeginBandMoveEvent type 1549
ExtCtrls.TBevel 163
ExtCtrls.TBevel.Align 165
ExtCtrls.TBevel.Anchors 169
ExtCtrls.TBevel.Constraints 169
ExtCtrls.TBevel.Create 164
ExtCtrls.TBevel.Paint 165
ExtCtrls.TBevel.ParentShowHint 170
ExtCtrls.TBevel.Shape 170
ExtCtrls.TBevel.ShowHint 171
ExtCtrls.TBevel.Style 172
ExtCtrls.TBevel.Visible 173
ExtCtrls.TBevelShape 1540
ExtCtrls.TBevelShape enumeration 1540

ExtCtrls.TBevelStyle 1541
ExtCtrls.TBevelStyle enumeration 1541
ExtCtrls.TBoundLabel 174
ExtCtrls.TBoundLabel.AdjustBounds 176
ExtCtrls.TBoundLabel.BiDiMode 178
ExtCtrls.TBoundLabel.Caption 178
ExtCtrls.TBoundLabel.Color 181
ExtCtrls.TBoundLabel.Create 176
ExtCtrls.TBoundLabel.DragCursor 181
ExtCtrls.TBoundLabel.DragKind 182
ExtCtrls.TBoundLabel.DragMode 182
ExtCtrls.TBoundLabel.Font 184
ExtCtrls.TBoundLabel.Height 185
ExtCtrls.TBoundLabel.Layout 186
ExtCtrls.TBoundLabel.Left 186
ExtCtrls.TBoundLabel.OnClick 201
ExtCtrls.TBoundLabel.OnContextPopup 203
ExtCtrls.TBoundLabel.OnDblClick 203
ExtCtrls.TBoundLabel.OnDragDrop 204
ExtCtrls.TBoundLabel.OnDragOver 207
ExtCtrls.TBoundLabel.OnEndDock 210
ExtCtrls.TBoundLabel.OnEndDrag 210
ExtCtrls.TBoundLabel.OnMouseActivate 211
ExtCtrls.TBoundLabel.OnMouseDown 212
ExtCtrls.TBoundLabel.OnMouseMove 214
ExtCtrls.TBoundLabel.OnMouseUp 217
ExtCtrls.TBoundLabel.OnStartDock 219
ExtCtrls.TBoundLabel.OnStartDrag 220
ExtCtrls.TBoundLabel.ParentBiDiMode 187
ExtCtrls.TBoundLabel.ParentColor 188
ExtCtrls.TBoundLabel.ParentFont 188
ExtCtrls.TBoundLabel.ParentShowHint 189
ExtCtrls.TBoundLabel.PopupMenu 189
ExtCtrls.TBoundLabel.ShowAccelChar 194
ExtCtrls.TBoundLabel.ShowHint 195
ExtCtrls.TBoundLabel.Top 197
ExtCtrls.TBoundLabel.Transparent 198
ExtCtrls.TBoundLabel.Width 198
ExtCtrls.TBoundLabel.WordWrap 200
ExtCtrls.TButtonedEdit 220
ExtCtrls.TButtonedEdit.Align 227
ExtCtrls.TButtonedEdit.Alignment 230
ExtCtrls.TButtonedEdit.Anchors 231
ExtCtrls.TButtonedEdit.AutoSelect 231
ExtCtrls.TButtonedEdit.AutoSize 232
ExtCtrls.TButtonedEdit.BevelEdges 233
ExtCtrls.TButtonedEdit.BevelInner 233
ExtCtrls.TButtonedEdit.BevelKind 234
ExtCtrls.TButtonedEdit.BevelOuter 234
ExtCtrls.TButtonedEdit.BevelWidth 234
ExtCtrls.TButtonedEdit.BiDiMode 235
ExtCtrls.TButtonedEdit.BorderStyle 235
ExtCtrls.TButtonedEdit.CharCase 236
ExtCtrls.TButtonedEdit.Color 236
ExtCtrls.TButtonedEdit.Constraints 237
ExtCtrls.TButtonedEdit.Ctl3D 237
ExtCtrls.TButtonedEdit.DoubleBuffered 238
ExtCtrls.TButtonedEdit.DragCursor 240
ExtCtrls.TButtonedEdit.DragKind 240
ExtCtrls.TButtonedEdit.DragMode 240
ExtCtrls.TButtonedEdit.Enabled 242
ExtCtrls.TButtonedEdit.Font 242
ExtCtrls.TButtonedEdit.HideSelection 244
ExtCtrls.TButtonedEdit.Images 244
ExtCtrls.TButtonedEdit.ImeMode 247
ExtCtrls.TButtonedEdit.ImeName 248
ExtCtrls.TButtonedEdit.LeftButton 248
ExtCtrls.TButtonedEdit.MaxLength 252
ExtCtrls.TButtonedEdit.NumbersOnly 252
ExtCtrls.TButtonedEdit.OEMConvert 252
ExtCtrls.TButtonedEdit.OnChange 270
ExtCtrls.TButtonedEdit.OnClick 271
ExtCtrls.TButtonedEdit.OnContextPopup 273
ExtCtrls.TButtonedEdit.OnDblClick 273
ExtCtrls.TButtonedEdit.OnDragDrop 274
ExtCtrls.TButtonedEdit.OnDragOver 277
ExtCtrls.TButtonedEdit.OnEndDock 280
ExtCtrls.TButtonedEdit.OnEndDrag 280

ExtCtrls.TButtonedEdit.OnEnter 281
ExtCtrls.TButtonedEdit.OnExit 284
ExtCtrls.TButtonedEdit.OnKeyDown 287
ExtCtrls.TButtonedEdit.OnKeyPress 290
ExtCtrls.TButtonedEdit.OnKeyUp 291
ExtCtrls.TButtonedEdit.OnLeftButtonClick 293
ExtCtrls.TButtonedEdit.OnMouseActivate 296
ExtCtrls.TButtonedEdit.OnMouseDown 297
ExtCtrls.TButtonedEdit.OnMouseEnter 299
ExtCtrls.TButtonedEdit.OnMouseLeave 300
ExtCtrls.TButtonedEdit.OnMouseMove 300
ExtCtrls.TButtonedEdit.OnMouseUp 302
ExtCtrls.TButtonedEdit.OnRightButtonClick 304
ExtCtrls.TButtonedEdit.OnStartDock 307
ExtCtrls.TButtonedEdit.OnStartDrag 308
ExtCtrls.TButtonedEdit.ParentBiDiMode 253
ExtCtrls.TButtonedEdit.ParentColor 253
ExtCtrls.TButtonedEdit.ParentCtl3D 254
ExtCtrls.TButtonedEdit.ParentDoubleBuffered 254
ExtCtrls.TButtonedEdit.ParentFont 254
ExtCtrls.TButtonedEdit.ParentShowHint 255
ExtCtrls.TButtonedEdit.PasswordChar 255
ExtCtrls.TButtonedEdit.PopupMenu 256
ExtCtrls.TButtonedEdit.ReadOnly 261
ExtCtrls.TButtonedEdit.RightButton 261
ExtCtrls.TButtonedEdit.ShowHint 264
ExtCtrls.TButtonedEdit.TabOrder 266
ExtCtrls.TButtonedEdit.TabStop 266
ExtCtrls.TButtonedEdit.Text 267
ExtCtrls.TButtonedEdit.TextHint 268
ExtCtrls.TButtonedEdit.Visible 268
ExtCtrls.TCanResizeEvent 1550
ExtCtrls.TCanResizeEvent type 1550
ExtCtrls.TCategoryPanel 309
ExtCtrls.TCategoryPanel.BiDiMode 313
ExtCtrls.TCategoryPanel.Caption 314
ExtCtrls.TCategoryPanel.Collapsed 316
ExtCtrls.TCategoryPanel.CollapsedHotImageIndex 316
ExtCtrls.TCategoryPanel.CollapsedImageIndex 317
ExtCtrls.TCategoryPanel.CollapsedPressedImageIndex 317
ExtCtrls.TCategoryPanel.Color 318
ExtCtrls.TCategoryPanel.Ctl3D 318
ExtCtrls.TCategoryPanel.DockSite 319
ExtCtrls.TCategoryPanel.DoubleBuffered 320
ExtCtrls.TCategoryPanel.DragCursor 322
ExtCtrls.TCategoryPanel.DragKind 322
ExtCtrls.TCategoryPanel.DragMode 322
ExtCtrls.TCategoryPanel.Enabled 324
ExtCtrls.TCategoryPanel.ExpandedHotImageIndex 324
ExtCtrls.TCategoryPanel.ExpandedImageIndex 325
ExtCtrls.TCategoryPanel.ExpandedPressedImageIndex 325
ExtCtrls.TCategoryPanel.Font 326
ExtCtrls.TCategoryPanel.FullRepaint 327
ExtCtrls.TCategoryPanel.Height 327
ExtCtrls.TCategoryPanel.Left 328
ExtCtrls.TCategoryPanel.Locked 329
ExtCtrls.TCategoryPanel.OnClick 344
ExtCtrls.TCategoryPanel.OnCollapse 346
ExtCtrls.TCategoryPanel.OnContextPopup 346
ExtCtrls.TCategoryPanel.OnDblClick 347
ExtCtrls.TCategoryPanel.OnDockDrop 347
ExtCtrls.TCategoryPanel.OnDockOver 349
ExtCtrls.TCategoryPanel.OnDragDrop 352
ExtCtrls.TCategoryPanel.OnDragOver 355
ExtCtrls.TCategoryPanel.OnEndDock 358
ExtCtrls.TCategoryPanel.OnEndDrag 358
ExtCtrls.TCategoryPanel.OnEnter 359
ExtCtrls.TCategoryPanel.OnExit 362
ExtCtrls.TCategoryPanel.OnExpand 364
ExtCtrls.TCategoryPanel.OnGetSiteInfo 365
ExtCtrls.TCategoryPanel.OnMouseActivate 367
ExtCtrls.TCategoryPanel.OnMouseDown 368
ExtCtrls.TCategoryPanel.OnMouseEnter 370
ExtCtrls.TCategoryPanel.OnMouseLeave 370
ExtCtrls.TCategoryPanel.OnMouseMove 371
ExtCtrls.TCategoryPanel.OnMouseUp 373
ExtCtrls.TCategoryPanel.OnStartDock 375
ExtCtrls.TCategoryPanel.OnStartDrag 376

ExtCtrls.TCategoryPanel.OnUnDock 376
ExtCtrls.TCategoryPanel.Padding 329
ExtCtrls.TCategoryPanel.ParentBackground 330
ExtCtrls.TCategoryPanel.ParentBiDiMode 330
ExtCtrls.TCategoryPanel.ParentColor 330
ExtCtrls.TCategoryPanel.ParentCtl3D 331
ExtCtrls.TCategoryPanel.ParentDoubleBuffered 331
ExtCtrls.TCategoryPanel.ParentFont 332
ExtCtrls.TCategoryPanel.ParentShowHint 332
ExtCtrls.TCategoryPanel.PopupMenu 333
ExtCtrls.TCategoryPanel.ShowHint 338
ExtCtrls.TCategoryPanel.TabOrder 339
ExtCtrls.TCategoryPanel.TabStop 340
ExtCtrls.TCategoryPanel.UseDockManager 340
ExtCtrls.TCategoryPanel.Visible 340
ExtCtrls.TCategoryPanel.Width 342
ExtCtrls.TCategoryPanelClass 1550
ExtCtrls.TCategoryPanelClass type 1550
ExtCtrls.TCategoryPanelGroup 379
ExtCtrls.TCategoryPanelGroup.Align 383
ExtCtrls.TCategoryPanelGroup.Anchors 387
ExtCtrls.TCategoryPanelGroup.BevelEdges 387
ExtCtrls.TCategoryPanelGroup.BevelInner 388
ExtCtrls.TCategoryPanelGroup.BevelKind 388
ExtCtrls.TCategoryPanelGroup.BevelOuter 389
ExtCtrls.TCategoryPanelGroup.BevelWidth 389
ExtCtrls.TCategoryPanelGroup.BiDiMode 389
ExtCtrls.TCategoryPanelGroup.ChevronAlignment 390
ExtCtrls.TCategoryPanelGroup.ChevronColor 390
ExtCtrls.TCategoryPanelGroup.ChevronHotColor 391
ExtCtrls.TCategoryPanelGroup.Color 391
ExtCtrls.TCategoryPanelGroup.Constraints 392
ExtCtrls.TCategoryPanelGroup.Ctl3D 393
ExtCtrls.TCategoryPanelGroup.DockSite 393
ExtCtrls.TCategoryPanelGroup.DoubleBuffered 394
ExtCtrls.TCategoryPanelGroup.DragCursor 396
ExtCtrls.TCategoryPanelGroup.DragKind 396
ExtCtrls.TCategoryPanelGroup.DragMode 397
ExtCtrls.TCategoryPanelGroup.Enabled 398
ExtCtrls.TCategoryPanelGroup.Font 399
ExtCtrls.TCategoryPanelGroup.GradientBaseColor 400
ExtCtrls.TCategoryPanelGroup.GradientColor 400
ExtCtrls.TCategoryPanelGroup.GradientDirection 400
ExtCtrls.TCategoryPanelGroup.HeaderAlignment 401
ExtCtrls.TCategoryPanelGroup.HeaderFont 401
ExtCtrls.TCategoryPanelGroup.HeaderHeight 401
ExtCtrls.TCategoryPanelGroup.HeaderImage 402
ExtCtrls.TCategoryPanelGroup.HeaderStyle 402
ExtCtrls.TCategoryPanelGroup.Height 402
ExtCtrls.TCategoryPanelGroup.Images 403
ExtCtrls.TCategoryPanelGroup.OnCanResize 418
ExtCtrls.TCategoryPanelGroup.OnClick 419
ExtCtrls.TCategoryPanelGroup.OnConstrainedResize 421
ExtCtrls.TCategoryPanelGroup.OnContextPopup 421
ExtCtrls.TCategoryPanelGroup.OnDblClick 422
ExtCtrls.TCategoryPanelGroup.OnDockDrop 422
ExtCtrls.TCategoryPanelGroup.OnDockOver 424
ExtCtrls.TCategoryPanelGroup.OnDragDrop 427
ExtCtrls.TCategoryPanelGroup.OnDragOver 430
ExtCtrls.TCategoryPanelGroup.OnEndDock 433
ExtCtrls.TCategoryPanelGroup.OnEndDrag 433
ExtCtrls.TCategoryPanelGroup.OnEnter 434
ExtCtrls.TCategoryPanelGroup.OnExit 437
ExtCtrls.TCategoryPanelGroup.OnGetSiteInfo 439
ExtCtrls.TCategoryPanelGroup.OnMouseActivate 442
ExtCtrls.TCategoryPanelGroup.OnMouseDown 443
ExtCtrls.TCategoryPanelGroup.OnMouseEnter 445
ExtCtrls.TCategoryPanelGroup.OnMouseLeave 445
ExtCtrls.TCategoryPanelGroup.OnMouseMove 445
ExtCtrls.TCategoryPanelGroup.OnMouseUp 448
ExtCtrls.TCategoryPanelGroup.OnMouseWheel 450
ExtCtrls.TCategoryPanelGroup.OnMouseWheelDown 450
ExtCtrls.TCategoryPanelGroup.OnMouseWheelUp 451
ExtCtrls.TCategoryPanelGroup.OnResize 451
ExtCtrls.TCategoryPanelGroup.OnStartDock 452
ExtCtrls.TCategoryPanelGroup.OnStartDrag 452
ExtCtrls.TCategoryPanelGroup.OnUnDock 453
ExtCtrls.TCategoryPanelGroup.Padding 404

ExtCtrls.TCategoryPanelGroup.ParentBackground 404
ExtCtrls.TCategoryPanelGroup.ParentBiDiMode 404
ExtCtrls.TCategoryPanelGroup.ParentColor 405
ExtCtrls.TCategoryPanelGroup.ParentCtl3D 405
ExtCtrls.TCategoryPanelGroup.ParentDoubleBuffered 406
ExtCtrls.TCategoryPanelGroup.ParentFont 406
ExtCtrls.TCategoryPanelGroup.ParentShowHint 406
ExtCtrls.TCategoryPanelGroup.PopupMenu 407
ExtCtrls.TCategoryPanelGroup.ShowHint 412
ExtCtrls.TCategoryPanelGroup.TabOrder 414
ExtCtrls.TCategoryPanelGroup.TabStop 414
ExtCtrls.TCategoryPanelGroup.Visible 415
ExtCtrls.TCategoryPanelGroup.Width 416
ExtCtrls.TCategoryPanelSurface 456
ExtCtrls.TCategoryPanelSurface.Create 456
ExtCtrls.TCategoryPanelSurface.CreateParams 457
ExtCtrls.TCategoryPanelSurfaceClass 1550
ExtCtrls.TCategoryPanelSurfaceClass type 1550
ExtCtrls.TCellCollection 458
ExtCtrls.TCellCollection.GetAttr 459
ExtCtrls.TCellCollection.GetAttrCount 459
ExtCtrls.TCellCollection.GetItem 460
ExtCtrls.TCellCollection.GetItemAttr 460
ExtCtrls.TCellCollection.Items 461
ExtCtrls.TCellCollection.Owner 460
ExtCtrls.TCellCollection.SetItem 461
ExtCtrls.TCellCollection.Update 461
ExtCtrls.TCellItem 462
ExtCtrls.TCellItem.AssignTo 463
ExtCtrls.TCellItem.AutoAdded 464
ExtCtrls.TCellItem.Create 462
ExtCtrls.TCellItem.SetSizeStyle 464
ExtCtrls.TCellItem.SetValue 464
ExtCtrls.TCellItem.Size 465
ExtCtrls.TCellItem.SizeStyle 465
ExtCtrls.TCellItem.Value 465
ExtCtrls.TCellSpan 1551
ExtCtrls.TCellSpan type 1551
ExtCtrls.TColorBox 466
ExtCtrls.TColorBox.Align 469
ExtCtrls.TColorBox.Anchors 472
ExtCtrls.TColorBox.AutoComplete 473
ExtCtrls.TColorBox.AutoDropDown 474
ExtCtrls.TColorBox.BevelEdges 474
ExtCtrls.TColorBox.BevelInner 475
ExtCtrls.TColorBox.BevelKind 475
ExtCtrls.TColorBox.BevelOuter 476
ExtCtrls.TColorBox.BiDiMode 476
ExtCtrls.TColorBox.Color 477
ExtCtrls.TColorBox.Constraints 477
ExtCtrls.TColorBox.Ctl3D 478
ExtCtrls.TColorBox.DefaultColorColor 479
ExtCtrls.TColorBox.DoubleBuffered 479
ExtCtrls.TColorBox.DropDownCount 481
ExtCtrls.TColorBox.Enabled 481
ExtCtrls.TColorBox.Font 482
ExtCtrls.TColorBox.ItemHeight 483
ExtCtrls.TColorBox.NoneColorColor 483
ExtCtrls.TColorBox.OnChange 497
ExtCtrls.TColorBox.OnClick 499
ExtCtrls.TColorBox.OnCloseUp 501
ExtCtrls.TColorBox.OnContextPopup 501
ExtCtrls.TColorBox.OnDblClick 502
ExtCtrls.TColorBox.OnDragDrop 502
ExtCtrls.TColorBox.OnDragOver 505
ExtCtrls.TColorBox.OnDropDown 508
ExtCtrls.TColorBox.OnEndDock 510
ExtCtrls.TColorBox.OnEndDrag 510
ExtCtrls.TColorBox.OnEnter 511
ExtCtrls.TColorBox.OnExit 514
ExtCtrls.TColorBox.OnGetColors 516
ExtCtrls.TColorBox.OnKeyDown 517
ExtCtrls.TColorBox.OnKeyPress 520
ExtCtrls.TColorBox.OnKeyUp 521
ExtCtrls.TColorBox.OnMouseEnter 523
ExtCtrls.TColorBox.OnMouseLeave 523
ExtCtrls.TColorBox.OnSelect 524
ExtCtrls.TColorBox.OnStartDock 524

ExtCtrls.TColorBox.OnStartDrag 525
ExtCtrls.TColorBox.ParentBiDiMode 484
ExtCtrls.TColorBox.ParentColor 484
ExtCtrls.TColorBox.ParentCtl3D 485
ExtCtrls.TColorBox.ParentDoubleBuffered 485
ExtCtrls.TColorBox.ParentFont 485
ExtCtrls.TColorBox.ParentShowHint 486
ExtCtrls.TColorBox.PopupMenu 486
ExtCtrls.TColorBox.Selected 491
ExtCtrls.TColorBox.ShowHint 492
ExtCtrls.TColorBox.Style 493
ExtCtrls.TColorBox.TabOrder 493
ExtCtrls.TColorBox.TabStop 494
ExtCtrls.TColorBox.Visible 494
ExtCtrls.TColorBoxStyle 1551
ExtCtrls.TColorBoxStyle type 1551
ExtCtrls.TColorBoxStyles 1541
ExtCtrls.TColorBoxStyles enumeration 1541
ExtCtrls.TColorListBox 525
ExtCtrls.TColorListBox.Align 528
ExtCtrls.TColorListBox.Anchors 532
ExtCtrls.TColorListBox.AutoComplete 532
ExtCtrls.TColorListBox.BevelEdges 533
ExtCtrls.TColorListBox.BevelInner 533
ExtCtrls.TColorListBox.BevelKind 533
ExtCtrls.TColorListBox.BevelOuter 534
ExtCtrls.TColorListBox.BiDiMode 534
ExtCtrls.TColorListBox.Color 535
ExtCtrls.TColorListBox.Constraints 535
ExtCtrls.TColorListBox.Ctl3D 536
ExtCtrls.TColorListBox.DefaultColorColor 536
ExtCtrls.TColorListBox.DoubleBuffered 536
ExtCtrls.TColorListBox.Enabled 538
ExtCtrls.TColorListBox.Font 539
ExtCtrls.TColorListBox.ItemHeight 540
ExtCtrls.TColorListBox.NoneColorColor 540
ExtCtrls.TColorListBox.OnClick 555
ExtCtrls.TColorListBox.OnContextPopup 557
ExtCtrls.TColorListBox.OnDblClick 558
ExtCtrls.TColorListBox.OnDragDrop 558
ExtCtrls.TColorListBox.OnDragOver 561
ExtCtrls.TColorListBox.OnEndDock 564
ExtCtrls.TColorListBox.OnEndDrag 565
ExtCtrls.TColorListBox.OnEnter 566
ExtCtrls.TColorListBox.OnExit 568
ExtCtrls.TColorListBox.OnGetColors 571
ExtCtrls.TColorListBox.OnKeyDown 571
ExtCtrls.TColorListBox.OnKeyPress 575
ExtCtrls.TColorListBox.OnKeyUp 576
ExtCtrls.TColorListBox.OnMouseActivate 577
ExtCtrls.TColorListBox.OnMouseDown 578
ExtCtrls.TColorListBox.OnMouseEnter 581
ExtCtrls.TColorListBox.OnMouseLeave 581
ExtCtrls.TColorListBox.OnMouseMove 581
ExtCtrls.TColorListBox.OnMouseUp 584
ExtCtrls.TColorListBox.OnStartDock 586
ExtCtrls.TColorListBox.OnStartDrag 586
ExtCtrls.TColorListBox.ParentBiDiMode 540
ExtCtrls.TColorListBox.ParentColor 541
ExtCtrls.TColorListBox.ParentCtl3D 541
ExtCtrls.TColorListBox.ParentDoubleBuffered 542
ExtCtrls.TColorListBox.ParentFont 542
ExtCtrls.TColorListBox.ParentShowHint 542
ExtCtrls.TColorListBox.PopupMenu 543
ExtCtrls.TColorListBox.Selected 548
ExtCtrls.TColorListBox.ShowHint 550
ExtCtrls.TColorListBox.Style 551
ExtCtrls.TColorListBox.TabOrder 552
ExtCtrls.TColorListBox.TabStop 553
ExtCtrls.TColorListBox.Visible 553
ExtCtrls.TColumnCollection 587
ExtCtrls.TColumnCollection.Add 589
ExtCtrls.TColumnCollection.Create 588
ExtCtrls.TColumnCollection.GetItemAttr 589
ExtCtrls.TColumnCollection.Notify 589
ExtCtrls.TColumnItem 590
ExtCtrls.TControlBar 590
ExtCtrls.TControlBar.Align 594

ExtCtrls.TControlBar.Anchors 597
ExtCtrls.TControlBar.AutoDock 598
ExtCtrls.TControlBar.AutoDrag 598
ExtCtrls.TControlBar.AutoSize 599
ExtCtrls.TControlBar.BevelEdges 599
ExtCtrls.TControlBar.BevelInner 600
ExtCtrls.TControlBar.BevelKind 600
ExtCtrls.TControlBar.BevelOuter 601
ExtCtrls.TControlBar.BevelWidth 601
ExtCtrls.TControlBar.BorderWidth 601
ExtCtrls.TControlBar.Canvas 602
ExtCtrls.TControlBar.Color 604
ExtCtrls.TControlBar.Constraints 604
ExtCtrls.TControlBar.CornerEdge 605
ExtCtrls.TControlBar.DockSite 605
ExtCtrls.TControlBar.DragCursor 606
ExtCtrls.TControlBar.DragKind 607
ExtCtrls.TControlBar.DragMode 607
ExtCtrls.TControlBar.DrawingStyle 609
ExtCtrls.TControlBar.Enabled 609
ExtCtrls.TControlBar.GradientDirection 609
ExtCtrls.TControlBar.GradientEndColor 609
ExtCtrls.TControlBar.GradientStartColor 610
ExtCtrls.TControlBar.OnAlignInsertBefore 623
ExtCtrls.TControlBar.OnAlignPosition 624
ExtCtrls.TControlBar.OnBandDrag 624
ExtCtrls.TControlBar.OnBandInfo 625
ExtCtrls.TControlBar.OnBandMove 625
ExtCtrls.TControlBar.OnBandPaint 626
ExtCtrls.TControlBar.OnBeginBandMove 626
ExtCtrls.TControlBar.OnCanResize 626
ExtCtrls.TControlBar.OnClick 627
ExtCtrls.TControlBar.OnConstrainedResize 628
ExtCtrls.TControlBar.OnContextPopup 629
ExtCtrls.TControlBar.OnDblClick 630
ExtCtrls.TControlBar.OnDockDrop 630
ExtCtrls.TControlBar.OnDockOver 632
ExtCtrls.TControlBar.OnDragDrop 634
ExtCtrls.TControlBar.OnDragOver 637
ExtCtrls.TControlBar.OnEndBandMove 640
ExtCtrls.TControlBar.OnEndDock 640
ExtCtrls.TControlBar.OnEndDrag 641
ExtCtrls.TControlBar.OnEnter 642
ExtCtrls.TControlBar.OnExit 644
ExtCtrls.TControlBar.OnGetSiteInfo 647
ExtCtrls.TControlBar.OnMouseActivate 649
ExtCtrls.TControlBar.OnMouseDown 650
ExtCtrls.TControlBar.OnMouseEnter 652
ExtCtrls.TControlBar.OnMouseLeave 653
ExtCtrls.TControlBar.OnMouseMove 653
ExtCtrls.TControlBar.OnMouseUp 655
ExtCtrls.TControlBar.OnPaint 657
ExtCtrls.TControlBar.OnResize 658
ExtCtrls.TControlBar.OnStartDock 659
ExtCtrls.TControlBar.OnStartDrag 659
ExtCtrls.TControlBar.OnUnDock 660
ExtCtrls.TControlBar.ParentBackground 610
ExtCtrls.TControlBar.ParentColor 610
ExtCtrls.TControlBar.ParentCtl3D 611
ExtCtrls.TControlBar.ParentFont 611
ExtCtrls.TControlBar.ParentShowHint 612
ExtCtrls.TControlBar.Picture 612
ExtCtrls.TControlBar.PopupMenu 612
ExtCtrls.TControlBar.RowSize 617
ExtCtrls.TControlBar.RowSnap 618
ExtCtrls.TControlBar.ShowHint 618
ExtCtrls.TControlBar.TabOrder 620
ExtCtrls.TControlBar.TabStop 620
ExtCtrls.TControlBar.Visible 621
ExtCtrls.TControlCollection 663
ExtCtrls.TControlCollection.Add 664
ExtCtrls.TControlCollection.AddControl 665
ExtCtrls.TControlCollection.ControlItems 667
ExtCtrls.TControlCollection.Controls 668
ExtCtrls.TControlCollection.Create 664
ExtCtrls.TControlCollection.GetControl 665
ExtCtrls.TControlCollection.GetControlItem 665
ExtCtrls.TControlCollection.GetItem 665

ExtCtrls.TControlCollection.IndexOf 666
ExtCtrls.TControlCollection.Items 668
ExtCtrls.TControlCollection.Owner 666
ExtCtrls.TControlCollection.RemoveControl 666
ExtCtrls.TControlCollection.SetControl 666
ExtCtrls.TControlCollection.SetItem 666
ExtCtrls.TControlCollection.Update 667
ExtCtrls.TControlItem 668
ExtCtrls.TControlItem.AssignTo 670
ExtCtrls.TControlItem.Column 672
ExtCtrls.TControlItem.ColumnSpan 672
ExtCtrls.TControlItem.Control 672
ExtCtrls.TControlItem.Create 669
ExtCtrls.TControlItem.Destroy 669
ExtCtrls.TControlItem.GridPanel 672
ExtCtrls.TControlItem.InternalSetLocation 671
ExtCtrls.TControlItem.Pushed 672
ExtCtrls.TControlItem.Row 673
ExtCtrls.TControlItem.RowSpan 673
ExtCtrls.TControlItem.SetLocation 671
ExtCtrls.TCornerEdge 1542
ExtCtrls.TCornerEdge enumeration 1542
ExtCtrls.TCustomButtonedEdit 673
ExtCtrls.TCustomButtonedEdit.Create 674
ExtCtrls.TCustomButtonedEdit.DefaultHandler 675
ExtCtrls.TCustomButtonedEdit.Destroy 675
ExtCtrls.TCustomButtonedEdit.DoSetTextHint 676
ExtCtrls.TCustomButtonedEdit.GetEditButtonClass 676
ExtCtrls.TCustomButtonedEdit.Images 682
ExtCtrls.TCustomButtonedEdit.LeftButton 685
ExtCtrls.TCustomButtonedEdit.Notification 676
ExtCtrls.TCustomButtonedEdit.OnLeftButtonClick 692
ExtCtrls.TCustomButtonedEdit.OnRightButtonClick 695
ExtCtrls.TCustomButtonedEdit.RightButton 688
ExtCtrls.TCustomButtonedEdit.UpdateEditMargins 677
ExtCtrls.TCustomButtonedEdit.WndProc 677
ExtCtrls.TCustomCategoryPanel 698
ExtCtrls.TCustomCategoryPanel.CalcRectBounds 703
ExtCtrls.TCustomCategoryPanel.Collapse 703
ExtCtrls.TCustomCategoryPanel.Collapsed 714
ExtCtrls.TCustomCategoryPanel.CollapsedHotImageIndex 714
ExtCtrls.TCustomCategoryPanel.CollapsedImageIndex 715
ExtCtrls.TCustomCategoryPanel.CollapsedPressedImageIndex 715
ExtCtrls.TCustomCategoryPanel.Create 700
ExtCtrls.TCustomCategoryPanel.DefineProperties 703
ExtCtrls.TCustomCategoryPanel.Destroy 700
ExtCtrls.TCustomCategoryPanel.DrawCollapsedPanel 704
ExtCtrls.TCustomCategoryPanel.DrawHeader 704
ExtCtrls.TCustomCategoryPanel.DrawHeaderBackground 705
ExtCtrls.TCustomCategoryPanel.DrawHeaderCaption 705
ExtCtrls.TCustomCategoryPanel.DrawHeaderChevron 705
ExtCtrls.TCustomCategoryPanel.Expand 705
ExtCtrls.TCustomCategoryPanel.ExpandedHotImageIndex 716
ExtCtrls.TCustomCategoryPanel.ExpandedImageIndex 716
ExtCtrls.TCustomCategoryPanel.ExpandedPressedImageIndex 716
ExtCtrls.TCustomCategoryPanel.FPanelSurface 702
ExtCtrls.TCustomCategoryPanel.GetChildren 706
ExtCtrls.TCustomCategoryPanel.GetCollapsedHeight 706
ExtCtrls.TCustomCategoryPanel.GetTabOrderList 706
ExtCtrls.TCustomCategoryPanel.HeaderHeight 717
ExtCtrls.TCustomCategoryPanel.HeaderState 717
ExtCtrls.TCustomCategoryPanel.Loaded 707
ExtCtrls.TCustomCategoryPanel.OnCollapse 718
ExtCtrls.TCustomCategoryPanel.OnExpand 718
ExtCtrls.TCustomCategoryPanel.PanelGroup 717
ExtCtrls.TCustomCategoryPanel.ReadState 707
ExtCtrls.TCustomCategoryPanel.ResizeHeader 708
ExtCtrls.TCustomCategoryPanel.SetBounds 708
ExtCtrls.TCustomCategoryPanel.SetParent 708
ExtCtrls.TCustomCategoryPanel.THeaderState 701
ExtCtrls.TCustomCategoryPanel.THeaderState enumeration 701
ExtCtrls.TCustomCategoryPanel.UpdateControlOriginalParentSize 709
ExtCtrls.TCustomCategoryPanel.UpdateHeader 709
ExtCtrls.TCustomCategoryPanel.WndProc 709
ExtCtrls.TCustomCategoryPanelGroup 718

ExtCtrls.TCustomCategoryPanelGroup.ChevronAlignment 730
ExtCtrls.TCustomCategoryPanelGroup.ChevronColor 730
ExtCtrls.TCustomCategoryPanelGroup.ChevronHotColor 731
ExtCtrls.TCustomCategoryPanelGroup.CollapseAll 722
ExtCtrls.TCustomCategoryPanelGroup.Color 731
ExtCtrls.TCustomCategoryPanelGroup.Create 720
ExtCtrls.TCustomCategoryPanelGroup.CreatePanel 722
ExtCtrls.TCustomCategoryPanelGroup.CreateWnd 723
ExtCtrls.TCustomCategoryPanelGroup.Destroy 721
ExtCtrls.TCustomCategoryPanelGroup.ExpandAll 723
ExtCtrls.TCustomCategoryPanelGroup.GetCategoryPanelClass 723
ExtCtrls.TCustomCategoryPanelGroup.GradientBaseColor 732
ExtCtrls.TCustomCategoryPanelGroup.GradientColor 732
ExtCtrls.TCustomCategoryPanelGroup.GradientDirection 733
ExtCtrls.TCustomCategoryPanelGroup.HeaderAlignment 733
ExtCtrls.TCustomCategoryPanelGroup.HeaderFont 733
ExtCtrls.TCustomCategoryPanelGroup.HeaderHeight 734
ExtCtrls.TCustomCategoryPanelGroup.HeaderImage 734
ExtCtrls.TCustomCategoryPanelGroup.HeaderStyle 734
ExtCtrls.TCustomCategoryPanelGroup.Images 735
ExtCtrls.TCustomCategoryPanelGroup.Loaded 724
ExtCtrls.TCustomCategoryPanelGroup.Notification 724
ExtCtrls.TCustomCategoryPanelGroup.Panels 735
ExtCtrls.TCustomCategoryPanelGroup.ReorderPanels 725
ExtCtrls.TCustomCategoryPanelGroup.THeaderStyle 721
ExtCtrls.TCustomCategoryPanelGroup.THeaderStyle enumeration 721
ExtCtrls.TCustomCategoryPanelGroup.WndProc 725
ExtCtrls.TCustomColorBox 735
ExtCtrls.TCustomColorBox.CloseUp 737
ExtCtrls.TCustomColorBox.ColorNames 741
ExtCtrls.TCustomColorBox.Colors 742
ExtCtrls.TCustomColorBox.Create 737
ExtCtrls.TCustomColorBox.CreateWnd 738
ExtCtrls.TCustomColorBox.DefaultColorColor 742
ExtCtrls.TCustomColorBox.DrawItem 738
ExtCtrls.TCustomColorBox.KeyDown 738
ExtCtrls.TCustomColorBox.KeyPress 739
ExtCtrls.TCustomColorBox.Loaded 739
ExtCtrls.TCustomColorBox.NoneColorColor 743
ExtCtrls.TCustomColorBox.OnGetColors 744
ExtCtrls.TCustomColorBox.PickCustomColor 740
ExtCtrls.TCustomColorBox.PopulateList 740
ExtCtrls.TCustomColorBox.Select 740
ExtCtrls.TCustomColorBox.Selected 743
ExtCtrls.TCustomColorBox.SetStyle 741
ExtCtrls.TCustomColorBox.Style 744
ExtCtrls.TCustomColorListBox 745
ExtCtrls.TCustomColorListBox.ColorNames 750
ExtCtrls.TCustomColorListBox.Colors 750
ExtCtrls.TCustomColorListBox.Create 746
ExtCtrls.TCustomColorListBox.CreateWnd 746
ExtCtrls.TCustomColorListBox.DefaultColorColor 750
ExtCtrls.TCustomColorListBox.DrawItem 747
ExtCtrls.TCustomColorListBox.KeyDown 747
ExtCtrls.TCustomColorListBox.KeyPress 748
ExtCtrls.TCustomColorListBox.Loaded 748
ExtCtrls.TCustomColorListBox.NoneColorColor 750
ExtCtrls.TCustomColorListBox.OnGetColors 753
ExtCtrls.TCustomColorListBox.PickCustomColor 749
ExtCtrls.TCustomColorListBox.PopulateList 749
ExtCtrls.TCustomColorListBox.Selected 751
ExtCtrls.TCustomColorListBox.SetStyle 749
ExtCtrls.TCustomColorListBox.Style 753
ExtCtrls.TCustomControlBar 754
ExtCtrls.TCustomControlBar.AlignControls 757
ExtCtrls.TCustomControlBar.AutoDock 766
ExtCtrls.TCustomControlBar.AutoDrag 766
ExtCtrls.TCustomControlBar.AutoSize 767
ExtCtrls.TCustomControlBar.BevelKind 767
ExtCtrls.TCustomControlBar.CanAutoSize 757
ExtCtrls.TCustomControlBar.CornerEdge 768
ExtCtrls.TCustomControlBar.Create 755
ExtCtrls.TCustomControlBar.CreateParams 758
ExtCtrls.TCustomControlBar.Destroy 756
ExtCtrls.TCustomControlBar.DoBandMove 758
ExtCtrls.TCustomControlBar.DoBandPaint 758
ExtCtrls.TCustomControlBar.DoBeginBandMove 759

ExtCtrls.TCustomControlBar.DockOver 759
ExtCtrls.TCustomControlBar.DockSite 768
ExtCtrls.TCustomControlBar.DoEndBandMove 759
ExtCtrls.TCustomControlBar.DoPaletteChange 760
ExtCtrls.TCustomControlBar.DragControl 760
ExtCtrls.TCustomControlBar.DrawingStyle 769
ExtCtrls.TCustomControlBar.FlipChildren 761
ExtCtrls.TCustomControlBar.GetControlInfo 761
ExtCtrls.TCustomControlBar.GetPalette 762
ExtCtrls.TCustomControlBar.GetSiteInfo 762
ExtCtrls.TCustomControlBar.GradientDirection 769
ExtCtrls.TCustomControlBar.GradientEndColor 769
ExtCtrls.TCustomControlBar.GradientStartColor 769
ExtCtrls.TCustomControlBar.HitTest 763
ExtCtrls.TCustomControlBar.MouseDown 763
ExtCtrls.TCustomControlBar.MouseMove 763
ExtCtrls.TCustomControlBar.MouseUp 764
ExtCtrls.TCustomControlBar.OnBandDrag 771
ExtCtrls.TCustomControlBar.OnBandInfo 771
ExtCtrls.TCustomControlBar.OnBandMove 772
ExtCtrls.TCustomControlBar.OnBandPaint 772
ExtCtrls.TCustomControlBar.OnBeginBandMove 773
ExtCtrls.TCustomControlBar.OnEndBandMove 773
ExtCtrls.TCustomControlBar.OnPaint 773
ExtCtrls.TCustomControlBar.Paint 764
ExtCtrls.TCustomControlBar.PaintControlFrame 765
ExtCtrls.TCustomControlBar.Picture 770
ExtCtrls.TCustomControlBar.RowSize 770
ExtCtrls.TCustomControlBar.RowSnap 770
ExtCtrls.TCustomControlBar.StickControls 765
ExtCtrls.TCustomFlowPanel 774
ExtCtrls.TCustomFlowPanel.AlignControls 776
ExtCtrls.TCustomFlowPanel.AutoWrap 778
ExtCtrls.TCustomFlowPanel.Create 775
ExtCtrls.TCustomFlowPanel.Destroy 775
ExtCtrls.TCustomFlowPanel.FlowStyle 778
ExtCtrls.TCustomFlowPanel.GetChildren 776
ExtCtrls.TCustomFlowPanel.GetControlIndex 777
ExtCtrls.TCustomFlowPanel.SetControlIndex 777
ExtCtrls.TCustomGridPanel 781
ExtCtrls.TCustomGridPanel.AlignControls 784
ExtCtrls.TCustomGridPanel.AutoAddColumn 785
ExtCtrls.TCustomGridPanel.AutoAddRow 785
ExtCtrls.TCustomGridPanel.CellCount 788
ExtCtrls.TCustomGridPanel.CellIndexToCell 785
ExtCtrls.TCustomGridPanel.CellRect 788
ExtCtrls.TCustomGridPanel.CellSize 789
ExtCtrls.TCustomGridPanel.CellToCellIndex 785
ExtCtrls.TCustomGridPanel.ColumnCollection 789
ExtCtrls.TCustomGridPanel.ColumnSpanIndex 789
ExtCtrls.TCustomGridPanel.ControlCollection 789
ExtCtrls.TCustomGridPanel.Create 783
ExtCtrls.TCustomGridPanel.Destroy 783
ExtCtrls.TCustomGridPanel.ExpandStyle 790
ExtCtrls.TCustomGridPanel.GetChildren 785
ExtCtrls.TCustomGridPanel.IsColumnEmpty 786
ExtCtrls.TCustomGridPanel.IsRowEmpty 786
ExtCtrls.TCustomGridPanel.Loaded 786
ExtCtrls.TCustomGridPanel.Paint 787
ExtCtrls.TCustomGridPanel.RemoveEmptyAutoAddColumns 787
ExtCtrls.TCustomGridPanel.RemoveEmptyAutoAddRows 787
ExtCtrls.TCustomGridPanel.RowCollection 790
ExtCtrls.TCustomGridPanel.RowSpanIndex 790
ExtCtrls.TCustomGridPanel.UpdateControlOriginalParentSize 788
ExtCtrls.TCustomLabeledEdit 790
ExtCtrls.TCustomLabeledEdit.CMBidimodechanged 792
ExtCtrls.TCustomLabeledEdit.CMEnabledchanged 793
ExtCtrls.TCustomLabeledEdit.CMVisiblechanged 793
ExtCtrls.TCustomLabeledEdit.Create 791
ExtCtrls.TCustomLabeledEdit.EditLabel 796
ExtCtrls.TCustomLabeledEdit.LabelPosition 796
ExtCtrls.TCustomLabeledEdit.LabelSpacing 796
ExtCtrls.TCustomLabeledEdit.Notification 793
ExtCtrls.TCustomLabeledEdit.SetBounds 794
ExtCtrls.TCustomLabeledEdit.SetName 794
ExtCtrls.TCustomLabeledEdit.SetParent 795
ExtCtrls.TCustomLabeledEdit.SetupInternalLabel 795

ExtCtrls.TCustomLinkLabel 797
ExtCtrls.TCustomLinkLabel.Alignment 800
ExtCtrls.TCustomLinkLabel.AutoSize 800
ExtCtrls.TCustomLinkLabel.Create 798
ExtCtrls.TCustomLinkLabel.CreateParams 799
ExtCtrls.TCustomLinkLabel.CreateWnd 799
ExtCtrls.TCustomLinkLabel.OnLinkClick 802
ExtCtrls.TCustomLinkLabel.SetAutoSize 800
ExtCtrls.TCustomLinkLabel.TLinkAlignment 801
ExtCtrls.TCustomLinkLabel.UseThemes 800
ExtCtrls.TCustomLinkLabel.UseVisualStyle 801
ExtCtrls.TCustomPanel 802
ExtCtrls.TCustomPanel.AdjustClientRect 804
ExtCtrls.TCustomPanel.Alignment 807
ExtCtrls.TCustomPanel.BevelInner 807
ExtCtrls.TCustomPanel.BevelOuter 808
ExtCtrls.TCustomPanel.BevelWidth 809
ExtCtrls.TCustomPanel.BorderStyle 810
ExtCtrls.TCustomPanel.BorderWidth 811
ExtCtrls.TCustomPanel.CanAutoSize 804
ExtCtrls.TCustomPanel.Color 812
ExtCtrls.TCustomPanel.Create 803
ExtCtrls.TCustomPanel.CreateParams 805
ExtCtrls.TCustomPanel.FullRepaint 813
ExtCtrls.TCustomPanel.GetControlsAlignment 805
ExtCtrls.TCustomPanel.Locked 813
ExtCtrls.TCustomPanel.Paint 806
ExtCtrls.TCustomPanel.ParentBackground 813
ExtCtrls.TCustomPanel.ParentColor 814
ExtCtrls.TCustomPanel.SetParentBackground 806
ExtCtrls.TCustomPanel.ShowCaption 814
ExtCtrls.TCustomPanel.VerticalAlignment 814
ExtCtrls.TCustomRadioGroup 815
ExtCtrls.TCustomRadioGroup.Buttons 819
ExtCtrls.TCustomRadioGroup.CanModify 817
ExtCtrls.TCustomRadioGroup.Columns 820
ExtCtrls.TCustomRadioGroup.Create 816
ExtCtrls.TCustomRadioGroup.Destroy 816
ExtCtrls.TCustomRadioGroup.FlipChildren 817
ExtCtrls.TCustomRadioGroup.GetChildren 818
ExtCtrls.TCustomRadioGroup.ItemIndex 820
ExtCtrls.TCustomRadioGroup.Items 821
ExtCtrls.TCustomRadioGroup.Loaded 818
ExtCtrls.TCustomRadioGroup.ReadState 819
ExtCtrls.TCustomRadioGroup.WordWrap 822
ExtCtrls.TCustomTrayIcon 822
ExtCtrls.TCustomTrayIcon.Animate 830
ExtCtrls.TCustomTrayIcon.AnimateInterval 830
ExtCtrls.TCustomTrayIcon.BalloonFlags 831
ExtCtrls.TCustomTrayIcon.BalloonHint 831
ExtCtrls.TCustomTrayIcon.BalloonTimeout 831
ExtCtrls.TCustomTrayIcon.BalloonTitle 831
ExtCtrls.TCustomTrayIcon.Create 824
ExtCtrls.TCustomTrayIcon.Data 832
ExtCtrls.TCustomTrayIcon.Destroy 824
ExtCtrls.TCustomTrayIcon.DoOnAnimate 825
ExtCtrls.TCustomTrayIcon.GetAnimateInterval 825
ExtCtrls.TCustomTrayIcon.GetBalloonTimeout 825
ExtCtrls.TCustomTrayIcon.Hint 832
ExtCtrls.TCustomTrayIcon.Icon 832
ExtCtrls.TCustomTrayIcon.IconIndex 832
ExtCtrls.TCustomTrayIcon.Icons 833
ExtCtrls.TCustomTrayIcon.Notification 826
ExtCtrls.TCustomTrayIcon.OnAnimate 834
ExtCtrls.TCustomTrayIcon.OnBalloonClick 834
ExtCtrls.TCustomTrayIcon.OnClick 834
ExtCtrls.TCustomTrayIcon.OnDblClick 834
ExtCtrls.TCustomTrayIcon.OnMouseDown 835
ExtCtrls.TCustomTrayIcon.OnMouseMove 835
ExtCtrls.TCustomTrayIcon.OnMouseUp 835
ExtCtrls.TCustomTrayIcon.PopupMenu 833
ExtCtrls.TCustomTrayIcon.Refresh 826
ExtCtrls.TCustomTrayIcon.SetAnimate 827
ExtCtrls.TCustomTrayIcon.SetAnimateInterval 827
ExtCtrls.TCustomTrayIcon.SetBalloonHint 827
ExtCtrls.TCustomTrayIcon.SetBalloonTimeout 827
ExtCtrls.TCustomTrayIcon.SetBalloonTitle 828
ExtCtrls.TCustomTrayIcon.SetDefaultIcon 828

ExtCtrls.TCustomTrayIcon.SetHint 828
ExtCtrls.TCustomTrayIcon.SetIcon 828
ExtCtrls.TCustomTrayIcon.SetIconIndex 828
ExtCtrls.TCustomTrayIcon.SetIconList 829
ExtCtrls.TCustomTrayIcon.SetVisible 829
ExtCtrls.TCustomTrayIcon.ShowBalloonHint 829
ExtCtrls.TCustomTrayIcon.Visible 833
ExtCtrls.TCustomTrayIcon.WindowProc 829
ExtCtrls.TEditButton 835
ExtCtrls.TEditButton.Create 839
ExtCtrls.TEditButton.Destroy 840
ExtCtrls.TEditButton.DisabledImageIndex 842
ExtCtrls.TEditButton.DropDownMenu 846
ExtCtrls.TEditButton.EditControl 846
ExtCtrls.TEditButton.Enabled 846
ExtCtrls.TEditButton.GetOwner 841
ExtCtrls.TEditButton.Glyph 849
ExtCtrls.TEditButton.HotImageIndex 850
ExtCtrls.TEditButton.ImageIndex 853
ExtCtrls.TEditButton.Images 856
ExtCtrls.TEditButton.Position 856
ExtCtrls.TEditButton.PressedImageIndex 856
ExtCtrls.TEditButton.TButtonPosition 840
ExtCtrls.TEditButton.TButtonPosition enumeration 840
ExtCtrls.TEditButton.UpdateBounds 842
ExtCtrls.TEditButton.Visible 857
ExtCtrls.TEditButtonClass 1552
ExtCtrls.TEditButtonClass type 1552
ExtCtrls.TEndBandMoveEvent 1552
ExtCtrls.TEndBandMoveEvent type 1552
ExtCtrls.TExpandStyle 1542
ExtCtrls.TExpandStyle enumeration 1542
ExtCtrls.TFlowPanel 857
ExtCtrls.TFlowPanel.Align 863
ExtCtrls.TFlowPanel.Alignment 867
ExtCtrls.TFlowPanel.Anchors 867
ExtCtrls.TFlowPanel.AutoSize 868
ExtCtrls.TFlowPanel.AutoWrap 868
ExtCtrls.TFlowPanel.BevelEdges 868
ExtCtrls.TFlowPanel.BevelInner 869
ExtCtrls.TFlowPanel.BevelKind 870
ExtCtrls.TFlowPanel.BevelOuter 870
ExtCtrls.TFlowPanel.BevelWidth 871
ExtCtrls.TFlowPanel.BiDiMode 872
ExtCtrls.TFlowPanel.BorderStyle 873
ExtCtrls.TFlowPanel.BorderWidth 873
ExtCtrls.TFlowPanel.Caption 874
ExtCtrls.TFlowPanel.Color 877
ExtCtrls.TFlowPanel.Constraints 878
ExtCtrls.TFlowPanel.Ctl3D 878
ExtCtrls.TFlowPanel.DockSite 879
ExtCtrls.TFlowPanel.DoubleBuffered 880
ExtCtrls.TFlowPanel.DragCursor 882
ExtCtrls.TFlowPanel.DragKind 882
ExtCtrls.TFlowPanel.DragMode 882
ExtCtrls.TFlowPanel.Enabled 884
ExtCtrls.TFlowPanel.FlowStyle 884
ExtCtrls.TFlowPanel.Font 888
ExtCtrls.TFlowPanel.FullRepaint 889
ExtCtrls.TFlowPanel.Locked 889
ExtCtrls.TFlowPanel.OnAlignInsertBefore 903
ExtCtrls.TFlowPanel.OnAlignPosition 904
ExtCtrls.TFlowPanel.OnCanResize 905
ExtCtrls.TFlowPanel.OnClick 905
ExtCtrls.TFlowPanel.OnConstrainedResize 907
ExtCtrls.TFlowPanel.OnContextPopup 907
ExtCtrls.TFlowPanel.OnDblClick 908
ExtCtrls.TFlowPanel.OnDockDrop 908
ExtCtrls.TFlowPanel.OnDockOver 910
ExtCtrls.TFlowPanel.OnDragDrop 913
ExtCtrls.TFlowPanel.OnDragOver 916
ExtCtrls.TFlowPanel.OnEndDock 919
ExtCtrls.TFlowPanel.OnEndDrag 919
ExtCtrls.TFlowPanel.OnEnter 920
ExtCtrls.TFlowPanel.OnExit 923
ExtCtrls.TFlowPanel.OnGetSiteInfo 925
ExtCtrls.TFlowPanel.OnMouseActivate 928
ExtCtrls.TFlowPanel.OnMouseDown 929

ExtCtrls.TFlowPanel.OnMouseEnter 931
ExtCtrls.TFlowPanel.OnMouseLeave 931
ExtCtrls.TFlowPanel.OnMouseMove 931
ExtCtrls.TFlowPanel.OnMouseUp 934
ExtCtrls.TFlowPanel.OnResize 936
ExtCtrls.TFlowPanel.OnStartDock 937
ExtCtrls.TFlowPanel.OnStartDrag 937
ExtCtrls.TFlowPanel.OnUnDock 938
ExtCtrls.TFlowPanel.Padding 889
ExtCtrls.TFlowPanel.ParentBackground 890
ExtCtrls.TFlowPanel.ParentBiDiMode 890
ExtCtrls.TFlowPanel.ParentColor 890
ExtCtrls.TFlowPanel.ParentCtl3D 891
ExtCtrls.TFlowPanel.ParentDoubleBuffered 891
ExtCtrls.TFlowPanel.ParentFont 891
ExtCtrls.TFlowPanel.ParentShowHint 892
ExtCtrls.TFlowPanel.PopupMenu 893
ExtCtrls.TFlowPanel.ShowCaption 898
ExtCtrls.TFlowPanel.ShowHint 898
ExtCtrls.TFlowPanel.TabOrder 899
ExtCtrls.TFlowPanel.TabStop 900
ExtCtrls.TFlowPanel.UseDockManager 900
ExtCtrls.TFlowPanel.VerticalAlignment 901
ExtCtrls.TFlowPanel.Visible 901
ExtCtrls.TFlowStyle 1543
ExtCtrls.TFlowStyle enumeration 1543
ExtCtrls.TGetColorsEvent 1552
ExtCtrls.TGetColorsEvent type 1552
ExtCtrls.TGridPanel 941
ExtCtrls.TGridPanel.Align 945
ExtCtrls.TGridPanel.Alignment 948
ExtCtrls.TGridPanel.Anchors 949
ExtCtrls.TGridPanel.BevelEdges 949
ExtCtrls.TGridPanel.BevelInner 950
ExtCtrls.TGridPanel.BevelKind 951
ExtCtrls.TGridPanel.BevelOuter 951
ExtCtrls.TGridPanel.BevelWidth 952
ExtCtrls.TGridPanel.BiDiMode 953
ExtCtrls.TGridPanel.BorderStyle 954
ExtCtrls.TGridPanel.BorderWidth 954
ExtCtrls.TGridPanel.Caption 955
ExtCtrls.TGridPanel.Color 958
ExtCtrls.TGridPanel.ColumnCollection 959
ExtCtrls.TGridPanel.Constraints 959
ExtCtrls.TGridPanel.ControlCollection 959
ExtCtrls.TGridPanel.Ctl3D 960
ExtCtrls.TGridPanel.DockManager 960
ExtCtrls.TGridPanel.DockSite 962
ExtCtrls.TGridPanel.DoubleBuffered 963
ExtCtrls.TGridPanel.DragCursor 965
ExtCtrls.TGridPanel.DragKind 965
ExtCtrls.TGridPanel.DragMode 966
ExtCtrls.TGridPanel.Enabled 967
ExtCtrls.TGridPanel.ExpandStyle 968
ExtCtrls.TGridPanel.Font 968
ExtCtrls.TGridPanel.FullRepaint 969
ExtCtrls.TGridPanel.Locked 969
ExtCtrls.TGridPanel.OnAlignInsertBefore 984
ExtCtrls.TGridPanel.OnAlignPosition 984
ExtCtrls.TGridPanel.OnCanResize 985
ExtCtrls.TGridPanel.OnClick 986
ExtCtrls.TGridPanel.OnConstrainedResize 987
ExtCtrls.TGridPanel.OnContextPopup 988
ExtCtrls.TGridPanel.OnDblClick 989
ExtCtrls.TGridPanel.OnDockDrop 989
ExtCtrls.TGridPanel.OnDockOver 991
ExtCtrls.TGridPanel.OnDragDrop 993
ExtCtrls.TGridPanel.OnDragOver 996
ExtCtrls.TGridPanel.OnEndDock 999
ExtCtrls.TGridPanel.OnEndDrag 1000
ExtCtrls.TGridPanel.OnEnter 1001
ExtCtrls.TGridPanel.OnExit 1003
ExtCtrls.TGridPanel.OnGetSiteInfo 1006
ExtCtrls.TGridPanel.OnMouseActivate 1008
ExtCtrls.TGridPanel.OnMouseDown 1009
ExtCtrls.TGridPanel.OnMouseEnter 1011
ExtCtrls.TGridPanel.OnMouseLeave 1011
ExtCtrls.TGridPanel.OnMouseMove 1012

ExtCtrls.TGridPanel.OnMouseUp 1014
ExtCtrls.TGridPanel.OnResize 1016
ExtCtrls.TGridPanel.OnStartDock 1017
ExtCtrls.TGridPanel.OnStartDrag 1018
ExtCtrls.TGridPanel.OnUnDock 1018
ExtCtrls.TGridPanel.Padding 970
ExtCtrls.TGridPanel.ParentBackground 970
ExtCtrls.TGridPanel.ParentBiDiMode 970
ExtCtrls.TGridPanel.ParentColor 971
ExtCtrls.TGridPanel.ParentCtl3D 971
ExtCtrls.TGridPanel.ParentDoubleBuffered 972
ExtCtrls.TGridPanel.ParentFont 972
ExtCtrls.TGridPanel.ParentShowHint 972
ExtCtrls.TGridPanel.PopupMenu 973
ExtCtrls.TGridPanel.RowCollection 978
ExtCtrls.TGridPanel.ShowCaption 978
ExtCtrls.TGridPanel.ShowHint 978
ExtCtrls.TGridPanel.TabOrder 980
ExtCtrls.TGridPanel.TabStop 980
ExtCtrls.TGridPanel.UseDockManager 981
ExtCtrls.TGridPanel.VerticalAlignment 981
ExtCtrls.TGridPanel.Visible 982
ExtCtrls.THeader 1021
ExtCtrls.THeader.Align 1027
ExtCtrls.THeader.AllowResize 1030
ExtCtrls.THeader.Anchors 1030
ExtCtrls.THeader.BorderStyle 1031
ExtCtrls.THeader.Constraints 1031
ExtCtrls.THeader.Create 1022
ExtCtrls.THeader.CreateParams 1023
ExtCtrls.THeader.Destroy 1023
ExtCtrls.THeader.DoubleBuffered 1032
ExtCtrls.THeader.Enabled 1034
ExtCtrls.THeader.Font 1034
ExtCtrls.THeader.MouseDown 1024
ExtCtrls.THeader.MouseMove 1024
ExtCtrls.THeader.MouseUp 1025
ExtCtrls.THeader.OnContextPopup 1046
ExtCtrls.THeader.OnSized 1047
ExtCtrls.THeader.OnSizing 1047
ExtCtrls.THeader.Paint 1025
ExtCtrls.THeader.ParentDoubleBuffered 1035
ExtCtrls.THeader.ParentFont 1036
ExtCtrls.THeader.ParentShowHint 1036
ExtCtrls.THeader.PopupMenu 1037
ExtCtrls.THeader.Sections 1042
ExtCtrls.THeader.SectionWidth 1042
ExtCtrls.THeader.ShowHint 1042
ExtCtrls.THeader.Sized 1026
ExtCtrls.THeader.Sizing 1026
ExtCtrls.THeader.TabOrder 1044
ExtCtrls.THeader.TabStop 1044
ExtCtrls.THeader.Visible 1045
ExtCtrls.TImage 1048
ExtCtrls.TImage.Align 1054
ExtCtrls.TImage.Anchors 1057
ExtCtrls.TImage.AutoSize 1058
ExtCtrls.TImage.CanAutoSize 1051
ExtCtrls.TImage.Canvas 1058
ExtCtrls.TImage.Center 1061
ExtCtrls.TImage.Constraints 1062
ExtCtrls.TImage.Create 1050
ExtCtrls.TImage.DestRect 1052
ExtCtrls.TImage.Destroy 1050
ExtCtrls.TImage.DoPaletteChange 1052
ExtCtrls.TImage.DragCursor 1062
ExtCtrls.TImage.DragKind 1063
ExtCtrls.TImage.DragMode 1063
ExtCtrls.TImage.Enabled 1065
ExtCtrls.TImage.GetPalette 1052
ExtCtrls.TImage.IncrementalDisplay 1065
ExtCtrls.TImage.OnClick 1079
ExtCtrls.TImage.OnContextPopup 1081
ExtCtrls.TImage.OnDblClick 1082
ExtCtrls.TImage.OnDragDrop 1082
ExtCtrls.TImage.OnDragOver 1085
ExtCtrls.TImage.OnEndDock 1088
ExtCtrls.TImage.OnEndDrag 1089

ExtCtrls.TImage.OnMouseActivate 1090
ExtCtrls.TImage.OnMouseDown 1091
ExtCtrls.TImage.OnMouseEnter 1093
ExtCtrls.TImage.OnMouseLeave 1093
ExtCtrls.TImage.OnMouseMove 1093
ExtCtrls.TImage.OnMouseUp 1096
ExtCtrls.TImage.OnProgress 1098
ExtCtrls.TImage.OnStartDock 1098
ExtCtrls.TImage.OnStartDrag 1099
ExtCtrls.TImage.Paint 1052
ExtCtrls.TImage.ParentShowHint 1066
ExtCtrls.TImage.Picture 1066
ExtCtrls.TImage.PopupMenu 1068
ExtCtrls.TImage.Progress 1053
ExtCtrls.TImage.Proportional 1073
ExtCtrls.TImage.ShowHint 1073
ExtCtrls.TImage.Stretch 1075
ExtCtrls.TImage.Transparent 1077
ExtCtrls.TImage.Visible 1077
ExtCtrls.TLabeledEdit 1100
ExtCtrls.TLabeledEdit.Alignment 1103
ExtCtrls.TLabeledEdit.Anchors 1104
ExtCtrls.TLabeledEdit.AutoSelect 1104
ExtCtrls.TLabeledEdit.AutoSize 1105
ExtCtrls.TLabeledEdit.BevelEdges 1106
ExtCtrls.TLabeledEdit.BevelInner 1106
ExtCtrls.TLabeledEdit.BevelKind 1107
ExtCtrls.TLabeledEdit.BevelOuter 1107
ExtCtrls.TLabeledEdit.BiDiMode 1107
ExtCtrls.TLabeledEdit.BorderStyle 1108
ExtCtrls.TLabeledEdit.CharCase 1108
ExtCtrls.TLabeledEdit.Color 1109
ExtCtrls.TLabeledEdit.Constraints 1109
ExtCtrls.TLabeledEdit.Ctl3D 1110
ExtCtrls.TLabeledEdit.DoubleBuffered 1110
ExtCtrls.TLabeledEdit.DragCursor 1112
ExtCtrls.TLabeledEdit.DragKind 1113
ExtCtrls.TLabeledEdit.DragMode 1113
ExtCtrls.TLabeledEdit.EditLabel 1115
ExtCtrls.TLabeledEdit.Enabled 1115
ExtCtrls.TLabeledEdit.Font 1115
ExtCtrls.TLabeledEdit.HideSelection 1116
ExtCtrls.TLabeledEdit.ImeMode 1117
ExtCtrls.TLabeledEdit.ImeName 1117
ExtCtrls.TLabeledEdit.LabelPosition 1118
ExtCtrls.TLabeledEdit.LabelSpacing 1118
ExtCtrls.TLabeledEdit.MaxLength 1119
ExtCtrls.TLabeledEdit.NumbersOnly 1119
ExtCtrls.TLabeledEdit.OEMConvert 1120
ExtCtrls.TLabeledEdit.OnChange 1134
ExtCtrls.TLabeledEdit.OnClick 1135
ExtCtrls.TLabeledEdit.OnContextPopup 1137
ExtCtrls.TLabeledEdit.OnDblClick 1137
ExtCtrls.TLabeledEdit.OnDragDrop 1138
ExtCtrls.TLabeledEdit.OnDragOver 1141
ExtCtrls.TLabeledEdit.OnEndDock 1144
ExtCtrls.TLabeledEdit.OnEndDrag 1144
ExtCtrls.TLabeledEdit.OnEnter 1145
ExtCtrls.TLabeledEdit.OnExit 1148
ExtCtrls.TLabeledEdit.OnKeyDown 1151
ExtCtrls.TLabeledEdit.OnKeyPress 1154
ExtCtrls.TLabeledEdit.OnKeyUp 1155
ExtCtrls.TLabeledEdit.OnMouseActivate 1157
ExtCtrls.TLabeledEdit.OnMouseDown 1158
ExtCtrls.TLabeledEdit.OnMouseEnter 1160
ExtCtrls.TLabeledEdit.OnMouseLeave 1160
ExtCtrls.TLabeledEdit.OnMouseMove 1161
ExtCtrls.TLabeledEdit.OnMouseUp 1163
ExtCtrls.TLabeledEdit.OnStartDock 1165
ExtCtrls.TLabeledEdit.OnStartDrag 1166
ExtCtrls.TLabeledEdit.ParentBiDiMode 1120
ExtCtrls.TLabeledEdit.ParentColor 1120
ExtCtrls.TLabeledEdit.ParentCtl3D 1121
ExtCtrls.TLabeledEdit.ParentDoubleBuffered 1121
ExtCtrls.TLabeledEdit.ParentFont 1121
ExtCtrls.TLabeledEdit.ParentShowHint 1122
ExtCtrls.TLabeledEdit.PasswordChar 1123
ExtCtrls.TLabeledEdit.PopupMenu 1123

ExtCtrls.TLabeledEdit.ReadOnly 1128
ExtCtrls.TLabeledEdit.ShowHint 1128
ExtCtrls.TLabeledEdit.TabOrder 1130
ExtCtrls.TLabeledEdit.TabStop 1130
ExtCtrls.TLabeledEdit.Text 1131
ExtCtrls.TLabeledEdit.TextHint 1132
ExtCtrls.TLabeledEdit.Visible 1132
ExtCtrls.TLabelPosition 1543
ExtCtrls.TLabelPosition enumeration 1543
ExtCtrls.TLBGetColorsEvent 1553
ExtCtrls.TLBGetColorsEvent type 1553
ExtCtrls.TLinkLabel 1167
ExtCtrls.TLinkLabel.Alignment 1169
ExtCtrls.TLinkLabel.Anchors 1169
ExtCtrls.TLinkLabel.AutoSize 1170
ExtCtrls.TLinkLabel.BevelEdges 1170
ExtCtrls.TLinkLabel.BevelInner 1171
ExtCtrls.TLinkLabel.BevelKind 1171
ExtCtrls.TLinkLabel.BevelOuter 1172
ExtCtrls.TLinkLabel.Caption 1172
ExtCtrls.TLinkLabel.Color 1175
ExtCtrls.TLinkLabel.Constraints 1175
ExtCtrls.TLinkLabel.DragCursor 1176
ExtCtrls.TLinkLabel.DragKind 1176
ExtCtrls.TLinkLabel.DragMode 1177
ExtCtrls.TLinkLabel.Enabled 1178
ExtCtrls.TLinkLabel.Font 1179
ExtCtrls.TLinkLabel.OnClick 1191
ExtCtrls.TLinkLabel.OnContextPopup 1193
ExtCtrls.TLinkLabel.OnDblClick 1194
ExtCtrls.TLinkLabel.OnDragDrop 1194
ExtCtrls.TLinkLabel.OnDragOver 1197
ExtCtrls.TLinkLabel.OnEndDock 1200
ExtCtrls.TLinkLabel.OnEndDrag 1201
ExtCtrls.TLinkLabel.OnLinkClick 1202
ExtCtrls.TLinkLabel.OnMouseActivate 1202
ExtCtrls.TLinkLabel.OnMouseDown 1203
ExtCtrls.TLinkLabel.OnMouseEnter 1205
ExtCtrls.TLinkLabel.OnMouseLeave 1205
ExtCtrls.TLinkLabel.OnMouseMove 1206
ExtCtrls.TLinkLabel.OnMouseUp 1208
ExtCtrls.TLinkLabel.OnStartDock 1210
ExtCtrls.TLinkLabel.OnStartDrag 1211
ExtCtrls.TLinkLabel.ParentColor 1180
ExtCtrls.TLinkLabel.ParentFont 1180
ExtCtrls.TLinkLabel.ParentShowHint 1181
ExtCtrls.TLinkLabel.PopupMenu 1181
ExtCtrls.TLinkLabel.ShowHint 1186
ExtCtrls.TLinkLabel.TabOrder 1188
ExtCtrls.TLinkLabel.TabStop 1188
ExtCtrls.TLinkLabel.UseVisualStyle 1189
ExtCtrls.TLinkLabel.Visible 1189
ExtCtrls.TNotebook 1211
ExtCtrls.TNotebook.ActivePage 1217
ExtCtrls.TNotebook.Align 1218
ExtCtrls.TNotebook.Anchors 1221
ExtCtrls.TNotebook.Color 1222
ExtCtrls.TNotebook.Constraints 1222
ExtCtrls.TNotebook.Create 1213
ExtCtrls.TNotebook.CreateParams 1214
ExtCtrls.TNotebook.Ctl3D 1223
ExtCtrls.TNotebook.Destroy 1214
ExtCtrls.TNotebook.DoubleBuffered 1223
ExtCtrls.TNotebook.DragCursor 1225
ExtCtrls.TNotebook.DragKind 1226
ExtCtrls.TNotebook.DragMode 1226
ExtCtrls.TNotebook.Enabled 1228
ExtCtrls.TNotebook.Font 1228
ExtCtrls.TNotebook.GetChildOwner 1215
ExtCtrls.TNotebook.GetChildren 1215
ExtCtrls.TNotebook.OnClick 1242
ExtCtrls.TNotebook.OnContextPopup 1244
ExtCtrls.TNotebook.OnDblClick 1244
ExtCtrls.TNotebook.OnDragDrop 1245
ExtCtrls.TNotebook.OnDragOver 1248
ExtCtrls.TNotebook.OnEndDock 1251
ExtCtrls.TNotebook.OnEndDrag 1251
ExtCtrls.TNotebook.OnEnter 1252

ExtCtrls.TNotebook.OnExit 1255
ExtCtrls.TNotebook.OnMouseActivate 1257
ExtCtrls.TNotebook.OnMouseDown 1258
ExtCtrls.TNotebook.OnMouseEnter 1261
ExtCtrls.TNotebook.OnMouseLeave 1261
ExtCtrls.TNotebook.OnMouseMove 1261
ExtCtrls.TNotebook.OnMouseUp 1263
ExtCtrls.TNotebook.OnPageChanged 1266
ExtCtrls.TNotebook.OnStartDock 1266
ExtCtrls.TNotebook.OnStartDrag 1267
ExtCtrls.TNotebook.PageIndex 1229
ExtCtrls.TNotebook.Pages 1230
ExtCtrls.TNotebook.ParentColor 1230
ExtCtrls.TNotebook.ParentCtl3D 1230
ExtCtrls.TNotebook.ParentDoubleBuffered 1231
ExtCtrls.TNotebook.ParentFont 1231
ExtCtrls.TNotebook.ParentShowHint 1232
ExtCtrls.TNotebook.PopupMenu 1232
ExtCtrls.TNotebook.ReadState 1216
ExtCtrls.TNotebook.ShowControl 1216
ExtCtrls.TNotebook.ShowHint 1237
ExtCtrls.TNotebook.TabOrder 1239
ExtCtrls.TNotebook.TabStop 1239
ExtCtrls.TNotebook.Visible 1240
ExtCtrls.TPage 1267
ExtCtrls.TPage.Caption 1270
ExtCtrls.TPage.Create 1268
ExtCtrls.TPage.Height 1273
ExtCtrls.TPage.OnAlignInsertBefore 1278
ExtCtrls.TPage.OnAlignPosition 1278
ExtCtrls.TPage.Paint 1269
ExtCtrls.TPage.ReadState 1269
ExtCtrls.TPage.TabOrder 1274
ExtCtrls.TPage.Visible 1274
ExtCtrls.TPage.Width 1276
ExtCtrls.TPaintBox 1279
ExtCtrls.TPaintBox.Align 1282
ExtCtrls.TPaintBox.Anchors 1286
ExtCtrls.TPaintBox.Canvas 1286
ExtCtrls.TPaintBox.Color 1288
ExtCtrls.TPaintBox.Constraints 1289
ExtCtrls.TPaintBox.Create 1281
ExtCtrls.TPaintBox.DragCursor 1290
ExtCtrls.TPaintBox.DragKind 1290
ExtCtrls.TPaintBox.DragMode 1290
ExtCtrls.TPaintBox.Enabled 1292
ExtCtrls.TPaintBox.Font 1292
ExtCtrls.TPaintBox.OnClick 1304
ExtCtrls.TPaintBox.OnContextPopup 1305
ExtCtrls.TPaintBox.OnDblClick 1306
ExtCtrls.TPaintBox.OnDragDrop 1306
ExtCtrls.TPaintBox.OnDragOver 1309
ExtCtrls.TPaintBox.OnEndDock 1313
ExtCtrls.TPaintBox.OnEndDrag 1313
ExtCtrls.TPaintBox.OnMouseActivate 1314
ExtCtrls.TPaintBox.OnMouseDown 1315
ExtCtrls.TPaintBox.OnMouseEnter 1317
ExtCtrls.TPaintBox.OnMouseLeave 1318
ExtCtrls.TPaintBox.OnMouseMove 1318
ExtCtrls.TPaintBox.OnMouseUp 1320
ExtCtrls.TPaintBox.OnPaint 1322
ExtCtrls.TPaintBox.OnStartDock 1324
ExtCtrls.TPaintBox.OnStartDrag 1325
ExtCtrls.TPaintBox.Paint 1281
ExtCtrls.TPaintBox.ParentColor 1293
ExtCtrls.TPaintBox.ParentFont 1294
ExtCtrls.TPaintBox.ParentShowHint 1294
ExtCtrls.TPaintBox.PopupMenu 1295
ExtCtrls.TPaintBox.ShowHint 1300
ExtCtrls.TPaintBox.Visible 1301
ExtCtrls.TPanel 1326
ExtCtrls.TPanel.Align 1330
ExtCtrls.TPanel.Alignment 1333
ExtCtrls.TPanel.Anchors 1334
ExtCtrls.TPanel.AutoSize 1334
ExtCtrls.TPanel.BevelEdges 1335
ExtCtrls.TPanel.BevelInner 1335
ExtCtrls.TPanel.BevelKind 1336

ExtCtrls.TPanel.BevelOuter 1336
ExtCtrls.TPanel.BevelWidth 1337
ExtCtrls.TPanel.BiDiMode 1339
ExtCtrls.TPanel.BorderStyle 1339
ExtCtrls.TPanel.BorderWidth 1339
ExtCtrls.TPanel.Caption 1341
ExtCtrls.TPanel.Color 1343
ExtCtrls.TPanel.Constraints 1344
ExtCtrls.TPanel.Ctl3D 1345
ExtCtrls.TPanel.DockManager 1345
ExtCtrls.TPanel.DockSite 1347
ExtCtrls.TPanel.DoubleBuffered 1348
ExtCtrls.TPanel.DragCursor 1350
ExtCtrls.TPanel.DragKind 1350
ExtCtrls.TPanel.DragMode 1351
ExtCtrls.TPanel.Enabled 1352
ExtCtrls.TPanel.Font 1353
ExtCtrls.TPanel.FullRepaint 1354
ExtCtrls.TPanel.Locked 1354
ExtCtrls.TPanel.OnAlignInsertBefore 1368
ExtCtrls.TPanel.OnAlignPosition 1369
ExtCtrls.TPanel.OnCanResize 1370
ExtCtrls.TPanel.OnClick 1370
ExtCtrls.TPanel.OnConstrainedResize 1372
ExtCtrls.TPanel.OnContextPopup 1372
ExtCtrls.TPanel.OnDblClick 1373
ExtCtrls.TPanel.OnDockDrop 1373
ExtCtrls.TPanel.OnDockOver 1375
ExtCtrls.TPanel.OnDragDrop 1378
ExtCtrls.TPanel.OnDragOver 1381
ExtCtrls.TPanel.OnEndDock 1384
ExtCtrls.TPanel.OnEndDrag 1384
ExtCtrls.TPanel.OnEnter 1385
ExtCtrls.TPanel.OnExit 1388
ExtCtrls.TPanel.OnGetSiteInfo 1390
ExtCtrls.TPanel.OnMouseActivate 1393
ExtCtrls.TPanel.OnMouseDown 1394
ExtCtrls.TPanel.OnMouseEnter 1396
ExtCtrls.TPanel.OnMouseLeave 1396
ExtCtrls.TPanel.OnMouseMove 1396
ExtCtrls.TPanel.OnMouseUp 1399
ExtCtrls.TPanel.OnResize 1401
ExtCtrls.TPanel.OnStartDock 1402
ExtCtrls.TPanel.OnStartDrag 1402
ExtCtrls.TPanel.OnUnDock 1403
ExtCtrls.TPanel.Padding 1354
ExtCtrls.TPanel.ParentBackground 1355
ExtCtrls.TPanel.ParentBiDiMode 1355
ExtCtrls.TPanel.ParentColor 1355
ExtCtrls.TPanel.ParentCtl3D 1356
ExtCtrls.TPanel.ParentDoubleBuffered 1356
ExtCtrls.TPanel.ParentFont 1356
ExtCtrls.TPanel.ParentShowHint 1357
ExtCtrls.TPanel.PopupMenu 1358
ExtCtrls.TPanel.ShowCaption 1363
ExtCtrls.TPanel.ShowHint 1363
ExtCtrls.TPanel.TabOrder 1364
ExtCtrls.TPanel.TabStop 1365
ExtCtrls.TPanel.UseDockManager 1365
ExtCtrls.TPanel.VerticalAlignment 1366
ExtCtrls.TPanel.Visible 1366
ExtCtrls.TPanelBevel 1553
ExtCtrls.TPanelBevel type 1553
ExtCtrls.TRadioGroup 1406
ExtCtrls.TRadioGroup.Align 1409
ExtCtrls.TRadioGroup.Anchors 1412
ExtCtrls.TRadioGroup.BiDiMode 1413
ExtCtrls.TRadioGroup.Caption 1413
ExtCtrls.TRadioGroup.Color 1416
ExtCtrls.TRadioGroup.Columns 1416
ExtCtrls.TRadioGroup.Constraints 1417
ExtCtrls.TRadioGroup.Ctl3D 1417
ExtCtrls.TRadioGroup.DoubleBuffered 1418
ExtCtrls.TRadioGroup.DragCursor 1420
ExtCtrls.TRadioGroup.DragKind 1420
ExtCtrls.TRadioGroup.DragMode 1420
ExtCtrls.TRadioGroup.Enabled 1422
ExtCtrls.TRadioGroup.Font 1422

ExtCtrls.TRadioGroup.ItemIndex 1423
ExtCtrls.TRadioGroup.Items 1424
ExtCtrls.TRadioGroup.OnClick 1438
ExtCtrls.TRadioGroup.OnContextPopup 1440
ExtCtrls.TRadioGroup.OnDragDrop 1440
ExtCtrls.TRadioGroup.OnDragOver 1443
ExtCtrls.TRadioGroup.OnEndDock 1447
ExtCtrls.TRadioGroup.OnEndDrag 1447
ExtCtrls.TRadioGroup.OnEnter 1448
ExtCtrls.TRadioGroup.OnExit 1451
ExtCtrls.TRadioGroup.OnStartDock 1453
ExtCtrls.TRadioGroup.OnStartDrag 1454
ExtCtrls.TRadioGroup.ParentBackground 1425
ExtCtrls.TRadioGroup.ParentBiDiMode 1426
ExtCtrls.TRadioGroup.ParentColor 1426
ExtCtrls.TRadioGroup.ParentCtl3D 1427
ExtCtrls.TRadioGroup.ParentDoubleBuffered 1427
ExtCtrls.TRadioGroup.ParentFont 1427
ExtCtrls.TRadioGroup.ParentShowHint 1428
ExtCtrls.TRadioGroup.PopupMenu 1428
ExtCtrls.TRadioGroup.ShowHint 1433
ExtCtrls.TRadioGroup.TabOrder 1435
ExtCtrls.TRadioGroup.TabStop 1435
ExtCtrls.TRadioGroup.Visible 1436
ExtCtrls.TRadioGroup.WordWrap 1437
ExtCtrls.TResizeStyle 1544
ExtCtrls.TResizeStyle enumeration 1544
ExtCtrls.TRowCollection 1455
ExtCtrls.TRowCollection.Add 1456
ExtCtrls.TRowCollection.Create 1456
ExtCtrls.TRowCollection.GetItemAttr 1456
ExtCtrls.TRowCollection.Notify 1457
ExtCtrls.TRowItem 1457
ExtCtrls.TRowSize 1554
ExtCtrls.TRowSize type 1554
ExtCtrls.TSectionEvent 1554
ExtCtrls.TSectionEvent type 1554
ExtCtrls.TShape 1458
ExtCtrls.TShape.Align 1461
ExtCtrls.TShape.Anchors 1465
ExtCtrls.TShape.Brush 1465
ExtCtrls.TShape.Constraints 1466
ExtCtrls.TShape.Create 1459
ExtCtrls.TShape.Destroy 1460
ExtCtrls.TShape.DragCursor 1466
ExtCtrls.TShape.DragKind 1467
ExtCtrls.TShape.DragMode 1467
ExtCtrls.TShape.Enabled 1469
ExtCtrls.TShape.OnContextPopup 1474
ExtCtrls.TShape.OnDragDrop 1475
ExtCtrls.TShape.OnDragOver 1478
ExtCtrls.TShape.OnEndDock 1481
ExtCtrls.TShape.OnEndDrag 1481
ExtCtrls.TShape.OnMouseActivate 1482
ExtCtrls.TShape.OnMouseDown 1483
ExtCtrls.TShape.OnMouseEnter 1486
ExtCtrls.TShape.OnMouseLeave 1486
ExtCtrls.TShape.OnMouseMove 1486
ExtCtrls.TShape.OnMouseUp 1488
ExtCtrls.TShape.OnStartDock 1491
ExtCtrls.TShape.OnStartDrag 1491
ExtCtrls.TShape.Paint 1460
ExtCtrls.TShape.ParentShowHint 1469
ExtCtrls.TShape.Pen 1470
ExtCtrls.TShape.Shape 1470
ExtCtrls.TShape.ShowHint 1471
ExtCtrls.TShape.StyleChanged 1461
ExtCtrls.TShape.Visible 1472
ExtCtrls.TShapeType 1544
ExtCtrls.TShapeType enumeration 1544
ExtCtrls.TSizeStyle 1545
ExtCtrls.TSizeStyle enumeration 1545
ExtCtrls.TSplitter 1492
ExtCtrls.TSplitter.Align 1499
ExtCtrls.TSplitter.AutoSnap 1503
ExtCtrls.TSplitter.Beveled 1503
ExtCtrls.TSplitter.CanResize 1496
ExtCtrls.TSplitter.Canvas 1504

ExtCtrls.TSplitter.Color 1506
ExtCtrls.TSplitter.Constraints 1507
ExtCtrls.TSplitter.Create 1493
ExtCtrls.TSplitter.Cursor 1507
ExtCtrls.TSplitter.Destroy 1495
ExtCtrls.TSplitter.DoCanResize 1496
ExtCtrls.TSplitter.MinSize 1511
ExtCtrls.TSplitter.MouseDown 1497
ExtCtrls.TSplitter.MouseMove 1497
ExtCtrls.TSplitter.MouseUp 1497
ExtCtrls.TSplitter.OnCanResize 1517
ExtCtrls.TSplitter.OnMoved 1517
ExtCtrls.TSplitter.OnPaint 1520
ExtCtrls.TSplitter.Paint 1498
ExtCtrls.TSplitter.ParentColor 1512
ExtCtrls.TSplitter.RequestAlign 1498
ExtCtrls.TSplitter.ResizeStyle 1513
ExtCtrls.TSplitter.StopSizing 1499
ExtCtrls.TSplitter.Visible 1514
ExtCtrls.TSplitter.Width 1515
ExtCtrls.TSplitterCanResizeEvent 1554
ExtCtrls.TSplitterCanResizeEvent type 1554
ExtCtrls.TSysLinkEvent 1555
ExtCtrls.TSysLinkEvent type 1555
ExtCtrls.TSysLinkType 1546
ExtCtrls.TSysLinkType enumeration 1546
ExtCtrls.TTimer 1520
ExtCtrls.TTimer.Create 1521
ExtCtrls.TTimer.Destroy 1522
ExtCtrls.TTimer.Enabled 1523
ExtCtrls.TTimer.Interval 1525
ExtCtrls.TTimer.OnTimer 1527
ExtCtrls.TTimer.Timer 1522
ExtCtrls.TTrayIcon 1528
ExtCtrls.TTrayIcon.Animate 1530
ExtCtrls.TTrayIcon.AnimateInterval 1530
ExtCtrls.TTrayIcon.BalloonFlags 1530
ExtCtrls.TTrayIcon.BalloonHint 1531
ExtCtrls.TTrayIcon.BalloonTimeout 1531
ExtCtrls.TTrayIcon.BalloonTitle 1531
ExtCtrls.TTrayIcon.Hint 1532
ExtCtrls.TTrayIcon.Icon 1532
ExtCtrls.TTrayIcon.IconIndex 1532
ExtCtrls.TTrayIcon.Icons 1532
ExtCtrls.TTrayIcon.OnAnimate 1533
ExtCtrls.TTrayIcon.OnBalloonClick 1534
ExtCtrls.TTrayIcon.OnClick 1534
ExtCtrls.TTrayIcon.OnDblClick 1534
ExtCtrls.TTrayIcon.OnMouseDown 1534
ExtCtrls.TTrayIcon.OnMouseMove 1535
ExtCtrls.TTrayIcon.OnMouseUp 1535
ExtCtrls.TTrayIcon.PopupMenu 1533
ExtCtrls.TTrayIcon.Visible 1533
ExtCtrls.WM_SYSTEM_TRAY_MESSAGE 1556
ExtCtrls.WM_SYSTEM_TRAY_MESSAGE constant 1556
ExtDlgs 1559
ExtDlgs namespace 1559
    Classes 1560
    Constants 1586
    Functions 1585
ExtDlgs.DefaultEncodingNames 1586
ExtDlgs.DefaultEncodingNames constant 1586
ExtDlgs.StandardEncodingFromName 1585
ExtDlgs.StandardEncodingFromName function 1585
ExtDlgs.TOpenPictureDialog 1560
ExtDlgs.TOpenPictureDialog.Create 1562
ExtDlgs.TOpenPictureDialog.DoClose 1562
ExtDlgs.TOpenPictureDialog.DoSelectionChange 1563
ExtDlgs.TOpenPictureDialog.DoShow 1563
ExtDlgs.TOpenPictureDialog.Execute 1564
ExtDlgs.TOpenPictureDialog.Filter 1566
ExtDlgs.TOpenPictureDialog.ImageCtrl 1569
ExtDlgs.TOpenPictureDialog.PictureLabel 1570
ExtDlgs.TOpenPictureDialog.PreviewClick 1565
ExtDlgs.TOpenTextFileDialog 1570
ExtDlgs.TOpenTextFileDialog.Create 1571
ExtDlgs.TOpenTextFileDialog.Destroy 1571
ExtDlgs.TOpenTextFileDialog.DoClose 1572

ExtDlgs.TOpenTextFileDialog.DoShow 1572
ExtDlgs.TOpenTextFileDialog.EncodingIndex 1576
ExtDlgs.TOpenTextFileDialog.Encodings 1576
ExtDlgs.TOpenTextFileDialog.Execute 1573
ExtDlgs.TSavePictureDialog 1577
ExtDlgs.TSavePictureDialog.Execute 1579
ExtDlgs.TSaveTextFileDialog 1580
ExtDlgs.TSaveTextFileDialog.Execute 1581
exUnderflow enumeration member 4769
exZeroDivide enumeration member 4769

F

FileCtrl 1587
FileCtrl namespace 1587
- Classes 1589
- Constants 1889
- Functions 1879
- Structs, Records, Enums 1883
- Types 1887

FileCtrl.DirectoryExists 1879
FileCtrl.DirectoryExists function 1879
FileCtrl.ForceDirectories 1879
FileCtrl.ForceDirectories function 1879
FileCtrl.MinimizeName 1880
FileCtrl.MinimizeName function 1880
FileCtrl.ProcessPath 1881
FileCtrl.ProcessPath function 1881
FileCtrl.SelectDirectory 1881
FileCtrl.SelectDirectory function 1881
FileCtrl.TDirectoryListBox 1589
FileCtrl.TDirectoryListBox.Align 1602
FileCtrl.TDirectoryListBox.Anchors 1605
FileCtrl.TDirectoryListBox.AutoComplete 1606
FileCtrl.TDirectoryListBox.BevelEdges 1606
FileCtrl.TDirectoryListBox.BevelInner 1606
FileCtrl.TDirectoryListBox.BevelKind 1607
FileCtrl.TDirectoryListBox.BevelOuter 1607
FileCtrl.TDirectoryListBox.BuildList 1595
FileCtrl.TDirectoryListBox.CaseSensitive 1608
FileCtrl.TDirectoryListBox.Change 1595
FileCtrl.TDirectoryListBox.ClosedBMP 1593
FileCtrl.TDirectoryListBox.Color 1608
FileCtrl.TDirectoryListBox.Columns 1609
FileCtrl.TDirectoryListBox.Constraints 1610
FileCtrl.TDirectoryListBox.Create 1592
FileCtrl.TDirectoryListBox.CreateWnd 1595
FileCtrl.TDirectoryListBox.Ctl3D 1610
FileCtrl.TDirectoryListBox.CurrentBMP 1593
FileCtrl.TDirectoryListBox.DblClick 1596
FileCtrl.TDirectoryListBox.Destroy 1593
FileCtrl.TDirectoryListBox.Directory 1611
FileCtrl.TDirectoryListBox.DirLabel 1612
FileCtrl.TDirectoryListBox.DisplayCase 1596
FileCtrl.TDirectoryListBox.DoubleBuffered 1613
FileCtrl.TDirectoryListBox.DragCursor 1615
FileCtrl.TDirectoryListBox.DragMode 1615
FileCtrl.TDirectoryListBox.DrawItem 1597
FileCtrl.TDirectoryListBox.Drive 1617
FileCtrl.TDirectoryListBox.Enabled 1618
FileCtrl.TDirectoryListBox.FDirectory 1594
FileCtrl.TDirectoryListBox.FileCompareText 1597
FileCtrl.TDirectoryListBox.FileList 1618
FileCtrl.TDirectoryListBox.FOnChange 1594
FileCtrl.TDirectoryListBox.Font 1621
FileCtrl.TDirectoryListBox.GetItemPath 1597
FileCtrl.TDirectoryListBox.ImeMode 1622
FileCtrl.TDirectoryListBox.ImeName 1622
FileCtrl.TDirectoryListBox.IntegralHeight 1623
FileCtrl.TDirectoryListBox.ItemHeight 1623
FileCtrl.TDirectoryListBox.KeyPress 1598
FileCtrl.TDirectoryListBox.Notification 1598
FileCtrl.TDirectoryListBox.OnChange 1636
FileCtrl.TDirectoryListBox.OnClick 1638
FileCtrl.TDirectoryListBox.OnContextPopup 1639
FileCtrl.TDirectoryListBox.OnDblClick 1640
FileCtrl.TDirectoryListBox.OnDragDrop 1640
FileCtrl.TDirectoryListBox.OnDragOver 1643
FileCtrl.TDirectoryListBox.OnEndDrag 1647

FileCtrl.TDirectoryListBox.OnEnter 1648
FileCtrl.TDirectoryListBox.OnExit 1650
FileCtrl.TDirectoryListBox.OnKeyDown 1653
FileCtrl.TDirectoryListBox.OnKeyPress 1657
FileCtrl.TDirectoryListBox.OnKeyUp 1658
FileCtrl.TDirectoryListBox.OnMouseActivate 1659
FileCtrl.TDirectoryListBox.OnMouseDown 1660
FileCtrl.TDirectoryListBox.OnMouseEnter 1663
FileCtrl.TDirectoryListBox.OnMouseLeave 1663
FileCtrl.TDirectoryListBox.OnMouseMove 1663
FileCtrl.TDirectoryListBox.OnMouseUp 1665
FileCtrl.TDirectoryListBox.OnStartDrag 1668
FileCtrl.TDirectoryListBox.OpenCurrent 1598
FileCtrl.TDirectoryListBox.OpenedBMP 1594
FileCtrl.TDirectoryListBox.ParentColor 1624
FileCtrl.TDirectoryListBox.ParentCtl3D 1624
FileCtrl.TDirectoryListBox.ParentDoubleBuffered 1625
FileCtrl.TDirectoryListBox.ParentFont 1625
FileCtrl.TDirectoryListBox.ParentShowHint 1625
FileCtrl.TDirectoryListBox.PopupMenu 1626
FileCtrl.TDirectoryListBox.PreserveCase 1631
FileCtrl.TDirectoryListBox.ReadBitmaps 1599
FileCtrl.TDirectoryListBox.ReadDirectoryNames 1599
FileCtrl.TDirectoryListBox.ShowHint 1631
FileCtrl.TDirectoryListBox.TabOrder 1633
FileCtrl.TDirectoryListBox.TabStop 1633
FileCtrl.TDirectoryListBox.Update 1600
FileCtrl.TDirectoryListBox.Visible 1634
FileCtrl.TDriveComboBox 1668
FileCtrl.TDriveComboBox.Align 1676
FileCtrl.TDriveComboBox.Anchors 1680
FileCtrl.TDriveComboBox.AutoComplete 1680
FileCtrl.TDriveComboBox.AutoDropDown 1681
FileCtrl.TDriveComboBox.BevelEdges 1682
FileCtrl.TDriveComboBox.BevelInner 1682
FileCtrl.TDriveComboBox.BevelKind 1683
FileCtrl.TDriveComboBox.BevelOuter 1683
FileCtrl.TDriveComboBox.BuildList 1673
FileCtrl.TDriveComboBox.CDROMBMP 1672
FileCtrl.TDriveComboBox.Click 1673
FileCtrl.TDriveComboBox.Color 1684
FileCtrl.TDriveComboBox.Constraints 1684
FileCtrl.TDriveComboBox.Create 1671
FileCtrl.TDriveComboBox.CreateWnd 1674
FileCtrl.TDriveComboBox.Ctl3D 1685
FileCtrl.TDriveComboBox.Destroy 1671
FileCtrl.TDriveComboBox.DirList 1685
FileCtrl.TDriveComboBox.DoubleBuffered 1687
FileCtrl.TDriveComboBox.DragCursor 1688
FileCtrl.TDriveComboBox.DragMode 1689
FileCtrl.TDriveComboBox.DrawItem 1674
FileCtrl.TDriveComboBox.Drive 1690
FileCtrl.TDriveComboBox.Enabled 1692
FileCtrl.TDriveComboBox.FixedBMP 1672
FileCtrl.TDriveComboBox.FloppyBMP 1672
FileCtrl.TDriveComboBox.Font 1692
FileCtrl.TDriveComboBox.ImeMode 1693
FileCtrl.TDriveComboBox.ImeName 1694
FileCtrl.TDriveComboBox.NetworkBMP 1672
FileCtrl.TDriveComboBox.Notification 1675
FileCtrl.TDriveComboBox.OnChange 1708
FileCtrl.TDriveComboBox.OnClick 1710
FileCtrl.TDriveComboBox.OnContextPopup 1712
FileCtrl.TDriveComboBox.OnDblClick 1712
FileCtrl.TDriveComboBox.OnDragDrop 1713
FileCtrl.TDriveComboBox.OnDragOver 1716
FileCtrl.TDriveComboBox.OnDropDown 1719
FileCtrl.TDriveComboBox.OnEndDrag 1720
FileCtrl.TDriveComboBox.OnEnter 1721
FileCtrl.TDriveComboBox.OnExit 1724
FileCtrl.TDriveComboBox.OnKeyDown 1727
FileCtrl.TDriveComboBox.OnKeyPress 1730
FileCtrl.TDriveComboBox.OnKeyUp 1731
FileCtrl.TDriveComboBox.OnMouseEnter 1733
FileCtrl.TDriveComboBox.OnMouseLeave 1733
FileCtrl.TDriveComboBox.OnStartDrag 1734
FileCtrl.TDriveComboBox.ParentColor 1694
FileCtrl.TDriveComboBox.ParentCtl3D 1695

FileCtrl.TDriveComboBox.ParentDoubleBuffered 1695
FileCtrl.TDriveComboBox.ParentFont 1695
FileCtrl.TDriveComboBox.ParentShowHint 1696
FileCtrl.TDriveComboBox.PopupMenu 1696
FileCtrl.TDriveComboBox.RAMBMP 1673
FileCtrl.TDriveComboBox.ShowHint 1701
FileCtrl.TDriveComboBox.TabOrder 1703
FileCtrl.TDriveComboBox.TabStop 1704
FileCtrl.TDriveComboBox.Text 1704
FileCtrl.TDriveComboBox.TextCase 1705
FileCtrl.TDriveComboBox.Visible 1706
FileCtrl.TDriveType 1883
FileCtrl.TDriveType enumeration 1883
FileCtrl.TFileAttr 1883
FileCtrl.TFileAttr enumeration 1883
FileCtrl.TFileListBox 1734
FileCtrl.TFileListBox.Align 1748
FileCtrl.TFileListBox.Anchors 1751
FileCtrl.TFileListBox.ApplyFilePath 1742
FileCtrl.TFileListBox.AutoComplete 1752
FileCtrl.TFileListBox.BevelEdges 1752
FileCtrl.TFileListBox.BevelInner 1753
FileCtrl.TFileListBox.BevelKind 1753
FileCtrl.TFileListBox.BevelOuter 1754
FileCtrl.TFileListBox.Change 1743
FileCtrl.TFileListBox.Click 1744
FileCtrl.TFileListBox.Color 1754
FileCtrl.TFileListBox.Constraints 1755
FileCtrl.TFileListBox.Create 1738
FileCtrl.TFileListBox.CreateWnd 1744
FileCtrl.TFileListBox.Ctl3D 1755
FileCtrl.TFileListBox.Destroy 1738
FileCtrl.TFileListBox.DirBMP 1739
FileCtrl.TFileListBox.Directory 1756
FileCtrl.TFileListBox.DoubleBuffered 1757
FileCtrl.TFileListBox.DragCursor 1759
FileCtrl.TFileListBox.DragMode 1759
FileCtrl.TFileListBox.DrawItem 1745
FileCtrl.TFileListBox.Drive 1761
FileCtrl.TFileListBox.Enabled 1762
FileCtrl.TFileListBox.ExeBMP 1739
FileCtrl.TFileListBox.ExtendedSelect 1762
FileCtrl.TFileListBox.FDirectory 1739
FileCtrl.TFileListBox.FDirList 1740
FileCtrl.TFileListBox.FFileEdit 1740
FileCtrl.TFileListBox.FFileType 1740
FileCtrl.TFileListBox.FFilterCombo 1740
FileCtrl.TFileListBox.FileEdit 1763
FileCtrl.TFileListBox.FileName 1764
FileCtrl.TFileListBox.FileType 1764
FileCtrl.TFileListBox.FLastSel 1741
FileCtrl.TFileListBox.FMask 1741
FileCtrl.TFileListBox.FOnChange 1741
FileCtrl.TFileListBox.Font 1765
FileCtrl.TFileListBox.FShowGlyphs 1741
FileCtrl.TFileListBox.GetFilePath 1745
FileCtrl.TFileListBox.ImeMode 1766
FileCtrl.TFileListBox.ImeName 1767
FileCtrl.TFileListBox.IntegralHeight 1767
FileCtrl.TFileListBox.ItemHeight 1768
FileCtrl.TFileListBox.Mask 1768
FileCtrl.TFileListBox.MultiSelect 1769
FileCtrl.TFileListBox.Notification 1745
FileCtrl.TFileListBox.OnChange 1782
FileCtrl.TFileListBox.OnClick 1782
FileCtrl.TFileListBox.OnContextPopup 1784
FileCtrl.TFileListBox.OnDblClick 1785
FileCtrl.TFileListBox.OnDragDrop 1785
FileCtrl.TFileListBox.OnDragOver 1788
FileCtrl.TFileListBox.OnEndDrag 1791
FileCtrl.TFileListBox.OnEnter 1792
FileCtrl.TFileListBox.OnExit 1795
FileCtrl.TFileListBox.OnKeyDown 1797
FileCtrl.TFileListBox.OnKeyPress 1801
FileCtrl.TFileListBox.OnKeyUp 1802
FileCtrl.TFileListBox.OnMouseActivate 1804
FileCtrl.TFileListBox.OnMouseDown 1805
FileCtrl.TFileListBox.OnMouseEnter 1807

FileCtrl.TFileListBox.OnMouseLeave 1807
FileCtrl.TFileListBox.OnMouseMove 1808
FileCtrl.TFileListBox.OnMouseUp 1810
FileCtrl.TFileListBox.OnStartDrag 1812
FileCtrl.TFileListBox.ParentColor 1770
FileCtrl.TFileListBox.ParentCtl3D 1770
FileCtrl.TFileListBox.ParentDoubleBuffered 1771
FileCtrl.TFileListBox.ParentFont 1771
FileCtrl.TFileListBox.ParentShowHint 1771
FileCtrl.TFileListBox.PopupMenu 1772
FileCtrl.TFileListBox.ReadBitmaps 1746
FileCtrl.TFileListBox.ReadFileNames 1746
FileCtrl.TFileListBox.ShowGlyphs 1777
FileCtrl.TFileListBox.ShowHint 1777
FileCtrl.TFileListBox.TabOrder 1779
FileCtrl.TFileListBox.TabStop 1779
FileCtrl.TFileListBox.UnknownBMP 1741
FileCtrl.TFileListBox.Update 1746
FileCtrl.TFileListBox.Visible 1780
FileCtrl.TFileType 1887
FileCtrl.TFileType type 1887
FileCtrl.TFilterComboBox 1813
FileCtrl.TFilterComboBox.Align 1819
FileCtrl.TFilterComboBox.Anchors 1822
FileCtrl.TFilterComboBox.AutoComplete 1823
FileCtrl.TFilterComboBox.AutoDropDown 1824
FileCtrl.TFilterComboBox.BevelEdges 1825
FileCtrl.TFilterComboBox.BevelInner 1825
FileCtrl.TFilterComboBox.BevelKind 1825
FileCtrl.TFilterComboBox.BevelOuter 1826
FileCtrl.TFilterComboBox.BuildList 1816
FileCtrl.TFilterComboBox.Change 1816
FileCtrl.TFilterComboBox.Click 1816
FileCtrl.TFilterComboBox.Color 1826
FileCtrl.TFilterComboBox.Constraints 1827
FileCtrl.TFilterComboBox.Create 1815
FileCtrl.TFilterComboBox.CreateWnd 1817
FileCtrl.TFilterComboBox.Ctl3D 1828
FileCtrl.TFilterComboBox.Destroy 1815
FileCtrl.TFilterComboBox.DoubleBuffered 1828
FileCtrl.TFilterComboBox.DragCursor 1830
FileCtrl.TFilterComboBox.DragMode 1830
FileCtrl.TFilterComboBox.Enabled 1832
FileCtrl.TFilterComboBox.FileList 1832
FileCtrl.TFilterComboBox.Filter 1833
FileCtrl.TFilterComboBox.Font 1835
FileCtrl.TFilterComboBox.ImeMode 1836
FileCtrl.TFilterComboBox.ImeName 1836
FileCtrl.TFilterComboBox.Mask 1837
FileCtrl.TFilterComboBox.Notification 1817
FileCtrl.TFilterComboBox.OnChange 1851
FileCtrl.TFilterComboBox.OnClick 1853
FileCtrl.TFilterComboBox.OnContextPopup 1855
FileCtrl.TFilterComboBox.OnDblClick 1856
FileCtrl.TFilterComboBox.OnDragDrop 1856
FileCtrl.TFilterComboBox.OnDragOver 1859
FileCtrl.TFilterComboBox.OnDropDown 1862
FileCtrl.TFilterComboBox.OnEndDrag 1864
FileCtrl.TFilterComboBox.OnEnter 1865
FileCtrl.TFilterComboBox.OnExit 1867
FileCtrl.TFilterComboBox.OnKeyDown 1870
FileCtrl.TFilterComboBox.OnKeyPress 1874
FileCtrl.TFilterComboBox.OnKeyUp 1875
FileCtrl.TFilterComboBox.OnMouseEnter 1876
FileCtrl.TFilterComboBox.OnMouseLeave 1877
FileCtrl.TFilterComboBox.OnStartDrag 1877
FileCtrl.TFilterComboBox.ParentColor 1838
FileCtrl.TFilterComboBox.ParentCtl3D 1839
FileCtrl.TFilterComboBox.ParentDoubleBuffered 1839
FileCtrl.TFilterComboBox.ParentFont 1839
FileCtrl.TFilterComboBox.ParentShowHint 1840
FileCtrl.TFilterComboBox.PopupMenu 1840
FileCtrl.TFilterComboBox.ShowHint 1845
FileCtrl.TFilterComboBox.TabOrder 1847
FileCtrl.TFilterComboBox.TabStop 1847
FileCtrl.TFilterComboBox.Text 1848
FileCtrl.TFilterComboBox.Visible 1849
FileCtrl.TSelectDirExtOpt 1884

FileCtrl.TSelectDirExtOpt enumeration 1884
FileCtrl.TSelectDirExtOpts 1887
FileCtrl.TSelectDirExtOpts type 1887
FileCtrl.TSelectDirOpt 1885
FileCtrl.TSelectDirOpt enumeration 1885
FileCtrl.TSelectDirOpts 1888
FileCtrl.TSelectDirOpts type 1888
FileCtrl.TTextCase 1886
FileCtrl.TTextCase enumeration 1886
FileCtrl.WNTYPE_DRIVE 1889
FileCtrl.WNTYPE_DRIVE constant 1889
fkCommand enumeration member 4965
fkHandle enumeration member 4965
fkShortCut enumeration member 4965
fmAlternate enumeration member 2864
fmGet enumeration member 5682
fmoAllBaseMethods enumeration member 4339
fmoNoBaseMethods enumeration member 4339
fmoRTTIBaseMethods enumeration member 4339
fmPost enumeration member 5682
FMTBcd 1891
FMTBcd namespace 1891
Classes 1894
Constants 1915
Functions 1896
Structs, Records, Enums 1913
Types 1914
FMTBcd._DefaultDecimals 1915
FMTBcd._DefaultDecimals constant 1915
FMTBcd._NoDecimal 1915
FMTBcd._NoDecimal constant 1915
FMTBcd.BcdAdd 1897
FMTBcd.BcdAdd function 1897
FMTBcd.BcdCompare 1897
FMTBcd.BcdCompare function 1897
FMTBcd.BcdDivide 1898
FMTBcd.BcdDivide function 1898
FMTBcd.BcdFromBytes 1899
FMTBcd.BcdFromBytes function 1899
FMTBcd.BcdMultiply 1899
FMTBcd.BcdMultiply function 1899
FMTBcd.BcdPrecision 1900
FMTBcd.BcdPrecision function 1900
FMTBcd.BcdScale 1900
FMTBcd.BcdScale function 1900
FMTBcd.BcdSubtract 1901
FMTBcd.BcdSubtract function 1901
FMTBcd.BcdToBytes 1901
FMTBcd.BcdToBytes function 1901
FMTBcd.BCDToCurr 1901
FMTBcd.BCDToCurr function 1901
FMTBcd.BcdToDouble 1902
FMTBcd.BcdToDouble function 1902
FMTBcd.BcdToInteger 1902
FMTBcd.BcdToInteger function 1902
FMTBcd.BcdToStr 1903
FMTBcd.BcdToStr function 1903
FMTBcd.BcdToStrF 1903
FMTBcd.BcdToStrF function 1903
FMTBcd.CurrToBCD 1904
FMTBcd.CurrToBCD function 1904
FMTBcd.DefaultFMTBcdScale 1916
FMTBcd.DefaultFMTBcdScale constant 1916
FMTBcd.DoubleToBcd 1905
FMTBcd.DoubleToBcd function 1905
FMTBcd.EBcdException 1894
FMTBcd.EBcdOverflowException 1894
FMTBcd.FormatBcd 1905
FMTBcd.FormatBcd function 1905
FMTBcd.IntegerToBcd 1906
FMTBcd.IntegerToBcd function 1906
FMTBcd.IsBcdNegative 1906
FMTBcd.IsBcdNegative function 1906
FMTBcd.MaxBcdPrecision 1916
FMTBcd.MaxBcdPrecision constant 1916
FMTBcd.MaxBcdScale 1916
FMTBcd.MaxBcdScale constant 1916
FMTBcd.MaxFMTBcdDigits 1917

FMTBcd.MaxFMTBcdDigits constant 1917
FMTBcd.MaxFMTBcdFractionSize 1917
FMTBcd.MaxFMTBcdFractionSize constant 1917
FMTBcd.MaxStringDigits 1917
FMTBcd.MaxStringDigits constant 1917
FMTBcd.NormalizeBcd 1907
FMTBcd.NormalizeBcd function 1907
FMTBcd.NullBcd 1918
FMTBcd.NullBcd constant 1918
FMTBcd.OldBCDToCurr 1907
FMTBcd.OldBCDToCurr function 1907
FMTBcd.OldCurrToBCD 1908
FMTBcd.OldCurrToBCD function 1908
FMTBcd.PBcd 1914
FMTBcd.PBcd type 1914
FMTBcd.RoundAt 1908
FMTBcd.RoundAt function 1908
FMTBcd.SizeOfFraction 1918
FMTBcd.SizeOfFraction constant 1918
FMTBcd.SizeOfTBCD 1918
FMTBcd.SizeOfTBCD constant 1918
FMTBcd.StrToBcd 1908
FMTBcd.StrToBcd function 1908
FMTBcd.TBcd 1913
FMTBcd.TBcd record 1913
FMTBcd.TNibbleArray 1914
FMTBcd.TNibbleArray type 1914
FMTBcd.TryStrToBcd 1909
FMTBcd.TryStrToBcd function 1909
FMTBcd.VarFMTBcd 1910
FMTBcd.VarFMTBcd function 1910
FMTBcd.VarFMTBcdCreate 1910
FMTBcd.VarFMTBcdCreate function 1910
FMTBcd.VarIsFMTBcd 1911
FMTBcd.VarIsFMTBcd function 1911
FMTBcd.VarToBcd 1911
FMTBcd.VarToBcd function 1911
fmWinding enumeration member 2864
Forced enumeration member 4022
Forms 1921
Forms namespace 1921
  Classes 1924
  Functions 2574
  Interfaces 2566
  Structs, Records, Enums 2582
  Types 2595
  Variables 2612
Forms.Application 2612
Forms.Application variable 2612
Forms.Ctl3DBtnWndProc 2614
Forms.Ctl3DBtnWndProc variable 2614
Forms.Ctl3DCtlColorEx 2614
Forms.Ctl3DCtlColorEx variable 2614
Forms.Ctl3DDlgFramePaint 2614
Forms.Ctl3DDlgFramePaint variable 2614
Forms.DisableTaskWindows 2574
Forms.DisableTaskWindows function 2574
Forms.DoneCtl3D 2574
Forms.DoneCtl3D function 2574
Forms.EnableTaskWindows 2575
Forms.EnableTaskWindows function 2575
Forms.ForegroundTask 2575
Forms.ForegroundTask function 2575
Forms.GetParentForm 2576
Forms.GetParentForm function 2576
Forms.HintWindowClass 2615
Forms.HintWindowClass variable 2615
Forms.IDesignerHook 2566
Forms.IDesignerHook.Form 2571
Forms.IDesignerHook.GetCustomForm 2567
Forms.IDesignerHook.GetIsControl 2568
Forms.IDesignerHook.GetRoot 2568
Forms.IDesignerHook.IsControl 2571
Forms.IDesignerHook.IsDesignMsg 2568
Forms.IDesignerHook.PaintGrid 2569
Forms.IDesignerHook.PaintMenu 2569
Forms.IDesignerHook.SetCustomForm 2569
Forms.IDesignerHook.SetIsControl 2570

Forms.IDesignerHook.UniqueName 2570
Forms.IDesignerHook.ValidateRename 2570
Forms.InitCtl3D 2576
Forms.InitCtl3D function 2576
Forms.IOleForm 2572
Forms.IOleForm.OnDestroy 2572
Forms.IOleForm.OnResize 2573
Forms.IsAccel 2576
Forms.IsAccel function 2576
Forms.KeyboardStateToShiftState 2577
Forms.KeyboardStateToShiftState function 2577
Forms.KeyDataToShiftState 2578
Forms.KeyDataToShiftState function 2578
Forms.KeysToShiftState 2578
Forms.KeysToShiftState function 2578
Forms.PCursorRec 2595
Forms.PCursorRec type 2595
Forms.PHintInfo 2596
Forms.PHintInfo type 2596
Forms.RestoreFocusState 2579
Forms.RestoreFocusState function 2579
Forms.SaveFocusState 2579
Forms.SaveFocusState function 2579
Forms.Screen 2615
Forms.Screen variable 2615
Forms.SetAutoSubClass 2579
Forms.SetAutoSubClass function 2579
Forms.SetLayeredWindowAttributes 2617
Forms.SetLayeredWindowAttributes variable 2617
Forms.Subclass3DDlg 2580
Forms.Subclass3DDlg function 2580
Forms.Subclass3DWnd 2580
Forms.Subclass3DWnd function 2580
Forms.TActiveFormBorderStyle 2582
Forms.TActiveFormBorderStyle enumeration 2582
Forms.TApplication 1924
Forms.TApplication.ActionUpdateDelay 1968
Forms.TApplication.ActivateHint 1930
Forms.TApplication.Active 1968
Forms.TApplication.ActiveFormHandle 1968
Forms.TApplication.AddPopupForm 1931
Forms.TApplication.AllowTesting 1969
Forms.TApplication.AutoDragDocking 1969
Forms.TApplication.BiDiKeyboard 1969
Forms.TApplication.BiDiMode 1970
Forms.TApplication.BringToFront 1931
Forms.TApplication.CancelHint 1931
Forms.TApplication.ControlDestroyed 1931
Forms.TApplication.Create 1928
Forms.TApplication.CreateForm 1932
Forms.TApplication.CreateHandle 1934
Forms.TApplication.CurrentHelpFile 1970
Forms.TApplication.DefaultFont 1971
Forms.TApplication.Destroy 1928
Forms.TApplication.DialogHandle 1971
Forms.TApplication.DoApplicationIdle 1935
Forms.TApplication.EnumAllWindowsOnActivateHint 1972
Forms.TApplication.ExecuteAction 1935
Forms.TApplication.ExeName 1972
Forms.TApplication.Handle 1974
Forms.TApplication.HandleException 1935
Forms.TApplication.HandleMessage 1937
Forms.TApplication.HelpCommand 1941
Forms.TApplication.HelpContext 1941
Forms.TApplication.HelpFile 1974
Forms.TApplication.HelpJump 1942
Forms.TApplication.HelpKeyword 1942
Forms.TApplication.HelpShowTableOfContents 1943
Forms.TApplication.HelpSystem 1976
Forms.TApplication.HideHint 1943
Forms.TApplication.Hint 1976
Forms.TApplication.HintColor 1979
Forms.TApplication.HintHidePause 1980
Forms.TApplication.HintMouseMessage 1944
Forms.TApplication.HintPause 1980
Forms.TApplication.HintShortCuts 1980
Forms.TApplication.HintShortPause 1981
Forms.TApplication.HookMainWindow 1944

Forms.TApplication.HookSynchronizeWakeup 1944
Forms.TApplication.Icon 1981
Forms.TApplication.Idle 1945
Forms.TApplication.Initialize 1945
Forms.TApplication.IsDlgMsg 1947
Forms.TApplication.IsHintMsg 1947
Forms.TApplication.IsKeyMsg 1948
Forms.TApplication.IsMDIMsg 1948
Forms.TApplication.IsPreProcessMessage 1948
Forms.TApplication.IsRightToLeft 1948
Forms.TApplication.IsShortCut 1949
Forms.TApplication.MainForm 1983
Forms.TApplication.MainFormHandle 1983
Forms.TApplication.MainFormOnTaskBar 1983
Forms.TApplication.MessageBox 1949
Forms.TApplication.Minimize 1951
Forms.TApplication.ModalFinished 1952
Forms.TApplication.ModalLevel 1985
Forms.TApplication.ModalPopupMode 1985
Forms.TApplication.ModalStarted 1952
Forms.TApplication.NonBiDiKeyboard 1985
Forms.TApplication.NormalizeAllTopMosts 1952
Forms.TApplication.NormalizeTopMosts 1953
Forms.TApplication.OnActionExecute 1990
Forms.TApplication.OnActionUpdate 1991
Forms.TApplication.OnActivate 1992
Forms.TApplication.OnDeactivate 1992
Forms.TApplication.OnException 1993
Forms.TApplication.OnGetActiveFormHandle 1995
Forms.TApplication.OnGetMainFormHandle 1995
Forms.TApplication.OnHelp 1995
Forms.TApplication.OnHint 1997
Forms.TApplication.OnIdle 1998
Forms.TApplication.OnMessage 2003
Forms.TApplication.OnMinimize 2004
Forms.TApplication.OnModalBegin 2006
Forms.TApplication.OnModalEnd 2006
Forms.TApplication.OnRestore 2006
Forms.TApplication.OnSettingChange 2007
Forms.TApplication.OnShortCut 2007
Forms.TApplication.OnShowHint 2008
Forms.TApplication.PopupControlWnd 1986
Forms.TApplication.ProcessMessages 1954
Forms.TApplication.RemovePopupForm 1956
Forms.TApplication.Restore 1957
Forms.TApplication.RestoreTopMosts 1958
Forms.TApplication.Run 1959
Forms.TApplication.ShowException 1961
Forms.TApplication.ShowHint 1986
Forms.TApplication.ShowMainForm 1987
Forms.TApplication.Terminate 1962
Forms.TApplication.Terminated 1988
Forms.TApplication.Title 1988
Forms.TApplication.UnhookMainWindow 1964
Forms.TApplication.UnhookSynchronizeWakeup 1964
Forms.TApplication.UpdateAction 1965
Forms.TApplication.UpdateFormatSettings 1989
Forms.TApplication.UpdateMetricSettings 1989
Forms.TApplication.UseRightToLeftAlignment 1966
Forms.TApplication.UseRightToLeftReading 1966
Forms.TApplication.UseRightToLeftScrollBar 1966
Forms.TBorderIcon 2583
Forms.TBorderIcon enumeration 2583
Forms.TBorderIcons 2596
Forms.TBorderIcons type 2596
Forms.TBorderStyle 2597
Forms.TBorderStyle type 2597
Forms.TCloseAction 2584
Forms.TCloseAction enumeration 2584
Forms.TCloseEvent 2597
Forms.TCloseEvent type 2597
Forms.TCloseQueryEvent 2598
Forms.TCloseQueryEvent type 2598
Forms.TCMHintShowPause 2584
Forms.TCMHintShowPause record 2584
Forms.TControlScrollBar 2009
Forms.TControlScrollBar.Assign 2011
Forms.TControlScrollBar.ButtonSize 2012

Forms.TControlScrollBar.ChangeBiDiPosition 2011
Forms.TControlScrollBar.Color 2013
Forms.TControlScrollBar.Increment 2013
Forms.TControlScrollBar.IsScrollBarVisible 2011
Forms.TControlScrollBar.Kind 2015
Forms.TControlScrollBar.Margin 2015
Forms.TControlScrollBar.ParentColor 2016
Forms.TControlScrollBar.Position 2017
Forms.TControlScrollBar.Range 2018
Forms.TControlScrollBar.ScrollPos 2020
Forms.TControlScrollBar.Size 2021
Forms.TControlScrollBar.Smooth 2021
Forms.TControlScrollBar.Style 2022
Forms.TControlScrollBar.ThumbSize 2022
Forms.TControlScrollBar.Tracking 2022
Forms.TControlScrollBar.Visible 2022
Forms.TCursorRec 2585
Forms.TCursorRec record 2585
Forms.TCursorRecType 2598
Forms.TCursorRecType type 2598
Forms.TCustomActiveForm 2024
Forms.TCustomActiveForm.ActiveControl 2031
Forms.TCustomActiveForm.Anchors 2032
Forms.TCustomActiveForm.AutoScroll 2033
Forms.TCustomActiveForm.AutoSize 2033
Forms.TCustomActiveForm.AxBorderStyle 2033
Forms.TCustomActiveForm.BorderWidth 2034
Forms.TCustomActiveForm.Caption 2034
Forms.TCustomActiveForm.Color 2037
Forms.TCustomActiveForm.Constraints 2038
Forms.TCustomActiveForm.Create 2029
Forms.TCustomActiveForm.CreateParams 2029
Forms.TCustomActiveForm.Font 2038
Forms.TCustomActiveForm.Height 2039
Forms.TCustomActiveForm.HorzScrollBar 2040
Forms.TCustomActiveForm.KeyPreview 2041
Forms.TCustomActiveForm.OldCreateOrder 2045
Forms.TCustomActiveForm.OnActivate 2058
Forms.TCustomActiveForm.OnClick 2064
Forms.TCustomActiveForm.OnContextPopup 2066
Forms.TCustomActiveForm.OnCreate 2066
Forms.TCustomActiveForm.OnDblClick 2068
Forms.TCustomActiveForm.OnDeactivate 2068
Forms.TCustomActiveForm.OnDestroy 2069
Forms.TCustomActiveForm.OnDragDrop 2070
Forms.TCustomActiveForm.OnDragOver 2073
Forms.TCustomActiveForm.OnKeyDown 2076
Forms.TCustomActiveForm.OnKeyPress 2080
Forms.TCustomActiveForm.OnKeyUp 2081
Forms.TCustomActiveForm.OnMouseActivate 2083
Forms.TCustomActiveForm.OnMouseDown 2084
Forms.TCustomActiveForm.OnMouseEnter 2086
Forms.TCustomActiveForm.OnMouseLeave 2086
Forms.TCustomActiveForm.OnMouseMove 2086
Forms.TCustomActiveForm.OnMouseUp 2089
Forms.TCustomActiveForm.OnPaint 2091
Forms.TCustomActiveForm.Padding 2045
Forms.TCustomActiveForm.PixelsPerInch 2046
Forms.TCustomActiveForm.PopupMenu 2046
Forms.TCustomActiveForm.PrintScale 2051
Forms.TCustomActiveForm.Scaled 2052
Forms.TCustomActiveForm.ShowHint 2052
Forms.TCustomActiveForm.VertScrollBar 2054
Forms.TCustomActiveForm.Visible 2054
Forms.TCustomActiveForm.WantChildKey 2030
Forms.TCustomActiveForm.Width 2055
Forms.TCustomDockForm 2092
Forms.TCustomDockForm.AutoScroll 2099
Forms.TCustomDockForm.BorderStyle 2100
Forms.TCustomDockForm.Create 2097
Forms.TCustomDockForm.DoAddDockClient 2097
Forms.TCustomDockForm.DoRemoveDockClient 2098
Forms.TCustomDockForm.FormStyle 2100
Forms.TCustomDockForm.GetSiteInfo 2098
Forms.TCustomDockForm.Loaded 2099
Forms.TCustomDockForm.PixelsPerInch 2103
Forms.TCustomForm 2103
Forms.TCustomForm.Action 2144

Forms.TCustomForm.Activate 2111
Forms.TCustomForm.Active 2145
Forms.TCustomForm.ActiveChanged 2112
Forms.TCustomForm.ActiveControl 2145
Forms.TCustomForm.ActiveMDIChild 2146
Forms.TCustomForm.ActiveOleControl 2146
Forms.TCustomForm.AfterConstruction 2112
Forms.TCustomForm.AlignControls 2113
Forms.TCustomForm.AlphaBlend 2147
Forms.TCustomForm.AlphaBlendValue 2147
Forms.TCustomForm.AutoScroll 2148
Forms.TCustomForm.BeforeDestruction 2113
Forms.TCustomForm.BeginAutoDrag 2113
Forms.TCustomForm.BorderIcons 2148
Forms.TCustomForm.BorderStyle 2149
Forms.TCustomForm.Canvas 2150
Forms.TCustomForm.Caption 2151
Forms.TCustomForm.ChangeScale 2114
Forms.TCustomForm.ClientHandle 2154
Forms.TCustomForm.ClientHeight 2154
Forms.TCustomForm.ClientWidth 2155
Forms.TCustomForm.Close 2114
Forms.TCustomForm.CloseModal 2115
Forms.TCustomForm.CloseQuery 2115
Forms.TCustomForm.Color 2157
Forms.TCustomForm.Create 2108
Forms.TCustomForm.CreateNew 2108
Forms.TCustomForm.CreateParams 2116
Forms.TCustomForm.CreateWindowHandle 2116
Forms.TCustomForm.CreateWnd 2117
Forms.TCustomForm.Ctl3D 2158
Forms.TCustomForm.Deactivate 2117
Forms.TCustomForm.DefaultHandler 2118
Forms.TCustomForm.DefaultMonitor 2158
Forms.TCustomForm.DefineProperties 2118
Forms.TCustomForm.DefocusControl 2118
Forms.TCustomForm.Designer 2159
Forms.TCustomForm.Destroy 2109
Forms.TCustomForm.DestroyHandle 2119
Forms.TCustomForm.DestroyWindowHandle 2119
Forms.TCustomForm.Dock 2120
Forms.TCustomForm.DoClose 2120
Forms.TCustomForm.DoCreate 2120
Forms.TCustomForm.DoDestroy 2121
Forms.TCustomForm.DoDock 2121
Forms.TCustomForm.DoHide 2122
Forms.TCustomForm.DoShow 2122
Forms.TCustomForm.DropTarget 2159
Forms.TCustomForm.FFormState 2110
Forms.TCustomForm.FocusControl 2122
Forms.TCustomForm.Font 2160
Forms.TCustomForm.FormState 2161
Forms.TCustomForm.FormStyle 2162
Forms.TCustomForm.get_ActiveMDIChild 2123
Forms.TCustomForm.get_MDIChildCount 2123
Forms.TCustomForm.get_MDIChildren 2123
Forms.TCustomForm.GetBorderIconStyles 2123
Forms.TCustomForm.GetBorderStyles 2124
Forms.TCustomForm.GetChildren 2124
Forms.TCustomForm.GetClientRect 2124
Forms.TCustomForm.GetFloating 2125
Forms.TCustomForm.GetFormImage 2125
Forms.TCustomForm.GetOwnerWindow 2127
Forms.TCustomForm.GlassFrame 2164
Forms.TCustomForm.HandleCreateException 2127
Forms.TCustomForm.Height 2164
Forms.TCustomForm.HelpFile 2165
Forms.TCustomForm.Hide 2127
Forms.TCustomForm.HorzScrollBar 2166
Forms.TCustomForm.Icon 2167
Forms.TCustomForm.InitializeNewForm 2128
Forms.TCustomForm.IsShortCut 2128
Forms.TCustomForm.KeyPreview 2167
Forms.TCustomForm.Left 2171
Forms.TCustomForm.Loaded 2129
Forms.TCustomForm.MakeFullyVisible 2129
Forms.TCustomForm.MDIChildCount 2172
Forms.TCustomForm.MDIChildren 2173

Forms.TCustomForm.Menu 2174
Forms.TCustomForm.ModalResult 2176
Forms.TCustomForm.Monitor 2177
Forms.TCustomForm.MouseWheelHandler 2130
Forms.TCustomForm.Notification 2130
Forms.TCustomForm.ObjectMenuItem 2177
Forms.TCustomForm.OldCreateOrder 2178
Forms.TCustomForm.OleFormObject 2178
Forms.TCustomForm.OnActivate 2196
Forms.TCustomForm.OnCanResize 2202
Forms.TCustomForm.OnClick 2203
Forms.TCustomForm.OnClose 2205
Forms.TCustomForm.OnCloseQuery 2206
Forms.TCustomForm.OnCreate 2207
Forms.TCustomForm.OnDblClick 2209
Forms.TCustomForm.OnDeactivate 2209
Forms.TCustomForm.OnDestroy 2209
Forms.TCustomForm.OnDragDrop 2211
Forms.TCustomForm.OnDragOver 2214
Forms.TCustomForm.OnHelp 2217
Forms.TCustomForm.OnHide 2217
Forms.TCustomForm.OnKeyDown 2219
Forms.TCustomForm.OnKeyPress 2223
Forms.TCustomForm.OnKeyUp 2224
Forms.TCustomForm.OnMouseActivate 2226
Forms.TCustomForm.OnMouseDown 2227
Forms.TCustomForm.OnMouseMove 2229
Forms.TCustomForm.OnMouseUp 2231
Forms.TCustomForm.OnPaint 2233
Forms.TCustomForm.OnResize 2235
Forms.TCustomForm.OnShortCut 2235
Forms.TCustomForm.OnShow 2236
Forms.TCustomForm.Paint 2131
Forms.TCustomForm.PaintWindow 2131
Forms.TCustomForm.PaletteChanged 2132
Forms.TCustomForm.ParentFont 2179
Forms.TCustomForm.PixelsPerInch 2179
Forms.TCustomForm.PopupChildren 2179
Forms.TCustomForm.PopupMenu 2180
Forms.TCustomForm.PopupMode 2185
Forms.TCustomForm.PopupParent 2185
Forms.TCustomForm.Position 2185
Forms.TCustomForm.Print 2132
Forms.TCustomForm.PrintScale 2186
Forms.TCustomForm.QueryInterface 2132
Forms.TCustomForm.ReadState 2133
Forms.TCustomForm.RecreateAsPopup 2133
Forms.TCustomForm.Release 2133
Forms.TCustomForm.RequestAlign 2134
Forms.TCustomForm.Resizing 2134
Forms.TCustomForm.Scaled 2187
Forms.TCustomForm.ScreenSnap 2187
Forms.TCustomForm.SendCancelMode 2134
Forms.TCustomForm.set_PopupParent 2135
Forms.TCustomForm.SetChildOrder 2135
Forms.TCustomForm.SetFocus 2135
Forms.TCustomForm.SetFocusedControl 2136
Forms.TCustomForm.SetParent 2136
Forms.TCustomForm.SetParentBiDiMode 2136
Forms.TCustomForm.Show 2137
Forms.TCustomForm.ShowModal 2139
Forms.TCustomForm.SnapBuffer 2188
Forms.TCustomForm.TileMode 2188
Forms.TCustomForm.Top 2189
Forms.TCustomForm.TransparentColor 2190
Forms.TCustomForm.TransparentColorValue 2191
Forms.TCustomForm.UpdateActions 2140
Forms.TCustomForm.UpdateWindowState 2141
Forms.TCustomForm.ValidateRename 2141
Forms.TCustomForm.VertScrollBar 2191
Forms.TCustomForm.Visible 2192
Forms.TCustomForm.VisibleChanging 2141
Forms.TCustomForm.WantChildKey 2142
Forms.TCustomForm.Width 2193
Forms.TCustomForm.WindowMenu 2194
Forms.TCustomForm.WindowState 2195
Forms.TCustomForm.WndProc 2142
Forms.TCustomFormClass 2599

Forms.TCustomFormClass type 2599
Forms.TCustomFrame 2237
Forms.TCustomFrame.Create 2238
Forms.TCustomFrame.CreateParams 2239
Forms.TCustomFrame.GetChildren 2239
Forms.TCustomFrame.PaintWindow 2240
Forms.TCustomFrame.SetParent 2240
Forms.TCustomFrameClass 2599
Forms.TCustomFrameClass type 2599
Forms.TDefaultMonitor 2585
Forms.TDefaultMonitor enumeration 2585
Forms.TExceptionEvent 2599
Forms.TExceptionEvent type 2599
Forms.TFocusState 2600
Forms.TFocusState type 2600
Forms.TForm 2241
Forms.TForm.Action 2252
Forms.TForm.ActiveControl 2253
Forms.TForm.ActiveMDIChild 2254
Forms.TForm.Align 2254
Forms.TForm.AlphaBlend 2257
Forms.TForm.AlphaBlendValue 2258
Forms.TForm.Anchors 2258
Forms.TForm.ArrangeIcons 2247
Forms.TForm.AutoScroll 2259
Forms.TForm.AutoSize 2259
Forms.TForm.BiDiMode 2260
Forms.TForm.BorderIcons 2260
Forms.TForm.BorderStyle 2261
Forms.TForm.BorderWidth 2262
Forms.TForm.Caption 2262
Forms.TForm.Cascade 2248
Forms.TForm.ClientHandle 2264
Forms.TForm.ClientHeight 2265
Forms.TForm.ClientWidth 2266
Forms.TForm.Color 2268
Forms.TForm.Constraints 2269
Forms.TForm.Ctl3D 2269
Forms.TForm.DefaultMonitor 2270
Forms.TForm.DockManager 2270
Forms.TForm.DockSite 2272
Forms.TForm.DoubleBuffered 2273
Forms.TForm.DragKind 2275
Forms.TForm.DragMode 2275
Forms.TForm.Enabled 2277
Forms.TForm.Font 2277
Forms.TForm.FormStyle 2279
Forms.TForm.GlassFrame 2281
Forms.TForm.Height 2281
Forms.TForm.HelpFile 2282
Forms.TForm.HorzScrollBar 2283
Forms.TForm.Icon 2284
Forms.TForm.KeyPreview 2284
Forms.TForm.MDIChildCount 2288
Forms.TForm.MDIChildren 2289
Forms.TForm.Menu 2290
Forms.TForm.Next 2249
Forms.TForm.ObjectMenuItem 2292
Forms.TForm.OldCreateOrder 2292
Forms.TForm.OnActivate 2312
Forms.TForm.OnAlignInsertBefore 2318
Forms.TForm.OnAlignPosition 2318
Forms.TForm.OnCanResize 2319
Forms.TForm.OnClick 2319
Forms.TForm.OnClose 2321
Forms.TForm.OnCloseQuery 2323
Forms.TForm.OnConstrainedResize 2324
Forms.TForm.OnContextPopup 2324
Forms.TForm.OnCreate 2325
Forms.TForm.OnDblClick 2327
Forms.TForm.OnDeactivate 2327
Forms.TForm.OnDestroy 2327
Forms.TForm.OnDockDrop 2329
Forms.TForm.OnDockOver 2331
Forms.TForm.OnDragDrop 2333
Forms.TForm.OnDragOver 2336
Forms.TForm.OnEndDock 2339
Forms.TForm.OnGetSiteInfo 2339

Forms.TForm.OnHelp 2341
Forms.TForm.OnHide 2342
Forms.TForm.OnKeyDown 2344
Forms.TForm.OnKeyPress 2347
Forms.TForm.OnKeyUp 2348
Forms.TForm.OnMouseActivate 2350
Forms.TForm.OnMouseDown 2351
Forms.TForm.OnMouseEnter 2353
Forms.TForm.OnMouseLeave 2354
Forms.TForm.OnMouseMove 2354
Forms.TForm.OnMouseUp 2356
Forms.TForm.OnMouseWheel 2359
Forms.TForm.OnMouseWheelDown 2359
Forms.TForm.OnMouseWheelUp 2359
Forms.TForm.OnPaint 2360
Forms.TForm.OnResize 2361
Forms.TForm.OnShortCut 2362
Forms.TForm.OnShow 2362
Forms.TForm.OnStartDock 2363
Forms.TForm.OnUnDock 2364
Forms.TForm.Padding 2292
Forms.TForm.ParentBiDiMode 2293
Forms.TForm.ParentFont 2293
Forms.TForm.PixelsPerInch 2294
Forms.TForm.PopupMenu 2294
Forms.TForm.PopupMode 2299
Forms.TForm.PopupParent 2300
Forms.TForm.Position 2300
Forms.TForm.Previous 2249
Forms.TForm.PrintScale 2301
Forms.TForm.Scaled 2302
Forms.TForm.ScreenSnap 2302
Forms.TForm.ShowHint 2302
Forms.TForm.SnapBuffer 2304
Forms.TForm.Tile 2249
Forms.TForm.TileMode 2304
Forms.TForm.TransparentColor 2305
Forms.TForm.TransparentColorValue 2306
Forms.TForm.UseDockManager 2306
Forms.TForm.VertScrollBar 2306
Forms.TForm.Visible 2307
Forms.TForm.Width 2308
Forms.TForm.WindowMenu 2310
Forms.TForm.WindowState 2310
Forms.TFormBorderStyle 2586
Forms.TFormBorderStyle enumeration 2586
Forms.TFormClass 2600
Forms.TFormClass type 2600
Forms.TFormState 2601
Forms.TFormState type 2601
Forms.TFormStyle 2587
Forms.TFormStyle enumeration 2587
Forms.TFrame 2366
Forms.TFrame.Align 2370
Forms.TFrame.Anchors 2373
Forms.TFrame.AutoScroll 2374
Forms.TFrame.AutoSize 2374
Forms.TFrame.BiDiMode 2375
Forms.TFrame.Color 2375
Forms.TFrame.Constraints 2376
Forms.TFrame.Ctl3D 2376
Forms.TFrame.DockSite 2377
Forms.TFrame.DoubleBuffered 2378
Forms.TFrame.DragCursor 2380
Forms.TFrame.DragKind 2380
Forms.TFrame.DragMode 2380
Forms.TFrame.Enabled 2382
Forms.TFrame.Font 2382
Forms.TFrame.OnAlignInsertBefore 2397
Forms.TFrame.OnAlignPosition 2397
Forms.TFrame.OnCanResize 2398
Forms.TFrame.OnClick 2398
Forms.TFrame.OnConstrainedResize 2400
Forms.TFrame.OnContextPopup 2401
Forms.TFrame.OnDblClick 2401
Forms.TFrame.OnDockDrop 2402
Forms.TFrame.OnDockOver 2404
Forms.TFrame.OnDragDrop 2406

Forms.TFrame.OnDragOver 2409
Forms.TFrame.OnEndDock 2412
Forms.TFrame.OnEndDrag 2412
Forms.TFrame.OnEnter 2413
Forms.TFrame.OnExit 2416
Forms.TFrame.OnGetSiteInfo 2419
Forms.TFrame.OnMouseActivate 2421
Forms.TFrame.OnMouseDown 2422
Forms.TFrame.OnMouseEnter 2424
Forms.TFrame.OnMouseLeave 2424
Forms.TFrame.OnMouseMove 2424
Forms.TFrame.OnMouseUp 2427
Forms.TFrame.OnMouseWheel 2429
Forms.TFrame.OnMouseWheelDown 2429
Forms.TFrame.OnMouseWheelUp 2430
Forms.TFrame.OnResize 2430
Forms.TFrame.OnStartDock 2431
Forms.TFrame.OnStartDrag 2431
Forms.TFrame.OnUnDock 2432
Forms.TFrame.Padding 2384
Forms.TFrame.ParentBackground 2384
Forms.TFrame.ParentBiDiMode 2384
Forms.TFrame.ParentColor 2385
Forms.TFrame.ParentCtl3D 2385
Forms.TFrame.ParentDoubleBuffered 2385
Forms.TFrame.ParentFont 2386
Forms.TFrame.ParentShowHint 2386
Forms.TFrame.PopupMenu 2387
Forms.TFrame.ShowHint 2392
Forms.TFrame.TabOrder 2393
Forms.TFrame.TabStop 2394
Forms.TFrame.Visible 2394
Forms.TGetHandleEvent 2601
Forms.TGetHandleEvent type 2601
Forms.TGlassFrame 2435
Forms.TGlassFrame.Assign 2436
Forms.TGlassFrame.Bottom 2438
Forms.TGlassFrame.Change 2437
Forms.TGlassFrame.Create 2436
Forms.TGlassFrame.Enabled 2438
Forms.TGlassFrame.FrameExtended 2437
Forms.TGlassFrame.IntersectsControl 2438
Forms.TGlassFrame.Left 2438
Forms.TGlassFrame.OnChange 2439
Forms.TGlassFrame.Right 2439
Forms.TGlassFrame.SheetOfGlass 2439
Forms.TGlassFrame.Top 2439
Forms.THelpEvent 2602
Forms.THelpEvent type 2602
Forms.THintInfo 2602
Forms.THintInfo type 2602
Forms.TIdleEvent 2604
Forms.TIdleEvent type 2604
Forms.TMessageEvent 2605
Forms.TMessageEvent type 2605
Forms.TMonitor 2439
Forms.TMonitor.BoundsRect 2441
Forms.TMonitor.Handle 2441
Forms.TMonitor.Height 2442
Forms.TMonitor.Left 2442
Forms.TMonitor.MonitorNum 2443
Forms.TMonitor.Primary 2443
Forms.TMonitor.Top 2443
Forms.TMonitor.Width 2444
Forms.TMonitor.WorkareaRect 2444
Forms.TMonitorDefaultTo 2587
Forms.TMonitorDefaultTo enumeration 2587
Forms.TPopupForm 2588
Forms.TPopupForm record 2588
Forms.TPopupFormArray 2605
Forms.TPopupFormArray type 2605
Forms.TPopupMode 2588
Forms.TPopupMode enumeration 2588
Forms.TPopupWnd 2589
Forms.TPopupWnd record 2589
Forms.TPopupWndArray 2605
Forms.TPopupWndArray type 2605
Forms.TPosition 2590

Forms.TPosition enumeration 2590
Forms.TPrintScale 2591
Forms.TPrintScale enumeration 2591
Forms.TScreen 2445
Forms.TScreen.ActiveControl 2452
Forms.TScreen.ActiveCustomForm 2452
Forms.TScreen.ActiveForm 2453
Forms.TScreen.Create 2447
Forms.TScreen.Cursor 2455
Forms.TScreen.CursorCount 2458
Forms.TScreen.Cursors 2458
Forms.TScreen.CustomFormCount 2460
Forms.TScreen.CustomForms 2460
Forms.TScreen.DataModuleCount 2461
Forms.TScreen.DataModules 2462
Forms.TScreen.DefaultIme 2462
Forms.TScreen.DefaultKbLayout 2463
Forms.TScreen.DesktopHeight 2463
Forms.TScreen.DesktopLeft 2463
Forms.TScreen.DesktopRect 2464
Forms.TScreen.DesktopTop 2465
Forms.TScreen.DesktopWidth 2465
Forms.TScreen.Destroy 2448
Forms.TScreen.DisableAlign 2448
Forms.TScreen.EnableAlign 2449
Forms.TScreen.FocusedForm 2466
Forms.TScreen.Fonts 2466
Forms.TScreen.FormCount 2467
Forms.TScreen.Forms 2468
Forms.TScreen.Height 2470
Forms.TScreen.HintFont 2471
Forms.TScreen.IconFont 2472
Forms.TScreen.Imes 2472
Forms.TScreen.MenuFont 2473
Forms.TScreen.MessageFont 2473
Forms.TScreen.MonitorCount 2473
Forms.TScreen.MonitorFromPoint 2449
Forms.TScreen.MonitorFromRect 2449
Forms.TScreen.MonitorFromWindow 2450
Forms.TScreen.Monitors 2474
Forms.TScreen.OnActiveControlChange 2479
Forms.TScreen.OnActiveFormChange 2481
Forms.TScreen.PixelsPerInch 2474
Forms.TScreen.PrimaryMonitor 2474
Forms.TScreen.Realign 2450
Forms.TScreen.ResetFonts 2451
Forms.TScreen.SaveFocusedList 2475
Forms.TScreen.Width 2475
Forms.TScreen.WorkAreaHeight 2476
Forms.TScreen.WorkAreaLeft 2477
Forms.TScreen.WorkAreaRect 2477
Forms.TScreen.WorkAreaTop 2478
Forms.TScreen.WorkAreaWidth 2478
Forms.TScrollBarInc 2606
Forms.TScrollBarInc type 2606
Forms.TScrollBarKind 2591
Forms.TScrollBarKind enumeration 2591
Forms.TScrollBarStyle 2592
Forms.TScrollBarStyle enumeration 2592
Forms.TScrollBox 2483
Forms.TScrollBox.Align 2488
Forms.TScrollBox.Anchors 2492
Forms.TScrollBox.AutoScroll 2492
Forms.TScrollBox.AutoSize 2493
Forms.TScrollBox.BevelEdges 2493
Forms.TScrollBox.BevelInner 2493
Forms.TScrollBox.BevelKind 2494
Forms.TScrollBox.BevelOuter 2494
Forms.TScrollBox.BevelWidth 2495
Forms.TScrollBox.BiDiMode 2495
Forms.TScrollBox.BorderStyle 2496
Forms.TScrollBox.Color 2496
Forms.TScrollBox.Constraints 2497
Forms.TScrollBox.Create 2486
Forms.TScrollBox.CreateParams 2487
Forms.TScrollBox.Ctl3D 2497
Forms.TScrollBox.DockSite 2498
Forms.TScrollBox.DoubleBuffered 2499

Forms.TScrollBox.DragCursor 2501
Forms.TScrollBox.DragKind 2501
Forms.TScrollBox.DragMode 2501
Forms.TScrollBox.Enabled 2503
Forms.TScrollBox.Font 2503
Forms.TScrollBox.OnCanResize 2517
Forms.TScrollBox.OnClick 2518
Forms.TScrollBox.OnConstrainedResize 2520
Forms.TScrollBox.OnContextPopup 2520
Forms.TScrollBox.OnDblClick 2521
Forms.TScrollBox.OnDockDrop 2521
Forms.TScrollBox.OnDockOver 2523
Forms.TScrollBox.OnDragDrop 2525
Forms.TScrollBox.OnDragOver 2528
Forms.TScrollBox.OnEndDock 2531
Forms.TScrollBox.OnEndDrag 2532
Forms.TScrollBox.OnEnter 2533
Forms.TScrollBox.OnExit 2535
Forms.TScrollBox.OnGetSiteInfo 2538
Forms.TScrollBox.OnMouseActivate 2540
Forms.TScrollBox.OnMouseDown 2541
Forms.TScrollBox.OnMouseEnter 2543
Forms.TScrollBox.OnMouseLeave 2544
Forms.TScrollBox.OnMouseMove 2544
Forms.TScrollBox.OnMouseUp 2546
Forms.TScrollBox.OnMouseWheel 2548
Forms.TScrollBox.OnMouseWheelDown 2549
Forms.TScrollBox.OnMouseWheelUp 2549
Forms.TScrollBox.OnResize 2549
Forms.TScrollBox.OnStartDock 2550
Forms.TScrollBox.OnStartDrag 2551
Forms.TScrollBox.OnUnDock 2551
Forms.TScrollBox.Padding 2504
Forms.TScrollBox.PaintWindow 2487
Forms.TScrollBox.ParentBackground 2505
Forms.TScrollBox.ParentBiDiMode 2505
Forms.TScrollBox.ParentColor 2505
Forms.TScrollBox.ParentCtl3D 2506
Forms.TScrollBox.ParentDoubleBuffered 2506
Forms.TScrollBox.ParentFont 2506
Forms.TScrollBox.ParentShowHint 2507
Forms.TScrollBox.PopupMenu 2507
Forms.TScrollBox.ShowHint 2512
Forms.TScrollBox.TabOrder 2514
Forms.TScrollBox.TabStop 2515
Forms.TScrollBox.Visible 2515
Forms.TScrollingWinControl 2554
Forms.TScrollingWinControl.AdjustClientRect 2557
Forms.TScrollingWinControl.AlignControls 2557
Forms.TScrollingWinControl.AutoScroll 2561
Forms.TScrollingWinControl.AutoScrollEnabled 2558
Forms.TScrollingWinControl.AutoScrollInView 2558
Forms.TScrollingWinControl.ChangeScale 2558
Forms.TScrollingWinControl.Create 2555
Forms.TScrollingWinControl.CreateParams 2559
Forms.TScrollingWinControl.CreateWnd 2559
Forms.TScrollingWinControl.Destroy 2556
Forms.TScrollingWinControl.DisableAutoRange 2559
Forms.TScrollingWinControl.DoFlipChildren 2560
Forms.TScrollingWinControl.EnableAutoRange 2560
Forms.TScrollingWinControl.HorzScrollBar 2562
Forms.TScrollingWinControl.OnAlignInsertBefore 2564
Forms.TScrollingWinControl.OnAlignPosition 2564
Forms.TScrollingWinControl.Resizing 2561
Forms.TScrollingWinControl.ScrollInView 2561
Forms.TScrollingWinControl.VertScrollBar 2563
Forms.TSetLayeredWindowAttributes 2606
Forms.TSetLayeredWindowAttributes type 2606
Forms.TSettingChangeEvent 2607
Forms.TSettingChangeEvent type 2607
Forms.TShortCutEvent 2609
Forms.TShortCutEvent type 2609
Forms.TShowAction 2592
Forms.TShowAction enumeration 2592
Forms.TShowHintEvent 2610
Forms.TShowHintEvent type 2610
Forms.TTaskWindowList 2610
Forms.TTaskWindowList type 2610

Forms.TTileMode 2593
Forms.TTileMode enumeration 2593
Forms.TTimerMode 2593
Forms.TTimerMode enumeration 2593
Forms.TWindowHook 2611
Forms.TWindowHook type 2611
Forms.TWindowState 2594
Forms.TWindowState enumeration 2594
Forms.ValidParentForm 2580
Forms.ValidParentForm function 2580
fpDefault enumeration member 2866
fpFixed enumeration member 2866
fpVariable enumeration member 2866
fsBold enumeration member 2866
fsBorder enumeration member 2864
fsBottomTopLeftRight enumeration member 1543
fsBottomTopRightLeft enumeration member 1543
fsItalic enumeration member 2866
fsLeftRightBottomTop enumeration member 1543
fsLeftRightTopBottom enumeration member 1543
fsMDIChild enumeration member 2587
fsMDIForm enumeration member 2587
fsNormal enumeration member 2587
fsRightLeftBottomTop enumeration member 1543
fsRightLeftTopBottom enumeration member 1543
fsStayOnTop enumeration member 2587
fsStrikeOut enumeration member 2866
fsSurface enumeration member 2864
fsTopBottomLeftRight enumeration member 1543
fsTopBottomRightLeft enumeration member 1543
fsUnderline enumeration member 2866
ftArchive enumeration member 1883
ftDirectory enumeration member 1883
ftHidden enumeration member 1883
ftNormal enumeration member 1883
ftReadOnly enumeration member 1883
ftSystem enumeration member 1883
ftVolumeID enumeration member 1883

G

gaHorizontal enumeration member 6774
gamOnNewRecord enumeration member 3691
gamOnPost enumeration member 3691
gamOnServer enumeration member 3691
gaVertical enumeration member 6774
gdHorizontal enumeration member 2990
gdVertical enumeration member 2990
goAlwaysShowEditor enumeration member 3201
goColMoving enumeration member 3201
goColSizing enumeration member 3201
goDrawFocusSelected enumeration member 3201
goEditing enumeration member 3201
goFixedHorzLine enumeration member 3201
goFixedVertLine enumeration member 3201
goHorzLine enumeration member 3201
goRangeSelect enumeration member 3201
goRowMoving enumeration member 3201
goRowSelect enumeration member 3201
goRowSizing enumeration member 3201
goTabs enumeration member 3201
goThumbTracking enumeration member 3201
goVertLine enumeration member 3201
Graphics 2619
Graphics namespace 2619
    Classes 2627
    Constants 2892
    Functions 2836
    Interfaces 2834
    Structs, Records, Enums 2860
    Types 2875
    Variables 2890
Graphics.AllocPatternBitmap 2836
Graphics.AllocPatternBitmap function 2836
Graphics.BytesPerScanline 2837
Graphics.BytesPerScanline function 2837
Graphics.CharsetToIdent 2837
Graphics.CharsetToIdent function 2837

Graphics.cl3DDkShadow 2896
Graphics.cl3DDkShadow constant 2896
Graphics.cl3DLight 2896
Graphics.cl3DLight constant 2896
Graphics.clActiveBorder 2897
Graphics.clActiveBorder constant 2897
Graphics.clActiveCaption 2897
Graphics.clActiveCaption constant 2897
Graphics.clAppWorkSpace 2898
Graphics.clAppWorkSpace constant 2898
Graphics.clAqua 2898
Graphics.clAqua constant 2898
Graphics.clBackground 2899
Graphics.clBackground constant 2899
Graphics.clBlack 2899
Graphics.clBlack constant 2899
Graphics.clBlue 2899
Graphics.clBlue constant 2899
Graphics.clBtnFace 2900
Graphics.clBtnFace constant 2900
Graphics.clBtnHighlight 2900
Graphics.clBtnHighlight constant 2900
Graphics.clBtnShadow 2901
Graphics.clBtnShadow constant 2901
Graphics.clBtnText 2901
Graphics.clBtnText constant 2901
Graphics.clCaptionText 2902
Graphics.clCaptionText constant 2902
Graphics.clCream 2902
Graphics.clCream constant 2902
Graphics.clDefault 2903
Graphics.clDefault constant 2903
Graphics.clDkGray 2903
Graphics.clDkGray constant 2903
Graphics.clFuchsia 2903
Graphics.clFuchsia constant 2903
Graphics.clGradientActiveCaption 2904
Graphics.clGradientActiveCaption constant 2904
Graphics.clGradientInactiveCaption 2904
Graphics.clGradientInactiveCaption constant 2904
Graphics.clGray 2905
Graphics.clGray constant 2905
Graphics.clGrayText 2905
Graphics.clGrayText constant 2905
Graphics.clGreen 2906
Graphics.clGreen constant 2906
Graphics.clHighlight 2906
Graphics.clHighlight constant 2906
Graphics.clHighlightText 2907
Graphics.clHighlightText constant 2907
Graphics.clHotLight 2907
Graphics.clHotLight constant 2907
Graphics.clInactiveBorder 2908
Graphics.clInactiveBorder constant 2908
Graphics.clInactiveCaption 2908
Graphics.clInactiveCaption constant 2908
Graphics.clInactiveCaptionText 2908
Graphics.clInactiveCaptionText constant 2908
Graphics.clInfoBk 2909
Graphics.clInfoBk constant 2909
Graphics.clInfoText 2909
Graphics.clInfoText constant 2909
Graphics.clLime 2910
Graphics.clLime constant 2910
Graphics.clLtGray 2910
Graphics.clLtGray constant 2910
Graphics.clMaroon 2911
Graphics.clMaroon constant 2911
Graphics.clMedGray 2911
Graphics.clMedGray constant 2911
Graphics.clMenu 2912
Graphics.clMenu constant 2912
Graphics.clMenuBar 2912
Graphics.clMenuBar constant 2912
Graphics.clMenuHighlight 2912
Graphics.clMenuHighlight constant 2912
Graphics.clMenuText 2913
Graphics.clMenuText constant 2913

Graphics.clMoneyGreen 2913
Graphics.clMoneyGreen constant 2913
Graphics.clNavy 2914
Graphics.clNavy constant 2914
Graphics.clNone 2914
Graphics.clNone constant 2914
Graphics.clOlive 2915
Graphics.clOlive constant 2915
Graphics.clPurple 2915
Graphics.clPurple constant 2915
Graphics.clRed 2916
Graphics.clRed constant 2916
Graphics.clScrollBar 2916
Graphics.clScrollBar constant 2916
Graphics.clSilver 2916
Graphics.clSilver constant 2916
Graphics.clSkyBlue 2917
Graphics.clSkyBlue constant 2917
Graphics.clSystemColor 2917
Graphics.clSystemColor constant 2917
Graphics.clTeal 2918
Graphics.clTeal constant 2918
Graphics.clWebAliceBlue 2918
Graphics.clWebAliceBlue constant 2918
Graphics.clWebAntiqueWhite 2919
Graphics.clWebAntiqueWhite constant 2919
Graphics.clWebAqua 2919
Graphics.clWebAqua constant 2919
Graphics.clWebAquamarine 2919
Graphics.clWebAquamarine constant 2919
Graphics.clWebAzure 2920
Graphics.clWebAzure constant 2920
Graphics.clWebBeige 2920
Graphics.clWebBeige constant 2920
Graphics.clWebBisque 2920
Graphics.clWebBisque constant 2920
Graphics.clWebBlack 2921
Graphics.clWebBlack constant 2921
Graphics.clWebBlanchedAlmond 2921
Graphics.clWebBlanchedAlmond constant 2921
Graphics.clWebBlue 2921
Graphics.clWebBlue constant 2921
Graphics.clWebBlueViolet 2922
Graphics.clWebBlueViolet constant 2922
Graphics.clWebBrown 2922
Graphics.clWebBrown constant 2922
Graphics.clWebBurlywood 2922
Graphics.clWebBurlywood constant 2922
Graphics.clWebCadetBlue 2923
Graphics.clWebCadetBlue constant 2923
Graphics.clWebChartreuse 2923
Graphics.clWebChartreuse constant 2923
Graphics.clWebChocolate 2923
Graphics.clWebChocolate constant 2923
Graphics.clWebCoral 2924
Graphics.clWebCoral constant 2924
Graphics.clWebCornFlowerBlue 2924
Graphics.clWebCornFlowerBlue constant 2924
Graphics.clWebCornSilk 2924
Graphics.clWebCornSilk constant 2924
Graphics.clWebCrimson 2925
Graphics.clWebCrimson constant 2925
Graphics.clWebCyan 2925
Graphics.clWebCyan constant 2925
Graphics.clWebDarkBlue 2925
Graphics.clWebDarkBlue constant 2925
Graphics.clWebDarkCyan 2926
Graphics.clWebDarkCyan constant 2926
Graphics.clWebDarkGoldenRod 2926
Graphics.clWebDarkGoldenRod constant 2926
Graphics.clWebDarkGray 2926
Graphics.clWebDarkGray constant 2926
Graphics.clWebDarkgreen 2927
Graphics.clWebDarkgreen constant 2927
Graphics.clWebDarkKhaki 2927
Graphics.clWebDarkKhaki constant 2927
Graphics.clWebDarkMagenta 2927
Graphics.clWebDarkMagenta constant 2927

Graphics.clWebDarkOliveGreen 2928
Graphics.clWebDarkOliveGreen constant 2928
Graphics.clWebDarkOrange 2928
Graphics.clWebDarkOrange constant 2928
Graphics.clWebDarkOrchid 2928
Graphics.clWebDarkOrchid constant 2928
Graphics.clWebDarkRed 2929
Graphics.clWebDarkRed constant 2929
Graphics.clWebDarkSalmon 2929
Graphics.clWebDarkSalmon constant 2929
Graphics.clWebDarkSeaGreen 2929
Graphics.clWebDarkSeaGreen constant 2929
Graphics.clWebDarkSlateBlue 2930
Graphics.clWebDarkSlateBlue constant 2930
Graphics.clWebDarkSlategray 2930
Graphics.clWebDarkSlategray constant 2930
Graphics.clWebDarkTurquoise 2930
Graphics.clWebDarkTurquoise constant 2930
Graphics.clWebDarkViolet 2931
Graphics.clWebDarkViolet constant 2931
Graphics.clWebDeepPink 2931
Graphics.clWebDeepPink constant 2931
Graphics.clWebDeepskyBlue 2931
Graphics.clWebDeepskyBlue constant 2931
Graphics.clWebDimGray 2932
Graphics.clWebDimGray constant 2932
Graphics.clWebDodgerBlue 2932
Graphics.clWebDodgerBlue constant 2932
Graphics.clWebFirebrick 2932
Graphics.clWebFirebrick constant 2932
Graphics.clWebFloralWhite 2933
Graphics.clWebFloralWhite constant 2933
Graphics.clWebForestGreen 2933
Graphics.clWebForestGreen constant 2933
Graphics.clWebFuchsia 2933
Graphics.clWebFuchsia constant 2933
Graphics.clWebGainsboro 2934
Graphics.clWebGainsboro constant 2934
Graphics.clWebGhostWhite 2934
Graphics.clWebGhostWhite constant 2934
Graphics.clWebGold 2934
Graphics.clWebGold constant 2934
Graphics.clWebGoldenRod 2935
Graphics.clWebGoldenRod constant 2935
Graphics.clWebGray 2935
Graphics.clWebGray constant 2935
Graphics.clWebGreen 2935
Graphics.clWebGreen constant 2935
Graphics.clWebGreenYellow 2936
Graphics.clWebGreenYellow constant 2936
Graphics.clWebHoneydew 2936
Graphics.clWebHoneydew constant 2936
Graphics.clWebHotPink 2936
Graphics.clWebHotPink constant 2936
Graphics.clWebIndianRed 2937
Graphics.clWebIndianRed constant 2937
Graphics.clWebIndigo 2937
Graphics.clWebIndigo constant 2937
Graphics.clWebIvory 2937
Graphics.clWebIvory constant 2937
Graphics.clWebKhaki 2938
Graphics.clWebKhaki constant 2938
Graphics.clWebLavender 2938
Graphics.clWebLavender constant 2938
Graphics.clWebLavenderBlush 2938
Graphics.clWebLavenderBlush constant 2938
Graphics.clWebLawnGreen 2939
Graphics.clWebLawnGreen constant 2939
Graphics.clWebLemonChiffon 2939
Graphics.clWebLemonChiffon constant 2939
Graphics.clWebLightBlue 2939
Graphics.clWebLightBlue constant 2939
Graphics.clWebLightCoral 2940
Graphics.clWebLightCoral constant 2940
Graphics.clWebLightCyan 2940
Graphics.clWebLightCyan constant 2940
Graphics.clWebLightGoldenrodYellow 2940
Graphics.clWebLightGoldenrodYellow constant 2940

Graphics.clWebLightGreen 2941
Graphics.clWebLightGreen constant 2941
Graphics.clWebLightgrey 2941
Graphics.clWebLightgrey constant 2941
Graphics.clWebLightPink 2941
Graphics.clWebLightPink constant 2941
Graphics.clWebLightSalmon 2942
Graphics.clWebLightSalmon constant 2942
Graphics.clWebLightSeaGreen 2942
Graphics.clWebLightSeaGreen constant 2942
Graphics.clWebLightSkyBlue 2942
Graphics.clWebLightSkyBlue constant 2942
Graphics.clWebLightSlateGray 2943
Graphics.clWebLightSlateGray constant 2943
Graphics.clWebLightSteelBlue 2943
Graphics.clWebLightSteelBlue constant 2943
Graphics.clWebLightYellow 2943
Graphics.clWebLightYellow constant 2943
Graphics.clWebLime 2944
Graphics.clWebLime constant 2944
Graphics.clWebLimeGreen 2944
Graphics.clWebLimeGreen constant 2944
Graphics.clWebLinen 2944
Graphics.clWebLinen constant 2944
Graphics.clWebMagenta 2945
Graphics.clWebMagenta constant 2945
Graphics.clWebMaroon 2945
Graphics.clWebMaroon constant 2945
Graphics.clWebMediumAquamarine 2945
Graphics.clWebMediumAquamarine constant 2945
Graphics.clWebMediumBlue 2946
Graphics.clWebMediumBlue constant 2946
Graphics.clWebMediumOrchid 2946
Graphics.clWebMediumOrchid constant 2946
Graphics.clWebMediumPurple 2946
Graphics.clWebMediumPurple constant 2946
Graphics.clWebMediumSeaGreen 2947
Graphics.clWebMediumSeaGreen constant 2947
Graphics.clWebMediumSlateBlue 2947
Graphics.clWebMediumSlateBlue constant 2947
Graphics.clWebMediumSpringGreen 2947
Graphics.clWebMediumSpringGreen constant 2947
Graphics.clWebMediumTurquoise 2948
Graphics.clWebMediumTurquoise constant 2948
Graphics.clWebMediumVioletRed 2948
Graphics.clWebMediumVioletRed constant 2948
Graphics.clWebMidnightBlue 2948
Graphics.clWebMidnightBlue constant 2948
Graphics.clWebMintcream 2949
Graphics.clWebMintcream constant 2949
Graphics.clWebMistyRose 2949
Graphics.clWebMistyRose constant 2949
Graphics.clWebMoccasin 2949
Graphics.clWebMoccasin constant 2949
Graphics.clWebNavajoWhite 2950
Graphics.clWebNavajoWhite constant 2950
Graphics.clWebNavy 2950
Graphics.clWebNavy constant 2950
Graphics.clWebOldLace 2950
Graphics.clWebOldLace constant 2950
Graphics.clWebOlive 2951
Graphics.clWebOlive constant 2951
Graphics.clWebOliveDrab 2951
Graphics.clWebOliveDrab constant 2951
Graphics.clWebOrange 2951
Graphics.clWebOrange constant 2951
Graphics.clWebOrangeRed 2952
Graphics.clWebOrangeRed constant 2952
Graphics.clWebOrchid 2952
Graphics.clWebOrchid constant 2952
Graphics.clWebPaleGoldenrod 2952
Graphics.clWebPaleGoldenrod constant 2952
Graphics.clWebPaleGreen 2953
Graphics.clWebPaleGreen constant 2953
Graphics.clWebPaleTurquoise 2953
Graphics.clWebPaleTurquoise constant 2953
Graphics.clWebPaleVioletRed 2953
Graphics.clWebPaleVioletRed constant 2953

Graphics.clWebPapayaWhip 2954
Graphics.clWebPapayaWhip constant 2954
Graphics.clWebPeachPuff 2954
Graphics.clWebPeachPuff constant 2954
Graphics.clWebPeru 2954
Graphics.clWebPeru constant 2954
Graphics.clWebPink 2955
Graphics.clWebPink constant 2955
Graphics.clWebPlum 2955
Graphics.clWebPlum constant 2955
Graphics.clWebPowderBlue 2955
Graphics.clWebPowderBlue constant 2955
Graphics.clWebPurple 2956
Graphics.clWebPurple constant 2956
Graphics.clWebRed 2956
Graphics.clWebRed constant 2956
Graphics.clWebRosyBrown 2956
Graphics.clWebRosyBrown constant 2956
Graphics.clWebRoyalBlue 2957
Graphics.clWebRoyalBlue constant 2957
Graphics.clWebSaddleBrown 2957
Graphics.clWebSaddleBrown constant 2957
Graphics.clWebSalmon 2957
Graphics.clWebSalmon constant 2957
Graphics.clWebSandyBrown 2958
Graphics.clWebSandyBrown constant 2958
Graphics.clWebSeaGreen 2958
Graphics.clWebSeaGreen constant 2958
Graphics.clWebSeashell 2958
Graphics.clWebSeashell constant 2958
Graphics.clWebSienna 2959
Graphics.clWebSienna constant 2959
Graphics.clWebSilver 2959
Graphics.clWebSilver constant 2959
Graphics.clWebSkyBlue 2959
Graphics.clWebSkyBlue constant 2959
Graphics.clWebSlateBlue 2960
Graphics.clWebSlateBlue constant 2960
Graphics.clWebSlateGray 2960
Graphics.clWebSlateGray constant 2960
Graphics.clWebSnow 2960
Graphics.clWebSnow constant 2960
Graphics.clWebSpringGreen 2961
Graphics.clWebSpringGreen constant 2961
Graphics.clWebSteelBlue 2961
Graphics.clWebSteelBlue constant 2961
Graphics.clWebTan 2961
Graphics.clWebTan constant 2961
Graphics.clWebTeal 2962
Graphics.clWebTeal constant 2962
Graphics.clWebThistle 2962
Graphics.clWebThistle constant 2962
Graphics.clWebTomato 2962
Graphics.clWebTomato constant 2962
Graphics.clWebTurquoise 2963
Graphics.clWebTurquoise constant 2963
Graphics.clWebViolet 2963
Graphics.clWebViolet constant 2963
Graphics.clWebWheat 2963
Graphics.clWebWheat constant 2963
Graphics.clWebWhite 2964
Graphics.clWebWhite constant 2964
Graphics.clWebWhiteSmoke 2964
Graphics.clWebWhiteSmoke constant 2964
Graphics.clWebYellow 2964
Graphics.clWebYellow constant 2964
Graphics.clWebYellowGreen 2965
Graphics.clWebYellowGreen constant 2965
Graphics.clWhite 2965
Graphics.clWhite constant 2965
Graphics.clWindow 2965
Graphics.clWindow constant 2965
Graphics.clWindowFrame 2966
Graphics.clWindowFrame constant 2966
Graphics.clWindowText 2966
Graphics.clWindowText constant 2966
Graphics.clYellow 2967
Graphics.clYellow constant 2967

Graphics.cmBlackness 2967
Graphics.cmBlackness constant 2967
Graphics.cmDstInvert 2968
Graphics.cmDstInvert constant 2968
Graphics.cmMergeCopy 2968
Graphics.cmMergeCopy constant 2968
Graphics.cmMergePaint 2969
Graphics.cmMergePaint constant 2969
Graphics.cmNotSrcCopy 2969
Graphics.cmNotSrcCopy constant 2969
Graphics.cmNotSrcErase 2970
Graphics.cmNotSrcErase constant 2970
Graphics.cmPatCopy 2970
Graphics.cmPatCopy constant 2970
Graphics.cmPatInvert 2971
Graphics.cmPatInvert constant 2971
Graphics.cmPatPaint 2971
Graphics.cmPatPaint constant 2971
Graphics.cmSrcAnd 2972
Graphics.cmSrcAnd constant 2972
Graphics.cmSrcCopy 2972
Graphics.cmSrcCopy constant 2972
Graphics.cmSrcErase 2973
Graphics.cmSrcErase constant 2973
Graphics.cmSrcInvert 2973
Graphics.cmSrcInvert constant 2973
Graphics.cmSrcPaint 2974
Graphics.cmSrcPaint constant 2974
Graphics.cmWhiteness 2974
Graphics.cmWhiteness constant 2974
Graphics.ColorToIdent 2838
Graphics.ColorToIdent function 2838
Graphics.ColorToRGB 2838
Graphics.ColorToRGB function 2838
Graphics.ColorToString 2840
Graphics.ColorToString function 2840
Graphics.CopyPalette 2841
Graphics.CopyPalette function 2841
Graphics.CreateGrayMappedBmp 2841
Graphics.CreateGrayMappedBmp function 2841
Graphics.CreateGrayMappedRes 2842
Graphics.CreateGrayMappedRes function 2842
Graphics.CreateMappedBmp 2842
Graphics.CreateMappedBmp function 2842
Graphics.CreateMappedRes 2843
Graphics.CreateMappedRes function 2843
Graphics.DDBsOnly 2890
Graphics.DDBsOnly variable 2890
Graphics.DefFontData 2891
Graphics.DefFontData variable 2891
Graphics.EInvalidGraphic 2627
Graphics.EInvalidGraphicOperation 2628
Graphics.ExtendedColorsCount 2975
Graphics.ExtendedColorsCount constant 2975
Graphics.FreeMemoryContexts 2844
Graphics.FreeMemoryContexts function 2844
Graphics.GetCharsetValues 2844
Graphics.GetCharsetValues function 2844
Graphics.GetColorValues 2848
Graphics.GetColorValues function 2848
Graphics.GetDefFontCharSet 2848
Graphics.GetDefFontCharSet function 2848
Graphics.GetDIB 2849
Graphics.GetDIB function 2849
Graphics.GetDIBSizes 2849
Graphics.GetDIBSizes function 2849
Graphics.GraphicExtension 2850
Graphics.GraphicExtension function 2850
Graphics.GraphicFileMask 2852
Graphics.GraphicFileMask function 2852
Graphics.GraphicFilter 2854
Graphics.GraphicFilter function 2854
Graphics.HENHMETAFILE 2875
Graphics.HENHMETAFILE type 2875
Graphics.HMETAFILE 2876
Graphics.HMETAFILE type 2876
Graphics.IChangeNotifier 2834
Graphics.IChangeNotifier.Changed 2835

Graphics.IdentToCharset 2857
Graphics.IdentToCharset function 2857
Graphics.IdentToColor 2857
Graphics.IdentToColor function 2857
Graphics.PaletteChanged 2858
Graphics.PaletteChanged function 2858
Graphics.PColor 2876
Graphics.PColor type 2876
Graphics.PCursorOrIcon 2876
Graphics.PCursorOrIcon type 2876
Graphics.PIconRec 2877
Graphics.PIconRec type 2877
Graphics.PResource 2877
Graphics.PResource type 2877
Graphics.rc3_Cursor 2976
Graphics.rc3_Cursor constant 2976
Graphics.rc3_Icon 2976
Graphics.rc3_Icon constant 2976
Graphics.rc3_StockIcon 2976
Graphics.rc3_StockIcon constant 2976
Graphics.StandardColorsCount 2977
Graphics.StandardColorsCount constant 2977
Graphics.StringToColor 2858
Graphics.StringToColor function 2858
Graphics.SystemPalette16 2891
Graphics.SystemPalette16 variable 2891
Graphics.TAlphaFormat 2860
Graphics.TAlphaFormat enumeration 2860
Graphics.TBitmap 2628
Graphics.TBitmap.AlphaFormat 2651
Graphics.TBitmap.Assign 2633
Graphics.TBitmap.Canvas 2651
Graphics.TBitmap.Changed 2635
Graphics.TBitmap.Create 2631
Graphics.TBitmap.Destroy 2632
Graphics.TBitmap.Dormant 2635
Graphics.TBitmap.Draw 2636
Graphics.TBitmap.DrawTransparent 2637
Graphics.TBitmap.FreeImage 2637
Graphics.TBitmap.GetEmpty 2638
Graphics.TBitmap.GetHeight 2638
Graphics.TBitmap.GetPalette 2639
Graphics.TBitmap.GetSupportsPartialTransparency 2639
Graphics.TBitmap.GetWidth 2639
Graphics.TBitmap.Handle 2652
Graphics.TBitmap.HandleAllocated 2639
Graphics.TBitmap.HandleNeeded 2640
Graphics.TBitmap.HandleType 2653
Graphics.TBitmap.IgnorePalette 2654
Graphics.TBitmap.LoadFromClipboardFormat 2640
Graphics.TBitmap.LoadFromResourceID 2642
Graphics.TBitmap.LoadFromResourceName 2642
Graphics.TBitmap.LoadFromStream 2643
Graphics.TBitmap.Mask 2644
Graphics.TBitmap.MaskHandle 2654
Graphics.TBitmap.MaskHandleNeeded 2644
Graphics.TBitmap.Monochrome 2654
Graphics.TBitmap.PaletteNeeded 2645
Graphics.TBitmap.PixelFormat 2656
Graphics.TBitmap.ReadData 2645
Graphics.TBitmap.ReleaseHandle 2645
Graphics.TBitmap.ReleaseMaskHandle 2646
Graphics.TBitmap.ReleasePalette 2647
Graphics.TBitmap.SaveToClipboardFormat 2647
Graphics.TBitmap.SaveToStream 2649
Graphics.TBitmap.ScanLine 2658
Graphics.TBitmap.SetHeight 2649
Graphics.TBitmap.SetPalette 2649
Graphics.TBitmap.SetSize 2650
Graphics.TBitmap.SetWidth 2650
Graphics.TBitmap.TransparentColor 2660
Graphics.TBitmap.TransparentMode 2661
Graphics.TBitmap.WriteData 2650
Graphics.TBitmapHandleType 2861
Graphics.TBitmapHandleType enumeration 2861
Graphics.TBitmapImage 2662
Graphics.TBitmapImage.Destroy 2663
Graphics.TBitmapImage.FreeHandle 2663

Graphics.TBrush 2664
Graphics.TBrush.Assign 2666
Graphics.TBrush.Bitmap 2668
Graphics.TBrush.Color 2669
Graphics.TBrush.Create 2665
Graphics.TBrush.Destroy 2665
Graphics.TBrush.GetBitmap 2666
Graphics.TBrush.GetColor 2666
Graphics.TBrush.GetHandle 2667
Graphics.TBrush.GetStyle 2667
Graphics.TBrush.Handle 2670
Graphics.TBrush.SetBitmap 2667
Graphics.TBrush.SetColor 2667
Graphics.TBrush.SetHandle 2668
Graphics.TBrush.SetStyle 2668
Graphics.TBrush.Style 2670
Graphics.TBrushData 2861
Graphics.TBrushData record 2861
Graphics.TBrushRecall 2671
Graphics.TBrushRecall.Create 2672
Graphics.TBrushStyle 2862
Graphics.TBrushStyle enumeration 2862
Graphics.TCanvas 2673
Graphics.TCanvas.Arc 2679
Graphics.TCanvas.Brush 2716
Graphics.TCanvas.BrushCopy 2680
Graphics.TCanvas.CanvasOrientation 2718
Graphics.TCanvas.Changed 2681
Graphics.TCanvas.Changing 2682
Graphics.TCanvas.Chord 2682
Graphics.TCanvas.ClipRect 2718
Graphics.TCanvas.CopyMode 2719
Graphics.TCanvas.CopyRect 2683
Graphics.TCanvas.Create 2675
Graphics.TCanvas.CreateHandle 2685
Graphics.TCanvas.Destroy 2678
Graphics.TCanvas.Draw 2685
Graphics.TCanvas.DrawFocusRect 2687
Graphics.TCanvas.Ellipse 2688
Graphics.TCanvas.FillRect 2689
Graphics.TCanvas.FloodFill 2690
Graphics.TCanvas.Font 2721
Graphics.TCanvas.FrameRect 2692
Graphics.TCanvas.Handle 2722
Graphics.TCanvas.HandleAllocated 2693
Graphics.TCanvas.LineTo 2693
Graphics.TCanvas.Lock 2694
Graphics.TCanvas.LockCount 2725
Graphics.TCanvas.MoveTo 2695
Graphics.TCanvas.OnChange 2729
Graphics.TCanvas.OnChanging 2730
Graphics.TCanvas.Pen 2725
Graphics.TCanvas.PenPos 2727
Graphics.TCanvas.Pie 2695
Graphics.TCanvas.Pixels 2727
Graphics.TCanvas.PolyBezier 2696
Graphics.TCanvas.PolyBezierTo 2697
Graphics.TCanvas.Polygon 2697
Graphics.TCanvas.Polyline 2698
Graphics.TCanvas.Rectangle 2700
Graphics.TCanvas.Refresh 2701
Graphics.TCanvas.RequiredState 2702
Graphics.TCanvas.RoundRect 2702
Graphics.TCanvas.StretchDraw 2704
Graphics.TCanvas.TextExtent 2705
Graphics.TCanvas.TextFlags 2728
Graphics.TCanvas.TextHeight 2706
Graphics.TCanvas.TextOut 2707
Graphics.TCanvas.TextRect 2713
Graphics.TCanvas.TextWidth 2714
Graphics.TCanvas.TryLock 2715
Graphics.TCanvas.Unlock 2716
Graphics.TCanvasOrientation 2863
Graphics.TCanvasOrientation enumeration 2863
Graphics.TCanvasState 2877
Graphics.TCanvasState type 2877
Graphics.TCanvasStates 2863
Graphics.TCanvasStates enumeration 2863

Graphics.TColor 2878
Graphics.TColor type 2878
Graphics.TCopyMode 2880
Graphics.TCopyMode type 2880
Graphics.TCursorOrIcon 2864
Graphics.TCursorOrIcon record 2864
Graphics.TFillMode 2864
Graphics.TFillMode enumeration 2864
Graphics.TFillStyle 2864
Graphics.TFillStyle enumeration 2864
Graphics.TFont 2730
Graphics.TFont.Assign 2733
Graphics.TFont.Changed 2734
Graphics.TFont.Charset 2739
Graphics.TFont.Color 2739
Graphics.TFont.Create 2732
Graphics.TFont.Destroy 2732
Graphics.TFont.FontAdapter 2741
Graphics.TFont.GetCharset 2735
Graphics.TFont.GetHandle 2735
Graphics.TFont.GetHeight 2735
Graphics.TFont.GetName 2735
Graphics.TFont.GetOrientation 2736
Graphics.TFont.GetPitch 2736
Graphics.TFont.GetSize 2736
Graphics.TFont.GetStyle 2736
Graphics.TFont.Handle 2741
Graphics.TFont.Height 2742
Graphics.TFont.Name 2743
Graphics.TFont.Orientation 2744
Graphics.TFont.Pitch 2744
Graphics.TFont.PixelsPerInch 2745
Graphics.TFont.SetCharset 2737
Graphics.TFont.SetColor 2737
Graphics.TFont.SetHandle 2737
Graphics.TFont.SetHeight 2737
Graphics.TFont.SetName 2737
Graphics.TFont.SetOrientation 2738
Graphics.TFont.SetPitch 2738
Graphics.TFont.SetSize 2738
Graphics.TFont.SetStyle 2738
Graphics.TFont.Size 2745
Graphics.TFont.Style 2746
Graphics.TFontCharset 2881
Graphics.TFontCharset type 2881
Graphics.TFontData 2865
Graphics.TFontData record 2865
Graphics.TFontDataName 2882
Graphics.TFontDataName type 2882
Graphics.TFontName 2882
Graphics.TFontName type 2882
Graphics.TFontPitch 2866
Graphics.TFontPitch enumeration 2866
Graphics.TFontRecall 2747
Graphics.TFontRecall.Create 2747
Graphics.TFontStyle 2866
Graphics.TFontStyle enumeration 2866
Graphics.TFontStyles 2883
Graphics.TFontStyles type 2883
Graphics.TFontStylesBase 2887
Graphics.TFontStylesBase type 2887
Graphics.TGraphic 2748
Graphics.TGraphic.Changed 2751
Graphics.TGraphic.Create 2750
Graphics.TGraphic.DefineProperties 2751
Graphics.TGraphic.Draw 2752
Graphics.TGraphic.DrawTransparent 2752
Graphics.TGraphic.Empty 2760
Graphics.TGraphic.Equals 2752, 2753
Graphics.TGraphic.GetEmpty 2753
Graphics.TGraphic.GetHeight 2753
Graphics.TGraphic.GetPalette 2754
Graphics.TGraphic.GetSupportsPartialTransparency 2754
Graphics.TGraphic.GetTransparent 2754
Graphics.TGraphic.GetWidth 2754
Graphics.TGraphic.Height 2760
Graphics.TGraphic.LoadFromClipboardFormat 2754
Graphics.TGraphic.LoadFromFile 2755

Graphics.TGraphic.LoadFromStream 2756
Graphics.TGraphic.Modified 2761
Graphics.TGraphic.OnChange 2763
Graphics.TGraphic.OnProgress 2764
Graphics.TGraphic.Palette 2761
Graphics.TGraphic.PaletteModified 2761
Graphics.TGraphic.Progress 2756
Graphics.TGraphic.ReadData 2757
Graphics.TGraphic.SaveToClipboardFormat 2757
Graphics.TGraphic.SaveToFile 2757
Graphics.TGraphic.SaveToStream 2758
Graphics.TGraphic.SetHeight 2758
Graphics.TGraphic.SetPalette 2759
Graphics.TGraphic.SetSize 2759
Graphics.TGraphic.SetTransparent 2759
Graphics.TGraphic.SetWidth 2759
Graphics.TGraphic.SupportsPartialTransparency 2762
Graphics.TGraphic.Transparent 2762
Graphics.TGraphic.Width 2763
Graphics.TGraphic.WriteData 2760
Graphics.TGraphicClass 2887
Graphics.TGraphicClass type 2887
Graphics.TGraphicsObject 2764
Graphics.TGraphicsObject.Changed 2765
Graphics.TGraphicsObject.HandleAllocated 2766
Graphics.TGraphicsObject.Lock 2766
Graphics.TGraphicsObject.OnChange 2768
Graphics.TGraphicsObject.OwnerCriticalSection 2767
Graphics.TGraphicsObject.Unlock 2767
Graphics.TIcon 2768
Graphics.TIcon.Assign 2772
Graphics.TIcon.Create 2770
Graphics.TIcon.Destroy 2772
Graphics.TIcon.Draw 2773
Graphics.TIcon.GetEmpty 2773
Graphics.TIcon.GetHeight 2773
Graphics.TIcon.GetWidth 2773
Graphics.TIcon.Handle 2777
Graphics.TIcon.HandleAllocated 2774
Graphics.TIcon.LoadFromClipboardFormat 2774
Graphics.TIcon.LoadFromResourceID 2774
Graphics.TIcon.LoadFromResourceName 2775
Graphics.TIcon.LoadFromStream 2775
Graphics.TIcon.ReleaseHandle 2775
Graphics.TIcon.SaveToClipboardFormat 2776
Graphics.TIcon.SaveToStream 2776
Graphics.TIcon.SetHeight 2776
Graphics.TIcon.SetSize 2777
Graphics.TIcon.SetTransparent 2777
Graphics.TIcon.SetWidth 2777
Graphics.TIconImage 2778
Graphics.TIconImage.Destroy 2779
Graphics.TIconImage.FreeHandle 2779
Graphics.TIconRec 2867
Graphics.TIconRec record 2867
Graphics.TMetafile 2780
Graphics.TMetafile.Assign 2785
Graphics.TMetafile.Clear 2785
Graphics.TMetafile.Create 2782
Graphics.TMetafile.CreatedBy 2792
Graphics.TMetafile.Description 2792
Graphics.TMetafile.Destroy 2783
Graphics.TMetafile.Draw 2785
Graphics.TMetafile.Enhanced 2793
Graphics.TMetafile.GetEmpty 2786
Graphics.TMetafile.GetHeight 2786
Graphics.TMetafile.GetPalette 2786
Graphics.TMetafile.GetWidth 2786
Graphics.TMetafile.Handle 2793
Graphics.TMetafile.HandleAllocated 2786
Graphics.TMetafile.Inch 2793
Graphics.TMetafile.LoadFromClipboardFormat 2787
Graphics.TMetafile.LoadFromStream 2787
Graphics.TMetafile.MMHeight 2793
Graphics.TMetafile.MMWidth 2794
Graphics.TMetafile.ReadData 2787
Graphics.TMetafile.ReadEMFStream 2788
Graphics.TMetafile.ReadWMFStream 2788

Graphics.TMetafile.ReleaseHandle 2788
Graphics.TMetafile.SaveToClipboardFormat 2788
Graphics.TMetafile.SaveToFile 2789
Graphics.TMetafile.SaveToStream 2789
Graphics.TMetafile.SetHeight 2790
Graphics.TMetafile.SetSize 2790
Graphics.TMetafile.SetTransparent 2790
Graphics.TMetafile.SetWidth 2790
Graphics.TMetafile.TestEMF 2791
Graphics.TMetafile.WriteData 2791
Graphics.TMetafile.WriteEMFStream 2791
Graphics.TMetafile.WriteWMFStream 2791
Graphics.TMetafileCanvas 2794
Graphics.TMetafileCanvas.Create 2796
Graphics.TMetafileCanvas.CreateWithComment 2798
Graphics.TMetafileCanvas.Destroy 2799
Graphics.TMetafileImage 2799
Graphics.TMetafileImage.Destroy 2800
Graphics.TMetafileImage.FreeHandle 2800
Graphics.TPen 2801
Graphics.TPen.Assign 2803
Graphics.TPen.Color 2806
Graphics.TPen.Create 2802
Graphics.TPen.Destroy 2802
Graphics.TPen.GetColor 2804
Graphics.TPen.GetHandle 2804
Graphics.TPen.GetStyle 2804
Graphics.TPen.GetWidth 2804
Graphics.TPen.Handle 2807
Graphics.TPen.Mode 2808
Graphics.TPen.SetColor 2805
Graphics.TPen.SetHandle 2805
Graphics.TPen.SetMode 2805
Graphics.TPen.SetStyle 2805
Graphics.TPen.SetWidth 2805
Graphics.TPen.Style 2808
Graphics.TPen.Width 2809
Graphics.TPenData 2868
Graphics.TPenData record 2868
Graphics.TPenMode 2868
Graphics.TPenMode enumeration 2868
Graphics.TPenRecall 2811
Graphics.TPenRecall.Create 2811
Graphics.TPenStyle 2870
Graphics.TPenStyle enumeration 2870
Graphics.TPicture 2812
Graphics.TPicture.Assign 2815
Graphics.TPicture.AssignTo 2816
Graphics.TPicture.Bitmap 2825
Graphics.TPicture.Changed 2817
Graphics.TPicture.Create 2814
Graphics.TPicture.DefineProperties 2817
Graphics.TPicture.Destroy 2814
Graphics.TPicture.Graphic 2827
Graphics.TPicture.Height 2828
Graphics.TPicture.Icon 2828
Graphics.TPicture.LoadFromClipboardFormat 2818
Graphics.TPicture.LoadFromFile 2818
Graphics.TPicture.LoadFromStream 2819
Graphics.TPicture.Metafile 2828
Graphics.TPicture.OnChange 2830
Graphics.TPicture.OnProgress 2830
Graphics.TPicture.PictureAdapter 2829
Graphics.TPicture.Progress 2819
Graphics.TPicture.RegisterClipboardFormat 2819
Graphics.TPicture.RegisterFileFormat 2821
Graphics.TPicture.RegisterFileFormatRes 2822
Graphics.TPicture.SaveToClipboardFormat 2822
Graphics.TPicture.SaveToFile 2823
Graphics.TPicture.SaveToStream 2824
Graphics.TPicture.SupportsClipboardFormat 2824
Graphics.TPicture.UnregisterGraphicClass 2825
Graphics.TPicture.Width 2829
Graphics.TPixelFormat 2871
Graphics.TPixelFormat enumeration 2871
Graphics.TProgressEvent 2888
Graphics.TProgressEvent type 2888
Graphics.TProgressStage 2872

Graphics.TProgressStage enumeration 2872
Graphics.TransparentStretchBlt 2859
Graphics.TransparentStretchBlt function 2859
Graphics.TResData 2872
Graphics.TResData record 2872
Graphics.TResource 2873
Graphics.TResource record 2873
Graphics.TSharedImage 2830
Graphics.TSharedImage.Destroy 2831
Graphics.TSharedImage.FreeHandle 2832
Graphics.TSharedImage.RefCount 2832
Graphics.TSharedImage.Reference 2832
Graphics.TSharedImage.Release 2832
Graphics.TTextFormat 2889
Graphics.TTextFormat type 2889
Graphics.TTextFormats 2873
Graphics.TTextFormats enumeration 2873
Graphics.TTransparentMode 2874
Graphics.TTransparentMode enumeration 2874
Graphics.WebColorsCount 2978
Graphics.WebColorsCount constant 2978
GraphUtil 2979
GraphUtil namespace 2979
Constants 2993
Functions 2981
Structs, Records, Enums 2989
Types 2992
GraphUtil.ColorAdjustLuma 2981
GraphUtil.ColorAdjustLuma function 2981
GraphUtil.ColorHLSToRGB 2982
GraphUtil.ColorHLSToRGB function 2982
GraphUtil.ColorRGBToHLS 2982
GraphUtil.ColorRGBToHLS function 2982
GraphUtil.ColorToWebColorName 2982
GraphUtil.ColorToWebColorName function 2982
GraphUtil.ColorToWebColorStr 2983
GraphUtil.ColorToWebColorStr function 2983
GraphUtil.DrawArrow 2983
GraphUtil.DrawArrow function 2983
GraphUtil.DrawCheck 2983
GraphUtil.DrawCheck function 2983
GraphUtil.DrawChevron 2984
GraphUtil.DrawChevron function 2984
GraphUtil.DrawTransparentBitmap 2984
GraphUtil.DrawTransparentBitmap function 2984
GraphUtil.GetHighLightColor 2985
GraphUtil.GetHighLightColor function 2985
GraphUtil.GetShadowColor 2985
GraphUtil.GetShadowColor function 2985
GraphUtil.GradientFillCanvas 2985
GraphUtil.GradientFillCanvas function 2985
GraphUtil.LoadCompressedResourceBitmap 2986
GraphUtil.LoadCompressedResourceBitmap function 2986
GraphUtil.RGBToWebColorName 2986
GraphUtil.RGBToWebColorName function 2986
GraphUtil.RGBToWebColorStr 2986
GraphUtil.RGBToWebColorStr function 2986
GraphUtil.ScaleImage 2987
GraphUtil.ScaleImage function 2987
GraphUtil.SortColorArray 2987
GraphUtil.SortColorArray function 2987
GraphUtil.SplitTransparentBitmap 2987
GraphUtil.SplitTransparentBitmap function 2987
GraphUtil.TArrowType 2989
GraphUtil.TArrowType enumeration 2989
GraphUtil.TColorArray 2992
GraphUtil.TColorArray type 2992
GraphUtil.TColorArraySortType 2989
GraphUtil.TColorArraySortType enumeration 2989
GraphUtil.TGradientDirection 2990
GraphUtil.TGradientDirection enumeration 2990
GraphUtil.TScrollDirection 2990
GraphUtil.TScrollDirection enumeration 2990
GraphUtil.WebColorNameToColor 2988
GraphUtil.WebColorNameToColor function 2988
GraphUtil.WebColorStrToColor 2988
GraphUtil.WebColorStrToColor function 2988
GraphUtil.WebColorToRGB 2988

GraphUtil.WebColorToRGB function 2988
GraphUtil.WebNamedColors 2993
GraphUtil.WebNamedColors constant 2993
GraphUtil.WebNamedColorsCount 2995
GraphUtil.WebNamedColorsCount constant 2995
Grids 2997
Grids namespace 2997
Classes 2999
Constants 3212
Structs, Records, Enums 3199
Types 3205
Grids.EInvalidGridOperation 2999
Grids.MaxCustomExtents 3212
Grids.MaxCustomExtents constant 3212
Grids.MaxShortInt 3212
Grids.MaxShortInt constant 3212
Grids.TCustomDrawGrid 3000
Grids.TCustomDrawGrid.Canvas 3007
Grids.TCustomDrawGrid.CellRect 3003
Grids.TCustomDrawGrid.Col 3009
Grids.TCustomDrawGrid.ColumnMoved 3004
Grids.TCustomDrawGrid.ColWidths 3011
Grids.TCustomDrawGrid.DrawCell 3004
Grids.TCustomDrawGrid.EditorMode 3011
Grids.TCustomDrawGrid.GetEditMask 3004
Grids.TCustomDrawGrid.GetEditText 3005
Grids.TCustomDrawGrid.GridHeight 3012
Grids.TCustomDrawGrid.GridWidth 3012
Grids.TCustomDrawGrid.LeftCol 3012
Grids.TCustomDrawGrid.MouseToCell 3005
Grids.TCustomDrawGrid.OnColumnMoved 3019
Grids.TCustomDrawGrid.OnDrawCell 3020
Grids.TCustomDrawGrid.OnGetEditMask 3021
Grids.TCustomDrawGrid.OnGetEditText 3022
Grids.TCustomDrawGrid.OnRowMoved 3023
Grids.TCustomDrawGrid.OnSelectCell 3025
Grids.TCustomDrawGrid.OnSetEditText 3026
Grids.TCustomDrawGrid.OnTopLeftChanged 3027
Grids.TCustomDrawGrid.Row 3013
Grids.TCustomDrawGrid.RowHeights 3014
Grids.TCustomDrawGrid.RowMoved 3005
Grids.TCustomDrawGrid.SelectCell 3006
Grids.TCustomDrawGrid.Selection 3015
Grids.TCustomDrawGrid.SetEditText 3006
Grids.TCustomDrawGrid.TabStops 3017
Grids.TCustomDrawGrid.TopLeftChanged 3007
Grids.TCustomDrawGrid.TopRow 3017
Grids.TCustomGrid 3027
Grids.TCustomGrid.AdjustSize 3034
Grids.TCustomGrid.BeginColumnDrag 3034
Grids.TCustomGrid.BeginRowDrag 3034
Grids.TCustomGrid.BorderStyle 3060
Grids.TCustomGrid.BoxRect 3035
Grids.TCustomGrid.CalcDrawInfo 3035
Grids.TCustomGrid.CalcFixedInfo 3036
Grids.TCustomGrid.CalcSizingState 3036
Grids.TCustomGrid.CanEditAcceptKey 3036
Grids.TCustomGrid.CanEditModify 3037
Grids.TCustomGrid.CanEditShow 3037
Grids.TCustomGrid.CanGridAcceptKey 3037
Grids.TCustomGrid.CellRect 3038
Grids.TCustomGrid.ChangeGridOrientation 3038
Grids.TCustomGrid.CheckColumnDrag 3039
Grids.TCustomGrid.CheckRowDrag 3039
Grids.TCustomGrid.Col 3060
Grids.TCustomGrid.ColCount 3062
Grids.TCustomGrid.Color 3063
Grids.TCustomGrid.ColumnMoved 3039
Grids.TCustomGrid.ColWidths 3063
Grids.TCustomGrid.ColWidthsChanged 3040
Grids.TCustomGrid.Create 3030
Grids.TCustomGrid.CreateEditor 3040
Grids.TCustomGrid.CreateParams 3041
Grids.TCustomGrid.DefaultColWidth 3064
Grids.TCustomGrid.DefaultDrawing 3066
Grids.TCustomGrid.DefaultRowHeight 3066
Grids.TCustomGrid.DefineProperties 3041
Grids.TCustomGrid.DeleteColumn 3041

Grids.TCustomGrid.DeleteRow 3042
Grids.TCustomGrid.DesignOptionsBoost 3031
Grids.TCustomGrid.Destroy 3031
Grids.TCustomGrid.DoExit 3042
Grids.TCustomGrid.DoMouseWheelDown 3042
Grids.TCustomGrid.DoMouseWheelUp 3043
Grids.TCustomGrid.DrawCell 3043
Grids.TCustomGrid.EditorMode 3066
Grids.TCustomGrid.EndColumnDrag 3044
Grids.TCustomGrid.EndRowDrag 3044
Grids.TCustomGrid.FGridState 3031
Grids.TCustomGrid.FixedColor 3067
Grids.TCustomGrid.FixedCols 3067
Grids.TCustomGrid.FixedRows 3068
Grids.TCustomGrid.FocusCell 3044
Grids.TCustomGrid.FSaveCellExtents 3032
Grids.TCustomGrid.GetEditLimit 3045
Grids.TCustomGrid.GetEditMask 3045
Grids.TCustomGrid.GetEditStyle 3046
Grids.TCustomGrid.GetEditText 3046
Grids.TCustomGrid.GetGridHeight 3046
Grids.TCustomGrid.GetGridWidth 3047
Grids.TCustomGrid.GridHeight 3068
Grids.TCustomGrid.GridLineWidth 3069
Grids.TCustomGrid.GridWidth 3070
Grids.TCustomGrid.HideEdit 3047
Grids.TCustomGrid.HideEditor 3048
Grids.TCustomGrid.HitTest 3071
Grids.TCustomGrid.InplaceEditor 3071
Grids.TCustomGrid.InvalidateCell 3048
Grids.TCustomGrid.InvalidateCol 3048
Grids.TCustomGrid.InvalidateEditor 3049
Grids.TCustomGrid.InvalidateGrid 3049
Grids.TCustomGrid.InvalidateRow 3050
Grids.TCustomGrid.KeyDown 3050
Grids.TCustomGrid.KeyPress 3050
Grids.TCustomGrid.LeftCol 3072
Grids.TCustomGrid.MouseCoord 3051
Grids.TCustomGrid.MouseDown 3051
Grids.TCustomGrid.MouseMove 3052
Grids.TCustomGrid.MouseUp 3052
Grids.TCustomGrid.MoveColRow 3053
Grids.TCustomGrid.MoveColumn 3053
Grids.TCustomGrid.MoveRow 3053
Grids.TCustomGrid.Options 3072
Grids.TCustomGrid.Paint 3054
Grids.TCustomGrid.ParentColor 3073
Grids.TCustomGrid.Row 3074
Grids.TCustomGrid.RowCount 3075
Grids.TCustomGrid.RowHeights 3076
Grids.TCustomGrid.RowHeightsChanged 3054
Grids.TCustomGrid.RowMoved 3055
Grids.TCustomGrid.ScrollBars 3077
Grids.TCustomGrid.ScrollData 3055
Grids.TCustomGrid.SelectCell 3055
Grids.TCustomGrid.Selection 3077
Grids.TCustomGrid.SetEditText 3056
Grids.TCustomGrid.ShowEditor 3056
Grids.TCustomGrid.ShowEditorChar 3057
Grids.TCustomGrid.SizeChanged 3057
Grids.TCustomGrid.Sizing 3057
Grids.TCustomGrid.TabStop 3079
Grids.TCustomGrid.TabStops 3079
Grids.TCustomGrid.TimedScroll 3058
Grids.TCustomGrid.TopLeftChanged 3058
Grids.TCustomGrid.TopRow 3080
Grids.TCustomGrid.UpdateDesigner 3059
Grids.TCustomGrid.VirtualView 3032
Grids.TCustomGrid.VisibleColCount 3082
Grids.TCustomGrid.VisibleRowCount 3082
Grids.TDrawCellEvent 3205
Grids.TDrawCellEvent type 3205
Grids.TDrawGrid 3083
Grids.TDrawGrid.Align 3088
Grids.TDrawGrid.Anchors 3092
Grids.TDrawGrid.BevelEdges 3092
Grids.TDrawGrid.BevelInner 3093
Grids.TDrawGrid.BevelKind 3093

Grids.TDrawGrid.BevelOuter 3094
Grids.TDrawGrid.BevelWidth 3094
Grids.TDrawGrid.BiDiMode 3095
Grids.TDrawGrid.BorderStyle 3095
Grids.TDrawGrid.ColCount 3095
Grids.TDrawGrid.Color 3096
Grids.TDrawGrid.Constraints 3097
Grids.TDrawGrid.Ctl3D 3098
Grids.TDrawGrid.DefaultColWidth 3098
Grids.TDrawGrid.DefaultDrawing 3100
Grids.TDrawGrid.DefaultRowHeight 3100
Grids.TDrawGrid.DoubleBuffered 3101
Grids.TDrawGrid.DragCursor 3102
Grids.TDrawGrid.DragKind 3103
Grids.TDrawGrid.DragMode 3103
Grids.TDrawGrid.Enabled 3105
Grids.TDrawGrid.FixedColor 3105
Grids.TDrawGrid.FixedCols 3106
Grids.TDrawGrid.FixedRows 3106
Grids.TDrawGrid.Font 3107
Grids.TDrawGrid.GridLineWidth 3108
Grids.TDrawGrid.OnClick 3125
Grids.TDrawGrid.OnColumnMoved 3127
Grids.TDrawGrid.OnContextPopup 3127
Grids.TDrawGrid.OnDblClick 3128
Grids.TDrawGrid.OnDragDrop 3128
Grids.TDrawGrid.OnDragOver 3131
Grids.TDrawGrid.OnDrawCell 3134
Grids.TDrawGrid.OnEndDock 3135
Grids.TDrawGrid.OnEndDrag 3135
Grids.TDrawGrid.OnEnter 3136
Grids.TDrawGrid.OnExit 3139
Grids.TDrawGrid.OnGetEditMask 3142
Grids.TDrawGrid.OnGetEditText 3142
Grids.TDrawGrid.OnKeyDown 3142
Grids.TDrawGrid.OnKeyPress 3146
Grids.TDrawGrid.OnKeyUp 3147
Grids.TDrawGrid.OnMouseActivate 3149
Grids.TDrawGrid.OnMouseDown 3150
Grids.TDrawGrid.OnMouseEnter 3152
Grids.TDrawGrid.OnMouseLeave 3152
Grids.TDrawGrid.OnMouseMove 3153
Grids.TDrawGrid.OnMouseUp 3155
Grids.TDrawGrid.OnMouseWheelDown 3157
Grids.TDrawGrid.OnMouseWheelUp 3157
Grids.TDrawGrid.OnRowMoved 3158
Grids.TDrawGrid.OnSelectCell 3158
Grids.TDrawGrid.OnSetEditText 3159
Grids.TDrawGrid.OnStartDock 3159
Grids.TDrawGrid.OnStartDrag 3160
Grids.TDrawGrid.OnTopLeftChanged 3160
Grids.TDrawGrid.Options 3109
Grids.TDrawGrid.ParentBiDiMode 3111
Grids.TDrawGrid.ParentColor 3111
Grids.TDrawGrid.ParentCtl3D 3112
Grids.TDrawGrid.ParentDoubleBuffered 3112
Grids.TDrawGrid.ParentFont 3112
Grids.TDrawGrid.ParentShowHint 3113
Grids.TDrawGrid.PopupMenu 3113
Grids.TDrawGrid.RowCount 3118
Grids.TDrawGrid.ScrollBars 3119
Grids.TDrawGrid.ShowHint 3120
Grids.TDrawGrid.TabOrder 3121
Grids.TDrawGrid.Visible 3122
Grids.TDrawGrid.VisibleColCount 3123
Grids.TDrawGrid.VisibleRowCount 3124
Grids.TEditStyle 3199
Grids.TEditStyle enumeration 3199
Grids.TGetEditEvent 3206
Grids.TGetEditEvent type 3206
Grids.TGetExtentsFunc 3206
Grids.TGetExtentsFunc type 3206
Grids.TGridAxisDrawInfo 3200
Grids.TGridAxisDrawInfo record 3200
Grids.TGridCoord 3200
Grids.TGridCoord record 3200
Grids.TGridDrawInfo 3201
Grids.TGridDrawInfo record 3201

Grids.TGridDrawState 3206
Grids.TGridDrawState type 3206
Grids.TGridMovement 3207
Grids.TGridMovement type 3207
Grids.TGridOption 3201
Grids.TGridOption enumeration 3201
Grids.TGridOptions 3207
Grids.TGridOptions type 3207
Grids.TGridRect 3203
Grids.TGridRect record 3203
Grids.TGridScrollDirection 3209
Grids.TGridScrollDirection type 3209
Grids.TGridState 3203
Grids.TGridState enumeration 3203
Grids.TInplaceEdit 3161
Grids.TInplaceEdit.BoundsChanged 3163
Grids.TInplaceEdit.Create 3162
Grids.TInplaceEdit.CreateParams 3164
Grids.TInplaceEdit.DblClick 3164
Grids.TInplaceEdit.Deselect 3165
Grids.TInplaceEdit.DoMouseWheel 3165
Grids.TInplaceEdit.EditCanModify 3165
Grids.TInplaceEdit.Grid 3171
Grids.TInplaceEdit.Hide 3166
Grids.TInplaceEdit.Invalidate 3166
Grids.TInplaceEdit.KeyDown 3167
Grids.TInplaceEdit.KeyPress 3167
Grids.TInplaceEdit.KeyUp 3168
Grids.TInplaceEdit.Move 3168
Grids.TInplaceEdit.PosEqual 3168
Grids.TInplaceEdit.SetFocus 3169
Grids.TInplaceEdit.UpdateContents 3169
Grids.TInplaceEdit.UpdateLoc 3170
Grids.TInplaceEdit.Visible 3170
Grids.TInplaceEdit.WndProc 3170
Grids.TInplaceEditList 3171
Grids.TInplaceEditList.ActiveList 3178
Grids.TInplaceEditList.BoundsChanged 3173
Grids.TInplaceEditList.ButtonRect 3173
Grids.TInplaceEditList.ButtonWidth 3178
Grids.TInplaceEditList.CloseUp 3174
Grids.TInplaceEditList.Create 3172
Grids.TInplaceEditList.DblClick 3174
Grids.TInplaceEditList.DoDropDownKeys 3174
Grids.TInplaceEditList.DoEditButtonClick 3174
Grids.TInplaceEditList.DoGetPickListItems 3174
Grids.TInplaceEditList.DropDown 3175
Grids.TInplaceEditList.DropDownRows 3178
Grids.TInplaceEditList.EditStyle 3178
Grids.TInplaceEditList.KeyDown 3175
Grids.TInplaceEditList.ListMouseUp 3175
Grids.TInplaceEditList.ListVisible 3179
Grids.TInplaceEditList.MouseDown 3175
Grids.TInplaceEditList.MouseMove 3176
Grids.TInplaceEditList.MouseUp 3176
Grids.TInplaceEditList.OnEditButtonClick 3180
Grids.TInplaceEditList.OnGetPickListitems 3180
Grids.TInplaceEditList.OverButton 3176
Grids.TInplaceEditList.PaintWindow 3176
Grids.TInplaceEditList.PickList 3179
Grids.TInplaceEditList.PickListLoaded 3179
Grids.TInplaceEditList.Pressed 3179
Grids.TInplaceEditList.RestoreContents 3176
Grids.TInplaceEditList.StopTracking 3177
Grids.TInplaceEditList.TrackButton 3177
Grids.TInplaceEditList.UpdateContents 3177
Grids.TInplaceEditList.WndProc 3177
Grids.TMovedEvent 3210
Grids.TMovedEvent type 3210
Grids.TOnGetPickListItems 3210
Grids.TOnGetPickListItems type 3210
Grids.TSelectCellEvent 3210
Grids.TSelectCellEvent type 3210
Grids.TSetEditEvent 3211
Grids.TSetEditEvent type 3211
Grids.TStringGrid 3180
Grids.TStringGrid.Cells 3188
Grids.TStringGrid.Cols 3189

Grids.TStringGrid.ColumnMoved 3186
Grids.TStringGrid.Create 3185
Grids.TStringGrid.Destroy 3186
Grids.TStringGrid.DrawCell 3187
Grids.TStringGrid.GetEditText 3187
Grids.TStringGrid.Objects 3190
Grids.TStringGrid.RowMoved 3187
Grids.TStringGrid.Rows 3190
Grids.TStringGrid.SetEditText 3188
Grids.TStringGridStrings 3192
Grids.TStringGridStrings.Add 3193
Grids.TStringGridStrings.Assign 3194
Grids.TStringGridStrings.Clear 3194
Grids.TStringGridStrings.Create 3193
Grids.TStringGridStrings.Delete 3195
Grids.TStringGridStrings.Get 3195
Grids.TStringGridStrings.GetCount 3195
Grids.TStringGridStrings.GetObject 3196
Grids.TStringGridStrings.Insert 3196
Grids.TStringGridStrings.Put 3197
Grids.TStringGridStrings.PutObject 3197
Grids.TStringGridStrings.SetUpdateState 3197
grMetaData enumeration member 6584
grReset enumeration member 6584
grXML enumeration member 6584
grXMLUTF8 enumeration member 6584
gsColMoving enumeration member 3203
gsColSizing enumeration member 3203
gsNormal enumeration member 3203
gsRowMoving enumeration member 3203
gsRowSizing enumeration member 3203
gsSelecting enumeration member 3203

H

haBaseline enumeration member 3521
haBottom enumeration member 3521
haCenter enumeration member 3521
haDefault enumeration member 3521
haLeft enumeration member 3521
haMiddle enumeration member 3521
haRight enumeration member 3521
haTop enumeration member 3521
haVDefault enumeration member 3521
HeaderPages enumeration member 4023
HelpIntfs 3213
HelpIntfs namespace 3213
    Classes 3214
    Functions 3242
    Interfaces 3215
HelpIntfs.EHelpSystemException 3214
HelpIntfs.GetHelpSystem 3242
HelpIntfs.GetHelpSystem function 3242
HelpIntfs.ICustomHelpViewer 3215
HelpIntfs.ICustomHelpViewer.CanShowTableOfContents 3216
HelpIntfs.ICustomHelpViewer.GetHelpStrings 3217
HelpIntfs.ICustomHelpViewer.GetViewerName 3217
HelpIntfs.ICustomHelpViewer.NotifyID 3218
HelpIntfs.ICustomHelpViewer.ShowHelp 3218
HelpIntfs.ICustomHelpViewer.ShowTableOfContents 3218
HelpIntfs.ICustomHelpViewer.ShutDown 3218
HelpIntfs.ICustomHelpViewer.SoftShutDown 3219
HelpIntfs.ICustomHelpViewer.UnderstandsKeyword 3219
HelpIntfs.IExtendedHelpViewer 3219
HelpIntfs.IExtendedHelpViewer.DisplayHelpByContext 3221
HelpIntfs.IExtendedHelpViewer.DisplayTopic 3221
HelpIntfs.IExtendedHelpViewer.UnderstandsContext 3221
HelpIntfs.IExtendedHelpViewer.UnderstandsTopic 3222
HelpIntfs.IExtendedHelpViewer2 3222
HelpIntfs.IExtendedHelpViewer2.DisplayIndex 3223
HelpIntfs.IExtendedHelpViewer2.DisplaySearch 3223
HelpIntfs.IExtendedHelpViewer2.DisplayTopic 3224
HelpIntfs.IFilteredHelpViewer 3224
HelpIntfs.IFilteredHelpViewer.GetFilter 3225
HelpIntfs.IFilteredHelpViewer.SetFilter 3226
HelpIntfs.IHelpManager 3226
HelpIntfs.IHelpManager.GetHandle 3227
HelpIntfs.IHelpManager.GetHelpFile 3227
HelpIntfs.IHelpManager.Release 3227

HelpIntfs.IHelpSelector 3228
HelpIntfs.IHelpSelector.SelectKeyword 3229
HelpIntfs.IHelpSelector.TableOfContents 3229
HelpIntfs.IHelpSelector2 3230
HelpIntfs.IHelpSelector2.SelectContext 3230
HelpIntfs.IHelpSystem 3231
HelpIntfs.IHelpSystem.AssignHelpSelector 3232
HelpIntfs.IHelpSystem.Hook 3232
HelpIntfs.IHelpSystem.ShowContextHelp 3232
HelpIntfs.IHelpSystem.ShowHelp 3233
HelpIntfs.IHelpSystem.ShowTableOfContents 3233
HelpIntfs.IHelpSystem.ShowTopicHelp 3233
HelpIntfs.IHelpSystem2 3234
HelpIntfs.IHelpSystem2.UnderstandsKeyword 3235
HelpIntfs.IHelpSystem3 3235
HelpIntfs.IHelpSystem3.GetFilter 3236
HelpIntfs.IHelpSystem3.SetFilter 3236
HelpIntfs.IHelpSystem3.ShowIndex 3237
HelpIntfs.IHelpSystem3.ShowSearch 3237
HelpIntfs.IHelpSystem3.ShowTopicHelp 3237
HelpIntfs.IHelpSystemFlags 3238
HelpIntfs.IHelpSystemFlags.GetUseDefaultTopic 3239
HelpIntfs.IHelpSystemFlags.SetUseDefaultTopic 3239
HelpIntfs.ISpecialWinHelpViewer 3239
HelpIntfs.ISpecialWinHelpViewer.CallWinHelp 3241
HelpIntfs.RegisterViewer 3242
HelpIntfs.RegisterViewer function 3242
hmtAll enumeration member 4475
hmtRequest enumeration member 4475
hmtResponse enumeration member 4475
hsGradient enumeration member 721
hsHot enumeration member 701
hsImage enumeration member 721
hsNormal enumeration member 701
hsPressed enumeration member 701
hsThemed enumeration member 721
HTMLHelpViewer 3245
HTMLHelpViewer namespace 3245
Interfaces 3246
Variables 3249
HTMLHelpViewer.HtmlHelpTester 3249
HTMLHelpViewer.HtmlHelpTester variable 3249
HTMLHelpViewer.IHtmlHelpTester 3246
HTMLHelpViewer.IHtmlHelpTester.CanShowALink 3247
HTMLHelpViewer.IHtmlHelpTester.CanShowContext 3247
HTMLHelpViewer.IHtmlHelpTester.CanShowTopic 3247
HTMLHelpViewer.IHtmlHelpTester.ConvertWinHelp 3247
HTMLHelpViewer.IHtmlHelpTester.GetDefaultHelpFile 3248
HTMLHelpViewer.IHtmlHelpTester.GetHelpPath 3248
HTMLHelpViewer.IHtmlHelpTester.GetHelpStrings 3248
HTMLHelpViewer.ViewerName 3249
HTMLHelpViewer.ViewerName variable 3249
HTTPApp 3251
HTTPApp namespace 3251
Classes 3254
Constants 3448
Functions 3436
Interfaces 3416
Structs, Records, Enums 3444
Types 3446
HTTPApp.DayOfWeekStr 3436
HTTPApp.DayOfWeekStr function 3436
HTTPApp.DosPathToUnixPath 3437
HTTPApp.DosPathToUnixPath function 3437
HTTPApp.EncodingFromContentType 3437
HTTPApp.EncodingFromContentType function 3437
HTTPApp.EncodingGetBytes 3437
HTTPApp.EncodingGetBytes function 3437
HTTPApp.EncodingGetString 3438
HTTPApp.EncodingGetString function 3438
HTTPApp.EWebBrokerException 3254
HTTPApp.ExtractHeaderFields 3438
HTTPApp.ExtractHeaderFields function 3438
HTTPApp.ExtractHTTPFields 3439
HTTPApp.ExtractHTTPFields function 3439
HTTPApp.HTMLDecode 3439
HTTPApp.HTMLDecode function 3439
HTTPApp.HTMLEncode 3440

HTTPApp.HTMLEncode function 3440
HTTPApp.HTTPDecode 3440
HTTPApp.HTTPDecode function 3440
HTTPApp.HTTPEncode 3441
HTTPApp.HTTPEncode function 3441
HTTPApp.IGetAppDispatcher 3416
HTTPApp.IGetAppDispatcher.GetAppDispatcher 3417
HTTPApp.IGetWebAppServices 3417
HTTPApp.IGetWebAppServices.GetWebAppServices 3418
HTTPApp.IGetWebRequestHandler 3418
HTTPApp.IGetWebRequestHandler.GetWebRequestHandler 3419
HTTPApp.IMultiModuleSupport 3419
HTTPApp.IMultiModuleSupport.FinishContext 3420
HTTPApp.IMultiModuleSupport.InitContext 3420
HTTPApp.IMultiModuleSupport.InitModule 3420
HTTPApp.IProduceContent 3420
HTTPApp.IProduceContent.ProduceContent 3421
HTTPApp.IProduceContentFrom 3422
HTTPApp.IProduceContentFrom.ProduceContentFromStream 3422
HTTPApp.IProduceContentFrom.ProduceContentFromString 3423
HTTPApp.ISetAppDispatcher 3423
HTTPApp.ISetAppDispatcher.SetAppDispatcher 3424
HTTPApp.IWebAppServices 3424
HTTPApp.IWebAppServices.ExceptionHandler 3427
HTTPApp.IWebAppServices.FinishContext 3425
HTTPApp.IWebAppServices.GetExceptionHandler 3426
HTTPApp.IWebAppServices.HandleRequest 3426
HTTPApp.IWebAppServices.InitContext 3426
HTTPApp.IWebDispatch 3427
HTTPApp.IWebDispatch.DispatchEnabled 3428
HTTPApp.IWebDispatch.DispatchMask 3429
HTTPApp.IWebDispatch.DispatchMethodType 3429
HTTPApp.IWebDispatch.DispatchRequest 3429
HTTPApp.IWebDispatch.Enabled 3429
HTTPApp.IWebDispatch.Mask 3430
HTTPApp.IWebDispatch.MethodType 3430
HTTPApp.IWebDispatchActions 3430
HTTPApp.IWebDispatcherAccess 3431
HTTPApp.IWebDispatcherAccess.Request 3432
HTTPApp.IWebDispatcherAccess.Response 3432
HTTPApp.IWebExceptionHandler 3433
HTTPApp.IWebExceptionHandler.HandleException 3433
HTTPApp.IWebRequestHandler 3434
HTTPApp.IWebRequestHandler.HandleRequest 3435
HTTPApp.MAX_DATETIMES 3448
HTTPApp.MAX_DATETIMES constant 3448
HTTPApp.MAX_INTEGERS 3448
HTTPApp.MAX_INTEGERS constant 3448
HTTPApp.MAX_STRINGS 3449
HTTPApp.MAX_STRINGS constant 3449
HTTPApp.MonthStr 3441
HTTPApp.MonthStr function 3441
HTTPApp.ParseDate 3442
HTTPApp.ParseDate function 3442
HTTPApp.RegisterContentParser 3442
HTTPApp.RegisterContentParser function 3442
HTTPApp.sDateFormat 3449
HTTPApp.sDateFormat constant 3449
HTTPApp.StatusString 3442
HTTPApp.StatusString function 3442
HTTPApp.TAbstractContentParser 3255
HTTPApp.TAbstractContentParser.CanParse 3257
HTTPApp.TAbstractContentParser.Create 3256
HTTPApp.TAbstractContentParser.GetContentFields 3257
HTTPApp.TAbstractContentParser.GetFiles 3258
HTTPApp.TAbstractContentParser.WebRequest 3258
HTTPApp.TAbstractWebModuleFactory 3259
HTTPApp.TAbstractWebModuleFactory.CacheMode 3264
HTTPApp.TAbstractWebModuleFactory.ComponentClass 3264
HTTPApp.TAbstractWebModuleFactory.CreateMode 3265
HTTPApp.TAbstractWebModuleFactory.GetCacheMode 3260
HTTPApp.TAbstractWebModuleFactory.GetComponentClass 3261
HTTPApp.TAbstractWebModuleFactory.GetCreateMode 3261
HTTPApp.TAbstractWebModuleFactory.GetIsAppModule 3261
HTTPApp.TAbstractWebModuleFactory.GetModule 3262
HTTPApp.TAbstractWebModuleFactory.GetModuleName 3262

HTTPApp.TAbstractWebModuleFactory.GetWebPageInfo 3263
HTTPApp.TAbstractWebModuleFactory.IsAppModule 3265
HTTPApp.TAbstractWebModuleFactory.ModuleName 3265
HTTPApp.TAbstractWebModuleFactory.PreventDestruction 3263
HTTPApp.TAbstractWebModuleFactory.WebPageInfo 3266
HTTPApp.TAbstractWebModuleList 3266
HTTPApp.TAbstractWebModuleList.AddModuleClass 3268
HTTPApp.TAbstractWebModuleList.AddModuleName 3269
HTTPApp.TAbstractWebModuleList.Factory 3272
HTTPApp.TAbstractWebModuleList.FactoryCount 3272
HTTPApp.TAbstractWebModuleList.FindModuleClass 3269
HTTPApp.TAbstractWebModuleList.FindModuleName 3270
HTTPApp.TAbstractWebModuleList.GetFactory 3270
HTTPApp.TAbstractWebModuleList.GetFactoryCount 3270
HTTPApp.TAbstractWebModuleList.GetItem 3271
HTTPApp.TAbstractWebModuleList.GetItemCount 3271
HTTPApp.TAbstractWebModuleList.GetOnModuleAdded 3271
HTTPApp.TAbstractWebModuleList.ItemCount 3273
HTTPApp.TAbstractWebModuleList.Items 3273
HTTPApp.TAbstractWebModuleList.OnModuleAdded 3274
HTTPApp.TAbstractWebModuleList.SetOnModuleAdded 3271
HTTPApp.TAbstractWebPageInfo 3275
HTTPApp.TAbstractWebPageInfo.Factory 3281
HTTPApp.TAbstractWebPageInfo.GetFactory 3277
HTTPApp.TAbstractWebPageInfo.GetIsLoginRequired 3277
HTTPApp.TAbstractWebPageInfo.GetIsPublished 3277
HTTPApp.TAbstractWebPageInfo.GetPageDescription 3278
HTTPApp.TAbstractWebPageInfo.GetPageFile 3278
HTTPApp.TAbstractWebPageInfo.GetPageHREF 3278
HTTPApp.TAbstractWebPageInfo.GetPageName 3279
HTTPApp.TAbstractWebPageInfo.GetPageTitle 3279
HTTPApp.TAbstractWebPageInfo.GetViewAccess 3279
HTTPApp.TAbstractWebPageInfo.IsLoginRequired 3281
HTTPApp.TAbstractWebPageInfo.IsPublished 3282
HTTPApp.TAbstractWebPageInfo.PageDescription 3282
HTTPApp.TAbstractWebPageInfo.PageFile 3283
HTTPApp.TAbstractWebPageInfo.PageHREF 3283
HTTPApp.TAbstractWebPageInfo.PageName 3283
HTTPApp.TAbstractWebPageInfo.PageTitle 3284
HTTPApp.TAbstractWebPageInfo.SetFactory 3280
HTTPApp.TAbstractWebPageInfo.ViewAccess 3284
HTTPApp.TAbstractWebPageModuleFactory 3285
HTTPApp.TAbstractWebPageModuleFactory.Create 3286
HTTPApp.TAbstractWebPageModuleFactory.Destroy 3286
HTTPApp.TAbstractWebPageModuleFactory.GetWebPageInfo 3287
HTTPApp.TAbstractWebRequestFile 3288
HTTPApp.TAbstractWebRequestFile.ContentType 3291
HTTPApp.TAbstractWebRequestFile.FieldName 3291
HTTPApp.TAbstractWebRequestFile.FileName 3291
HTTPApp.TAbstractWebRequestFile.GetContentType 3289
HTTPApp.TAbstractWebRequestFile.GetFieldName 3289
HTTPApp.TAbstractWebRequestFile.GetFileName 3290
HTTPApp.TAbstractWebRequestFile.GetStream 3290
HTTPApp.TAbstractWebRequestFile.Stream 3292
HTTPApp.TAbstractWebRequestFiles 3292
HTTPApp.TAbstractWebRequestFiles.Count 3294
HTTPApp.TAbstractWebRequestFiles.GetCount 3293
HTTPApp.TAbstractWebRequestFiles.GetItem 3294
HTTPApp.TAbstractWebRequestFiles.Items 3295
HTTPApp.TAbstractWebSession 3295
HTTPApp.TAbstractWebSession.GetSessionID 3297
HTTPApp.TAbstractWebSession.GetTimeoutMinutes 3297
HTTPApp.TAbstractWebSession.GetValue 3298
HTTPApp.TAbstractWebSession.SessionID 3300
HTTPApp.TAbstractWebSession.SetTimeoutMinutes 3298
HTTPApp.TAbstractWebSession.SetValue 3299
HTTPApp.TAbstractWebSession.Terminate 3299
HTTPApp.TAbstractWebSession.TimeoutMinutes 3300
HTTPApp.TAbstractWebSession.UpdateResponse 3299
HTTPApp.TAbstractWebSession.Values 3301
HTTPApp.TContentParser 3301
HTTPApp.TContentParser.CanParse 3303
HTTPApp.TContentParser.Destroy 3302
HTTPApp.TContentParser.GetContentFields 3303
HTTPApp.TContentParser.GetFiles 3304
HTTPApp.TContentParserClass 3446
HTTPApp.TContentParserClass type 3446
HTTPApp.TCookie 3304

HTTPApp.TCookie.AssignTo 3306
HTTPApp.TCookie.Create 3305
HTTPApp.TCookie.Domain 3307
HTTPApp.TCookie.Expires 3307
HTTPApp.TCookie.GetHeaderValue 3306
HTTPApp.TCookie.HeaderValue 3307
HTTPApp.TCookie.Name 3308
HTTPApp.TCookie.Path 3308
HTTPApp.TCookie.Secure 3309
HTTPApp.TCookie.Value 3309
HTTPApp.TCookieCollection 3309
HTTPApp.TCookieCollection.Add 3311
HTTPApp.TCookieCollection.Create 3310
HTTPApp.TCookieCollection.GetCookie 3311
HTTPApp.TCookieCollection.Items 3312
HTTPApp.TCookieCollection.SetCookie 3311
HTTPApp.TCookieCollection.WebResponse 3312
HTTPApp.TCustomContentProducer 3313
HTTPApp.TCustomContentProducer.Content 3314
HTTPApp.TCustomContentProducer.ContentFromStream 3315
HTTPApp.TCustomContentProducer.ContentFromString 3315
HTTPApp.TCustomContentProducer.Dispatcher 3317
HTTPApp.TCustomContentProducer.DispatcherComponent 3318
HTTPApp.TCustomContentProducer.GetAppDispatcher 3316
HTTPApp.TCustomContentProducer.Notification 3316
HTTPApp.TCustomContentProducer.ProduceContent 3316
HTTPApp.TCustomContentProducer.ProduceContentFromStream 3317
HTTPApp.TCustomContentProducer.ProduceContentFromString 3317
HTTPApp.TCustomContentProducer.SetAppDispatcher 3317
HTTPApp.TCustomWebDispatcher 3318
HTTPApp.TCustomWebDispatcher.Access_Request 3322
HTTPApp.TCustomWebDispatcher.Access_Response 3322
HTTPApp.TCustomWebDispatcher.Action 3327
HTTPApp.TCustomWebDispatcher.ActionByName 3322
HTTPApp.TCustomWebDispatcher.Actions 3327
HTTPApp.TCustomWebDispatcher.AfterDispatch 3329
HTTPApp.TCustomWebDispatcher.BeforeDispatch 3329
HTTPApp.TCustomWebDispatcher.Create 3320
HTTPApp.TCustomWebDispatcher.Destroy 3321
HTTPApp.TCustomWebDispatcher.DispatchAction 3323
HTTPApp.TCustomWebDispatcher.DoAfterDispatch 3324
HTTPApp.TCustomWebDispatcher.DoBeforeDispatch 3324
HTTPApp.TCustomWebDispatcher.FinishContext 3325
HTTPApp.TCustomWebDispatcher.HandleException 3325
HTTPApp.TCustomWebDispatcher.HandleRequest 3325
HTTPApp.TCustomWebDispatcher.InitContext 3326
HTTPApp.TCustomWebDispatcher.InitModule 3326
HTTPApp.TCustomWebDispatcher.Notification 3326
HTTPApp.TCustomWebDispatcher.OnException 3330
HTTPApp.TCustomWebDispatcher.Request 3327
HTTPApp.TCustomWebDispatcher.Response 3328
HTTPApp.TDefaultWebAppServices 3330
HTTPApp.TDefaultWebAppServices.FindRequestHandler 3331
HTTPApp.TDefaultWebAppServices.FinishContext 3332
HTTPApp.TDefaultWebAppServices.GetExceptionHandler 3332
HTTPApp.TDefaultWebAppServices.GetRequestHandler 3332
HTTPApp.TDefaultWebAppServices.HandleRequest 3332
HTTPApp.TDefaultWebAppServices.InitContext 3333
HTTPApp.TDefaultWebAppServices.InvokeDispatcher 3334
HTTPApp.TDefaultWebAppServices.Request 3334
HTTPApp.TDefaultWebAppServices.RequestHandler 3334
HTTPApp.TDefaultWebAppServices.Response 3334
HTTPApp.TDefaultWebAppServices.WebModules 3335
HTTPApp.THTTPMethodEvent 3446
HTTPApp.THTTPMethodEvent type 3446
HTTPApp.TMethodType 3444
HTTPApp.TMethodType enumeration 3444
HTTPApp.TModuleAddedProc 3447
HTTPApp.TModuleAddedProc type 3447
HTTPApp.TWebActionItem 3335
HTTPApp.TWebActionItem.AssignTo 3337
HTTPApp.TWebActionItem.Create 3336
HTTPApp.TWebActionItem.Default 3339
HTTPApp.TWebActionItem.Destroy 3337
HTTPApp.TWebActionItem.Enabled 3340

HTTPApp.TWebActionItem.GetDisplayName 3338
HTTPApp.TWebActionItem.GetPathInfo 3338
HTTPApp.TWebActionItem.MethodType 3341
HTTPApp.TWebActionItem.Name 3341
HTTPApp.TWebActionItem.OnAction 3343
HTTPApp.TWebActionItem.PathInfo 3341
HTTPApp.TWebActionItem.Producer 3342
HTTPApp.TWebActionItem.ProducerContent 3342
HTTPApp.TWebActionItem.SetDisplayName 3338
HTTPApp.TWebActionItems 3344
HTTPApp.TWebActionItems.Add 3346
HTTPApp.TWebActionItems.Create 3345
HTTPApp.TWebActionItems.GetAttr 3346
HTTPApp.TWebActionItems.GetAttrCount 3346
HTTPApp.TWebActionItems.GetItemAttr 3347
HTTPApp.TWebActionItems.GetOwner 3347
HTTPApp.TWebActionItems.Items 3349
HTTPApp.TWebActionItems.SetItemName 3348
HTTPApp.TWebActionItems.Update 3348
HTTPApp.TWebActionItems.WebDispatcher 3349
HTTPApp.TWebDispatcher 3349
HTTPApp.TWebDispatcher.Actions 3351
HTTPApp.TWebDispatcher.AfterDispatch 3352
HTTPApp.TWebDispatcher.BeforeDispatch 3353
HTTPApp.TWebDispatcher.OnException 3353
HTTPApp.TWebExceptionEvent 3447
HTTPApp.TWebExceptionEvent type 3447
HTTPApp.TWebModule 3354
HTTPApp.TWebModule.Actions 3356
HTTPApp.TWebModule.AfterDispatch 3357
HTTPApp.TWebModule.BeforeDispatch 3357
HTTPApp.TWebModule.Create 3356
HTTPApp.TWebModule.OnException 3358
HTTPApp.TWebModuleCacheMode 3444
HTTPApp.TWebModuleCacheMode enumeration 3444
HTTPApp.TWebModuleContext 3447
HTTPApp.TWebModuleContext type 3447
HTTPApp.TWebModuleCreateMode 3445
HTTPApp.TWebModuleCreateMode enumeration 3445
HTTPApp.TWebRequest 3358
HTTPApp.TWebRequest.Accept 3371
HTTPApp.TWebRequest.Authorization 3372
HTTPApp.TWebRequest.CacheControl 3372
HTTPApp.TWebRequest.Connection 3373
HTTPApp.TWebRequest.Content 3373
HTTPApp.TWebRequest.ContentEncoding 3374
HTTPApp.TWebRequest.ContentFields 3375
HTTPApp.TWebRequest.ContentLength 3375
HTTPApp.TWebRequest.ContentParser 3376
HTTPApp.TWebRequest.ContentType 3376
HTTPApp.TWebRequest.ContentVersion 3376
HTTPApp.TWebRequest.Cookie 3377
HTTPApp.TWebRequest.CookieFields 3377
HTTPApp.TWebRequest.Create 3361
HTTPApp.TWebRequest.Date 3378
HTTPApp.TWebRequest.DerivedFrom 3378
HTTPApp.TWebRequest.Destroy 3361
HTTPApp.TWebRequest.Expires 3379
HTTPApp.TWebRequest.ExtractContentFields 3363
HTTPApp.TWebRequest.ExtractCookieFields 3363
HTTPApp.TWebRequest.ExtractFields 3364
HTTPApp.TWebRequest.ExtractQueryFields 3364
HTTPApp.TWebRequest.Files 3379
HTTPApp.TWebRequest.From 3379
HTTPApp.TWebRequest.GetDateVariable 3365
HTTPApp.TWebRequest.GetFieldByName 3365
HTTPApp.TWebRequest.GetIntegerVariable 3366
HTTPApp.TWebRequest.GetInternalPathInfo 3366
HTTPApp.TWebRequest.GetInternalScriptName 3366
HTTPApp.TWebRequest.GetStringVariable 3366
HTTPApp.TWebRequest.Host 3380
HTTPApp.TWebRequest.IfModifiedSince 3380
HTTPApp.TWebRequest.InternalPathInfo 3381
HTTPApp.TWebRequest.InternalScriptName 3381
HTTPApp.TWebRequest.Method 3382
HTTPApp.TWebRequest.MethodType 3382
HTTPApp.TWebRequest.PathInfo 3382
HTTPApp.TWebRequest.PathTranslated 3383

HTTPApp.TWebRequest.ProtocolVersion 3383
HTTPApp.TWebRequest.Query 3384
HTTPApp.TWebRequest.QueryFields 3384
HTTPApp.TWebRequest.RawContent 3385
HTTPApp.TWebRequest.ReadClient 3367
HTTPApp.TWebRequest.ReadString 3368
HTTPApp.TWebRequest.ReadUnicodeString 3368
HTTPApp.TWebRequest.Referer 3385
HTTPApp.TWebRequest.RemoteAddr 3386
HTTPApp.TWebRequest.RemoteHost 3386
HTTPApp.TWebRequest.ScriptName 3386
HTTPApp.TWebRequest.ServerPort 3387
HTTPApp.TWebRequest.Title 3387
HTTPApp.TWebRequest.TranslateURI 3368
HTTPApp.TWebRequest.UpdateMethodType 3369
HTTPApp.TWebRequest.URL 3388
HTTPApp.TWebRequest.UserAgent 3388
HTTPApp.TWebRequest.WriteClient 3369
HTTPApp.TWebRequest.WriteHeaders 3370
HTTPApp.TWebRequest.WriteString 3370
HTTPApp.TWebResponse 3389
HTTPApp.TWebResponse.AddCustomHeaders 3394
HTTPApp.TWebResponse.Allow 3404
HTTPApp.TWebResponse.Content 3404
HTTPApp.TWebResponse.ContentEncoding 3405
HTTPApp.TWebResponse.ContentLength 3405
HTTPApp.TWebResponse.ContentStream 3406
HTTPApp.TWebResponse.ContentType 3406
HTTPApp.TWebResponse.ContentVersion 3407
HTTPApp.TWebResponse.Cookies 3407
HTTPApp.TWebResponse.Create 3391
HTTPApp.TWebResponse.CustomHeaders 3408
HTTPApp.TWebResponse.Date 3408
HTTPApp.TWebResponse.DerivedFrom 3409
HTTPApp.TWebResponse.Destroy 3392
HTTPApp.TWebResponse.Expires 3409
HTTPApp.TWebResponse.FCustomHeaders 3392
HTTPApp.TWebResponse.FHTTPRequest 3393
HTTPApp.TWebResponse.FormatAuthenticate 3394
HTTPApp.TWebResponse.GetContent 3395
HTTPApp.TWebResponse.GetCustomHeader 3395
HTTPApp.TWebResponse.GetDateVariable 3395
HTTPApp.TWebResponse.GetIntegerVariable 3396
HTTPApp.TWebResponse.GetLogMessage 3396
HTTPApp.TWebResponse.GetStatusCode 3396
HTTPApp.TWebResponse.GetStringVariable 3397
HTTPApp.TWebResponse.HTTPRequest 3410
HTTPApp.TWebResponse.LastModified 3410
HTTPApp.TWebResponse.Location 3411
HTTPApp.TWebResponse.LogMessage 3411
HTTPApp.TWebResponse.RawContent 3411
HTTPApp.TWebResponse.Realm 3411
HTTPApp.TWebResponse.ReasonString 3412
HTTPApp.TWebResponse.SendRedirect 3398
HTTPApp.TWebResponse.SendResponse 3398
HTTPApp.TWebResponse.SendStream 3398
HTTPApp.TWebResponse.Sent 3399
HTTPApp.TWebResponse.Server 3412
HTTPApp.TWebResponse.SetContent 3399
HTTPApp.TWebResponse.SetContentStream 3400
HTTPApp.TWebResponse.SetCookieField 3400
HTTPApp.TWebResponse.SetCustomHeader 3400
HTTPApp.TWebResponse.SetDateVariable 3401
HTTPApp.TWebResponse.SetIntegerVariable 3401
HTTPApp.TWebResponse.SetLogMessage 3402
HTTPApp.TWebResponse.SetStatusCode 3402
HTTPApp.TWebResponse.SetStringVariable 3402
HTTPApp.TWebResponse.StatusCode 3413
HTTPApp.TWebResponse.Title 3414
HTTPApp.TWebResponse.Version 3415
HTTPApp.TWebResponse.WWWAuthenticate 3415
HTTPApp.UnixPathToDosPath 3443
HTTPApp.UnixPathToDosPath function 3443
HTTPProd 3451
HTTPProd namespace 3451
Classes 3453
Constants 3526
Functions 3518

Interfaces 3502
Structs, Records, Enums 3521
Types 3523
Variables 3525
HTTPProd.ContentFromScriptFile 3518
HTTPProd.ContentFromScriptFile function 3518
HTTPProd.ContentFromScriptStream 3518
HTTPProd.ContentFromScriptStream function 3518
HTTPProd.ContentFromStream 3519
HTTPProd.ContentFromStream function 3519
HTTPProd.ContentFromString 3519
HTTPProd.ContentFromString function 3519
HTTPProd.DesignerFileManager 3525
HTTPProd.DesignerFileManager variable 3525
HTTPProd.EScriptError 3453
HTTPProd.EScriptError.Content 3455
HTTPProd.EScriptError.Create 3454
HTTPProd.EScriptError.Destroy 3454
HTTPProd.EScriptError.Errors 3455
HTTPProd.FindComponentWebModuleContext 3519
HTTPProd.FindComponentWebModuleContext function 3519
HTTPProd.GetEncodingOfStream 3520
HTTPProd.GetEncodingOfStream function 3520
HTTPProd.GetTagID 3520
HTTPProd.GetTagID function 3520
HTTPProd.HTMLAlign 3526
HTTPProd.HTMLAlign constant 3526
HTTPProd.HTMLVAlign 3526
HTTPProd.HTMLVAlign constant 3526
HTTPProd.IDesignerFileManager 3502
HTTPProd.IDesignerFileManager.GetStream 3503
HTTPProd.IDesignerFileManager.QualifyFileName 3503
HTTPProd.IGetLocateFileService 3503
HTTPProd.IGetLocateFileService.GetLocateFileService 3504
HTTPProd.IGetProducerTemplate 3504
HTTPProd.IGetProducerTemplate.GetProducerTemplateFileName 3505
HTTPProd.IGetProducerTemplate.GetProducerTemplateStream 3505
HTTPProd.ILocateFileService 3505
HTTPProd.ILocateFileService.GetTemplateStream 3506
HTTPProd.IScriptContext 3506
HTTPProd.IScriptContext.GetWebModuleContext 3507
HTTPProd.IScriptContext.WebModuleContext 3507
HTTPProd.IScriptProducer 3508
HTTPProd.IScriptProducer.Content 3512
HTTPProd.IScriptProducer.Errors 3512
HTTPProd.IScriptProducer.Evaluate 3509
HTTPProd.IScriptProducer.GetContent 3509
HTTPProd.IScriptProducer.GetErrors 3509
HTTPProd.IScriptProducer.GetHTMLBlock 3510
HTTPProd.IScriptProducer.GetHTMLBlockCount 3510
HTTPProd.IScriptProducer.HandleScriptError 3510
HTTPProd.IScriptProducer.HTMLBlockCount 3513
HTTPProd.IScriptProducer.HTMLBlocks 3513
HTTPProd.IScriptProducer.ParseStream 3510
HTTPProd.IScriptProducer.ParseString 3510
HTTPProd.IScriptProducer.ReplaceTags 3511
HTTPProd.IScriptProducer.SetContent 3511
HTTPProd.IScriptProducer.Write 3511
HTTPProd.IScriptProducer.WriteItem 3512
HTTPProd.IWebVariableName 3513
HTTPProd.IWebVariableName.GetVariableName 3514
HTTPProd.IWebVariableName.VariableName 3514
HTTPProd.IWebVariablesContainer 3514
HTTPProd.IWebVariablesContainer.FindVariable 3515
HTTPProd.IWebVariablesContainer.FindVariableContainer 3516
HTTPProd.IWebVariablesContainer.GetVariable 3516
HTTPProd.IWebVariablesContainer.GetVariableCount 3516
HTTPProd.IWebVariablesContainer.VariableCount 3516
HTTPProd.IWebVariablesContainer.Variables 3517
HTTPProd.ScriptEnginesList 3525
HTTPProd.ScriptEnginesList variable 3525
HTTPProd.TAbstractScriptEnginesList 3455
HTTPProd.TAbstractScriptEnginesList.Create 3456
HTTPProd.TAbstractScriptEnginesList.FindScriptProducerClass 3456
HTTPProd.TAbstractScriptError 3457
HTTPProd.TAbstractScriptError.CharPos 3459

HTTPProd.TAbstractScriptError.Description 3459
HTTPProd.TAbstractScriptError.FileName 3460
HTTPProd.TAbstractScriptError.GetCharPos 3458
HTTPProd.TAbstractScriptError.GetDescription 3458
HTTPProd.TAbstractScriptError.GetFileName 3458
HTTPProd.TAbstractScriptError.GetLine 3458
HTTPProd.TAbstractScriptError.GetSourceLine 3458
HTTPProd.TAbstractScriptError.Line 3460
HTTPProd.TAbstractScriptError.SetDescription 3459
HTTPProd.TAbstractScriptError.SourceLine 3460
HTTPProd.TAbstractScriptErrors 3460
HTTPProd.TAbstractScriptErrors.Add 3461
HTTPProd.TAbstractScriptErrors.Count 3462
HTTPProd.TAbstractScriptErrors.Errors 3462
HTTPProd.TAbstractScriptErrors.GetError 3461
HTTPProd.TAbstractScriptErrors.GetErrorCount 3462
HTTPProd.TAbstractScriptProducer 3463
HTTPProd.TAbstractScriptProducer.Create 3463
HTTPProd.TBasePageProducer 3464
HTTPProd.TBasePageProducer.Content 3466
HTTPProd.TBasePageProducer.ContentFromStream 3467
HTTPProd.TBasePageProducer.ContentFromString 3467
HTTPProd.TBasePageProducer.Create 3466
HTTPProd.TBasePageProducer.DoTagEvent 3468
HTTPProd.TBasePageProducer.GetProducerTemplateFileNam e 3468
HTTPProd.TBasePageProducer.GetProducerTemplateStream 3468
HTTPProd.TBasePageProducer.GetScriptEngine 3469
HTTPProd.TBasePageProducer.GetTagID 3469
HTTPProd.TBasePageProducer.GetTemplateFileName 3469
HTTPProd.TBasePageProducer.GetTemplateStream 3469
HTTPProd.TBasePageProducer.HandleScriptTag 3469
HTTPProd.TBasePageProducer.HandleTag 3470
HTTPProd.TBasePageProducer.ImplHandleTag 3470
HTTPProd.TBasePageProducer.OnHTMLTag 3472
HTTPProd.TBasePageProducer.ScriptEngine 3471
HTTPProd.TBasePageProducer.ServerScriptFromStream 3470
HTTPProd.TBasePageProducer.StripParamQuotes 3471
HTTPProd.TBasePageProducer.UseScriptEngine 3470
HTTPProd.TBasePageProducer.WebModuleContext 3471
HTTPProd.TCustomPageProducer 3473
HTTPProd.TCustomPageProducer.Create 3474
HTTPProd.TCustomPageProducer.Destroy 3475
HTTPProd.TCustomPageProducer.GetTemplateFileName 3475
HTTPProd.TCustomPageProducer.GetTemplateStream 3476
HTTPProd.TCustomPageProducer.HandleTag 3476
HTTPProd.TCustomPageProducer.HTMLDoc 3476
HTTPProd.TCustomPageProducer.HTMLFile 3477
HTTPProd.THandledTagProc 3523
HTTPProd.THandledTagProc type 3523
HTTPProd.THandleTagProc 3523
HTTPProd.THandleTagProc type 3523
HTTPProd.THTMLAlign 3521
HTTPProd.THTMLAlign enumeration 3521
HTTPProd.THTMLBgColor 3524
HTTPProd.THTMLBgColor type 3524
HTTPProd.THTMLTableAttributes 3478
HTTPProd.THTMLTableAttributes.Align 3481
HTTPProd.THTMLTableAttributes.AssignTo 3480
HTTPProd.THTMLTableAttributes.BgColor 3482
HTTPProd.THTMLTableAttributes.Border 3482
HTTPProd.THTMLTableAttributes.CellPadding 3483
HTTPProd.THTMLTableAttributes.CellSpacing 3483
HTTPProd.THTMLTableAttributes.Create 3479
HTTPProd.THTMLTableAttributes.RestoreDefaults 3480
HTTPProd.THTMLTableAttributes.Width 3483
HTTPProd.THTMLTableCellAttributes 3484
HTTPProd.THTMLTableElementAttributes 3484
HTTPProd.THTMLTableElementAttributes.Align 3487
HTTPProd.THTMLTableElementAttributes.AssignTo 3486
HTTPProd.THTMLTableElementAttributes.BgColor 3487
HTTPProd.THTMLTableElementAttributes.RestoreDefaults 3486
HTTPProd.THTMLTableElementAttributes.VAlign 3488
HTTPProd.THTMLTableHeaderAttributes 3488
HTTPProd.THTMLTableHeaderAttributes.AssignTo 3490
HTTPProd.THTMLTableHeaderAttributes.Caption 3491
HTTPProd.THTMLTableHeaderAttributes.RestoreDefaults 3490

HTTPProd.THTMLTableRowAttributes 3491
HTTPProd.THTMLTagAttributes 3492
HTTPProd.THTMLTagAttributes.Changed 3494
HTTPProd.THTMLTagAttributes.Create 3493
HTTPProd.THTMLTagAttributes.Custom 3494
HTTPProd.THTMLTagAttributes.OnChange 3495
HTTPProd.THTMLTagAttributes.Producer 3495
HTTPProd.THTMLTagAttributes.RestoreDefaults 3494
HTTPProd.THTMLTagEvent 3524
HTTPProd.THTMLTagEvent type 3524
HTTPProd.THTMLVAlign 3521
HTTPProd.THTMLVAlign enumeration 3521
HTTPProd.TPageProducer 3496
HTTPProd.TPageProducer.HTMLDoc 3498
HTTPProd.TPageProducer.HTMLFile 3499
HTTPProd.TPageProducer.OnHTMLTag 3500
HTTPProd.TPageProducer.ScriptEngine 3500
HTTPProd.TPageProducer.StripParamQuotes 3500
HTTPProd.TScriptProducerClass 3524
HTTPProd.TScriptProducerClass type 3524
HTTPProd.TTag 3522
HTTPProd.TTag enumeration 3522
HTTPUtil 3527
HTTPUtil namespace 3527
- Functions 3537
- Interfaces 3528

HTTPUtil.FirstDelimiter 3537
HTTPUtil.FirstDelimiter function 3537
HTTPUtil.GetDefaultStreamLoader 3538
HTTPUtil.GetDefaultStreamLoader function 3538
HTTPUtil.HTMLEscape 3538
HTTPUtil.HTMLEscape function 3538
HTTPUtil.IStreamLoader 3528
HTTPUtil.IStreamLoader.GetPassword 3529
HTTPUtil.IStreamLoader.GetProxy 3530
HTTPUtil.IStreamLoader.GetTimeout 3530
HTTPUtil.IStreamLoader.GetUserName 3530
HTTPUtil.IStreamLoader.Load 3530
HTTPUtil.IStreamLoader.Password 3532
HTTPUtil.IStreamLoader.Proxy 3533
HTTPUtil.IStreamLoader.SetPassword 3531
HTTPUtil.IStreamLoader.SetProxy 3531
HTTPUtil.IStreamLoader.SetTimeout 3532
HTTPUtil.IStreamLoader.SetUserName 3532
HTTPUtil.IStreamLoader.Timeout 3533
HTTPUtil.IStreamLoader.UserName 3533
HTTPUtil.IStringTokenizer 3534
HTTPUtil.IStringTokenizer.countTokens 3536
HTTPUtil.IStringTokenizer.getTokenCounts 3535
HTTPUtil.IStringTokenizer.hasMoreTokens 3535
HTTPUtil.IStringTokenizer.nextToken 3535
HTTPUtil.StartsWith 3538
HTTPUtil.StartsWith function 3538
HTTPUtil.StringTokenizer 3539
HTTPUtil.StringTokenizer function 3539
HTTPUtil.StringToStringArray 3539
HTTPUtil.StringToStringArray function 3539

I

IAppServer interface 5222
- about IAppServer interface 5222
- AS_ApplyUpdates 5223
- AS_DataRequest 5224
- AS_Execute 5224
- AS_GetParams 5225
- AS_GetProviderNames 5226
- AS_GetRecords 5227
- AS_RowRequest 5228
- IAppServer Members 5223
- IAppServer methods 5223

IAppServerDisp interface 5229
- about IAppServerDisp interface 5229
- AS_ApplyUpdates 5230
- AS_DataRequest 5230
- AS_Execute 5230
- AS_GetParams 5230
- AS_GetProviderNames 5231
- AS_GetRecords 5231

AS_RowRequest 5231
IAppServerDisp Members 5229
IAppServerDisp methods 5229
IB 3541
IB namespace 3541
Classes 3543
Constants 3561
Functions 3547
Structs, Records, Enums 3551
Types 3557
Variables 3560
IB.CheckStatusVector 3547
IB.CheckStatusVector function 3547
IB.EIBClientError 3543
IB.EIBError 3543
IB.EIBError.Create 3544
IB.EIBError.IBErrorCode 3545
IB.EIBError.SQLCode 3545
IB.EIBInterBaseError 3545
IB.EIBInterBaseRoleError 3546
IB.GetIBDataBaseErrorMessages 3547
IB.GetIBDataBaseErrorMessages function 3547
IB.IBAlloc 3548
IB.IBAlloc function 3548
IB.IBBigLocalBufferLength 3561
IB.IBBigLocalBufferLength constant 3561
IB.IBCS 3560
IB.IBCS variable 3560
IB.IBDataBaseError 3548
IB.IBDataBaseError function 3548
IB.IBError 3548
IB.IBError function 3548
IB.IBErrorMessages 3561
IB.IBErrorMessages constant 3561
IB.IBHugeLocalBufferLength 3562
IB.IBHugeLocalBufferLength constant 3562
IB.IBLocalBufferLength 3563
IB.IBLocalBufferLength constant 3563
IB.IBPalette1 3563
IB.IBPalette1 constant 3563
IB.IBPalette2 3563
IB.IBPalette2 constant 3563
IB.PStatusVector 3557
IB.PStatusVector type 3557
IB.SetIBDataBaseErrorMessages 3549
IB.SetIBDataBaseErrorMessages function 3549
IB.StatusVector 3549
IB.StatusVector function 3549
IB.StatusVectorArray 3549
IB.StatusVectorArray function 3549
IB.StatusVectorAsText 3550
IB.StatusVectorAsText function 3550
IB.TIBClientError 3551
IB.TIBClientError enumeration 3551
IB.TIBDataBaseErrorMessage 3555
IB.TIBDataBaseErrorMessage enumeration 3555
IB.TIBDataBaseErrorMessages 3557
IB.TIBDataBaseErrorMessages type 3557
IB.TStatusVector 3558
IB.TStatusVector type 3558
IB.TTraceFlag 3555
IB.TTraceFlag enumeration 3555
IB.TTraceFlags 3558
IB.TTraceFlags type 3558
IBBatchMove 3565
IBBatchMove namespace 3565
Classes 3566
Structs, Records, Enums 3578
IBBatchMove.TBatchMode 3578
IBBatchMove.TBatchMode enumeration 3578
IBBatchMove.TIBBatchMove 3566
IBBatchMove.TIBBatchMove.AbortOnKeyViol 3570
IBBatchMove.TIBBatchMove.AbortOnProblem 3570
IBBatchMove.TIBBatchMove.ChangedCount 3571
IBBatchMove.TIBBatchMove.ChangedTableName 3571
IBBatchMove.TIBBatchMove.CommitCount 3571
IBBatchMove.TIBBatchMove.Create 3567
IBBatchMove.TIBBatchMove.Destination 3572

IBBatchMove.TIBBatchMove.Destroy 3568
IBBatchMove.TIBBatchMove.Execute 3568
IBBatchMove.TIBBatchMove.KeyViolCount 3572
IBBatchMove.TIBBatchMove.KeyViolTableName 3573
IBBatchMove.TIBBatchMove.Mappings 3573
IBBatchMove.TIBBatchMove.Mode 3574
IBBatchMove.TIBBatchMove.MovedCount 3574
IBBatchMove.TIBBatchMove.Notification 3569
IBBatchMove.TIBBatchMove.ProblemCount 3575
IBBatchMove.TIBBatchMove.ProblemTableName 3575
IBBatchMove.TIBBatchMove.RecordCount 3576
IBBatchMove.TIBBatchMove.Source 3576
IBBatchMove.TIBBatchMove.Transliterate 3577
IBBlob 3579
IBBlob namespace 3579
Classes 3580
Constants 3596
Functions 3594
IBBlob.DefaultBlobSegmentSize 3596
IBBlob.DefaultBlobSegmentSize constant 3596
IBBlob.GetBlobInfo 3594
IBBlob.GetBlobInfo function 3594
IBBlob.ReadBlob 3594
IBBlob.ReadBlob function 3594
IBBlob.TIBBlobStream 3580
IBBlob.TIBBlobStream.BlobID 3590
IBBlob.TIBBlobStream.BlobMaxSegmentSize 3591
IBBlob.TIBBlobStream.BlobNumSegments 3591
IBBlob.TIBBlobStream.BlobSize 3591
IBBlob.TIBBlobStream.BlobType 3591
IBBlob.TIBBlobStream.Call 3583
IBBlob.TIBBlobStream.Cancel 3583
IBBlob.TIBBlobStream.CheckReadable 3583
IBBlob.TIBBlobStream.CheckWritable 3584
IBBlob.TIBBlobStream.CloseBlob 3584
IBBlob.TIBBlobStream.Create 3581
IBBlob.TIBBlobStream.CreateBlob 3584
IBBlob.TIBBlobStream.Database 3592
IBBlob.TIBBlobStream.DBHandle 3592
IBBlob.TIBBlobStream.Destroy 3582
IBBlob.TIBBlobStream.EnsureBlobInitialized 3584
IBBlob.TIBBlobStream.Finalize 3585
IBBlob.TIBBlobStream.GetBlobInfo 3585
IBBlob.TIBBlobStream.GetDatabase 3585
IBBlob.TIBBlobStream.GetDBHandle 3585
IBBlob.TIBBlobStream.GetTransaction 3585
IBBlob.TIBBlobStream.GetTRHandle 3586
IBBlob.TIBBlobStream.Handle 3592
IBBlob.TIBBlobStream.LoadFromFile 3586
IBBlob.TIBBlobStream.LoadFromStream 3586
IBBlob.TIBBlobStream.Mode 3592
IBBlob.TIBBlobStream.Modified 3593
IBBlob.TIBBlobStream.OpenBlob 3586
IBBlob.TIBBlobStream.Read 3587
IBBlob.TIBBlobStream.SaveToFile 3587
IBBlob.TIBBlobStream.SaveToStream 3587
IBBlob.TIBBlobStream.Seek 3588
IBBlob.TIBBlobStream.SetBlobID 3588
IBBlob.TIBBlobStream.SetDatabase 3588
IBBlob.TIBBlobStream.SetMode 3589
IBBlob.TIBBlobStream.SetSize 3589
IBBlob.TIBBlobStream.SetTransaction 3589
IBBlob.TIBBlobStream.Transaction 3593
IBBlob.TIBBlobStream.TRHandle 3593
IBBlob.TIBBlobStream.Truncate 3589
IBBlob.TIBBlobStream.Write 3590
IBBlob.WriteBlob 3595
IBBlob.WriteBlob function 3595
IBCustomDataSet 3597
IBCustomDataSet namespace 3597
Classes 3599
Constants 3700
Structs, Records, Enums 3689
Types 3694
Variables 3699
IBCustomDataSet.BufferCacheSize 3700
IBCustomDataSet.BufferCacheSize constant 3700
IBCustomDataSet.CreateProviderProc 3699

IBCustomDataSet.CreateProviderProc variable 3699
IBCustomDataSet.DefaultFieldClasses 3700
IBCustomDataSet.DefaultFieldClasses constant 3700
IBCustomDataSet.PBlobDataArray 3694
IBCustomDataSet.PBlobDataArray type 3694
IBCustomDataSet.PDateTime 3694
IBCustomDataSet.PDateTime type 3694
IBCustomDataSet.PFieldData 3695
IBCustomDataSet.PFieldData type 3695
IBCustomDataSet.PIBDBKey 3695
IBCustomDataSet.PIBDBKey type 3695
IBCustomDataSet.PRecordData 3695
IBCustomDataSet.PRecordData type 3695
IBCustomDataSet.TBlobDataArray 3696
IBCustomDataSet.TBlobDataArray type 3696
IBCustomDataSet.TCachedUpdateStatus 3689
IBCustomDataSet.TCachedUpdateStatus enumeration 3689
IBCustomDataSet.TFieldData 3690
IBCustomDataSet.TFieldData record 3690
IBCustomDataSet.TIBBCDField 3599
IBCustomDataSet.TIBBCDField.CheckTypeSize 3600
IBCustomDataSet.TIBBCDField.Create 3600
IBCustomDataSet.TIBBCDField.GetAsCurrency 3601
IBCustomDataSet.TIBBCDField.GetAsString 3601
IBCustomDataSet.TIBBCDField.GetAsVariant 3601
IBCustomDataSet.TIBBCDField.GetDataSize 3601
IBCustomDataSet.TIBBCDField.Size 3602
IBCustomDataSet.TIBCustomDataSet 3602
IBCustomDataSet.TIBCustomDataSet.ActivateConnection 3609
IBCustomDataSet.TIBCustomDataSet.ActivateTransaction 3609
IBCustomDataSet.TIBCustomDataSet.AfterCancel 3634
IBCustomDataSet.TIBCustomDataSet.AfterClose 3634
IBCustomDataSet.TIBCustomDataSet.AfterDatabaseDisconnect 3649
IBCustomDataSet.TIBCustomDataSet.AfterDelete 3634
IBCustomDataSet.TIBCustomDataSet.AfterEdit 3634
IBCustomDataSet.TIBCustomDataSet.AfterInsert 3635
IBCustomDataSet.TIBCustomDataSet.AfterOpen 3635
IBCustomDataSet.TIBCustomDataSet.AfterPost 3635
IBCustomDataSet.TIBCustomDataSet.AfterRefresh 3635
IBCustomDataSet.TIBCustomDataSet.AfterScroll 3635
IBCustomDataSet.TIBCustomDataSet.AfterTransactionEnd 3649
IBCustomDataSet.TIBCustomDataSet.AllocRecordBuffer 3609
IBCustomDataSet.TIBCustomDataSet.ApplyUpdates 3609
IBCustomDataSet.TIBCustomDataSet.AutoCalcFields 3636
IBCustomDataSet.TIBCustomDataSet.BeforeCancel 3636
IBCustomDataSet.TIBCustomDataSet.BeforeClose 3636
IBCustomDataSet.TIBCustomDataSet.BeforeDatabaseDisconnect 3649
IBCustomDataSet.TIBCustomDataSet.BeforeDelete 3636
IBCustomDataSet.TIBCustomDataSet.BeforeEdit 3637
IBCustomDataSet.TIBCustomDataSet.BeforeInsert 3637
IBCustomDataSet.TIBCustomDataSet.BeforeOpen 3637
IBCustomDataSet.TIBCustomDataSet.BeforePost 3637
IBCustomDataSet.TIBCustomDataSet.BeforeRefresh 3637
IBCustomDataSet.TIBCustomDataSet.BeforeScroll 3638
IBCustomDataSet.TIBCustomDataSet.BeforeTransactionEnd 3649
IBCustomDataSet.TIBCustomDataSet.BookmarkValid 3610
IBCustomDataSet.TIBCustomDataSet.BufferChunks 3638
IBCustomDataSet.TIBCustomDataSet.CachedUpdates 3638
IBCustomDataSet.TIBCustomDataSet.CachedUpdateStatus 3610
IBCustomDataSet.TIBCustomDataSet.CancelUpdates 3610
IBCustomDataSet.TIBCustomDataSet.CheckDatasetClosed 3611
IBCustomDataSet.TIBCustomDataSet.CheckDatasetOpen 3611
IBCustomDataSet.TIBCustomDataSet.CheckNotUniDirectional 3611
IBCustomDataSet.TIBCustomDataSet.ClearCalcFields 3611
IBCustomDataSet.TIBCustomDataSet.CompareBookmarks 3611
IBCustomDataSet.TIBCustomDataSet.ConstraintsStored 3612
IBCustomDataSet.TIBCustomDataSet.Create 3606
IBCustomDataSet.TIBCustomDataSet.CreateBlobStream 3612
IBCustomDataSet.TIBCustomDataSet.CreateFields 3612
IBCustomDataSet.TIBCustomDataSet.Current 3613
IBCustomDataSet.TIBCustomDataSet.Database 3639
IBCustomDataSet.TIBCustomDataSet.DatabaseFree 3650

IBCustomDataSet.TIBCustomDataSet.DBHandle 3639
IBCustomDataSet.TIBCustomDataSet.DeactivateTransaction 3613
IBCustomDataSet.TIBCustomDataSet.DeleteSQL 3639
IBCustomDataSet.TIBCustomDataSet.Destroy 3606
IBCustomDataSet.TIBCustomDataSet.Disconnect 3613
IBCustomDataSet.TIBCustomDataSet.DoBeforeDelete 3613
IBCustomDataSet.TIBCustomDataSet.DoBeforeEdit 3613
IBCustomDataSet.TIBCustomDataSet.DoBeforeInsert 3614
IBCustomDataSet.TIBCustomDataSet.DoOnNewRecord 3614
IBCustomDataSet.TIBCustomDataSet.FetchAll 3614
IBCustomDataSet.TIBCustomDataSet.ForcedRefresh 3639
IBCustomDataSet.TIBCustomDataSet.FreeRecordBuffer 3615
IBCustomDataSet.TIBCustomDataSet.GeneratorField 3640
IBCustomDataSet.TIBCustomDataSet.GetActiveBuf 3615
IBCustomDataSet.TIBCustomDataSet.GetBookmarkData 3615
IBCustomDataSet.TIBCustomDataSet.GetBookmarkFlag 3615
IBCustomDataSet.TIBCustomDataSet.GetCanModify 3615
IBCustomDataSet.TIBCustomDataSet.GetCurrentRecord 3616
IBCustomDataSet.TIBCustomDataSet.GetDataSource 3616
IBCustomDataSet.TIBCustomDataSet.GetFieldClass 3616
IBCustomDataSet.TIBCustomDataSet.GetFieldData 3617
IBCustomDataSet.TIBCustomDataSet.GetRecNo 3617
IBCustomDataSet.TIBCustomDataSet.GetRecord 3617
IBCustomDataSet.TIBCustomDataSet.GetRecordCount 3618
IBCustomDataSet.TIBCustomDataSet.GetRecordSize 3618
IBCustomDataSet.TIBCustomDataSet.InitRecord 3618
IBCustomDataSet.TIBCustomDataSet.InsertSQL 3640
IBCustomDataSet.TIBCustomDataSet.InternalAddRecord 3618
IBCustomDataSet.TIBCustomDataSet.InternalBatchInput 3619
IBCustomDataSet.TIBCustomDataSet.InternalBatchOutput 3619
IBCustomDataSet.TIBCustomDataSet.InternalCancel 3619
IBCustomDataSet.TIBCustomDataSet.InternalClose 3619
IBCustomDataSet.TIBCustomDataSet.InternalDelete 3619
IBCustomDataSet.TIBCustomDataSet.InternalExecQuery 3620
IBCustomDataSet.TIBCustomDataSet.InternalFirst 3620
IBCustomDataSet.TIBCustomDataSet.InternalGotoBookmark 3620
IBCustomDataSet.TIBCustomDataSet.InternalHandleException 3620
IBCustomDataSet.TIBCustomDataSet.InternalInitFieldDefs 3621
IBCustomDataSet.TIBCustomDataSet.InternalInitRecord 3621
IBCustomDataSet.TIBCustomDataSet.InternalInsert 3621
IBCustomDataSet.TIBCustomDataSet.InternalLast 3621
IBCustomDataSet.TIBCustomDataSet.InternalOpen 3621
IBCustomDataSet.TIBCustomDataSet.InternalPost 3622
IBCustomDataSet.TIBCustomDataSet.InternalPrepare 3622
IBCustomDataSet.TIBCustomDataSet.InternalPrepared 3640
IBCustomDataSet.TIBCustomDataSet.InternalRefresh 3622
IBCustomDataSet.TIBCustomDataSet.InternalRefreshRow 3622
IBCustomDataSet.TIBCustomDataSet.InternalSetParamsFromCursor 3623
IBCustomDataSet.TIBCustomDataSet.InternalSetToRecord 3623
IBCustomDataSet.TIBCustomDataSet.InternalUnPrepare 3623
IBCustomDataSet.TIBCustomDataSet.IsCursorOpen 3623
IBCustomDataSet.TIBCustomDataSet.IsSequenced 3623
IBCustomDataSet.TIBCustomDataSet.LiveMode 3641
IBCustomDataSet.TIBCustomDataSet.Loaded 3624
IBCustomDataSet.TIBCustomDataSet.Locate 3624
IBCustomDataSet.TIBCustomDataSet.LocateNext 3625
IBCustomDataSet.TIBCustomDataSet.Lookup 3625
IBCustomDataSet.TIBCustomDataSet.ModifySQL 3641
IBCustomDataSet.TIBCustomDataSet.ObjectView 3641
IBCustomDataSet.TIBCustomDataSet.OnCalcFields 3641
IBCustomDataSet.TIBCustomDataSet.OnDeleteError 3642
IBCustomDataSet.TIBCustomDataSet.OnEditError 3642
IBCustomDataSet.TIBCustomDataSet.OnNewRecord 3642
IBCustomDataSet.TIBCustomDataSet.OnPostError 3642
IBCustomDataSet.TIBCustomDataSet.OnUpdateError 3650
IBCustomDataSet.TIBCustomDataSet.OnUpdateRecord 3651
IBCustomDataSet.TIBCustomDataSet.ParamCheck 3642
IBCustomDataSet.TIBCustomDataSet.Params 3643
IBCustomDataSet.TIBCustomDataSet.Plan 3643
IBCustomDataSet.TIBCustomDataSet.Post 3626
IBCustomDataSet.TIBCustomDataSet.PSEndTransaction 3626
IBCustomDataSet.TIBCustomDataSet.PSExecuteStatement 3626
IBCustomDataSet.TIBCustomDataSet.PSGetQuoteChar 3627
IBCustomDataSet.TIBCustomDataSet.PSGetTableName 3627

IBCustomDataSet.TIBCustomDataSet.PSGetUpdateException 3627
IBCustomDataSet.TIBCustomDataSet.PSInTransaction 3627
IBCustomDataSet.TIBCustomDataSet.PSIsSQLBased 3628
IBCustomDataSet.TIBCustomDataSet.PSIsSQLSupported 3628
IBCustomDataSet.TIBCustomDataSet.PSReset 3628
IBCustomDataSet.TIBCustomDataSet.PSStartTransaction 3628
IBCustomDataSet.TIBCustomDataSet.PSUpdateRecord 3628
IBCustomDataSet.TIBCustomDataSet.QDelete 3643
IBCustomDataSet.TIBCustomDataSet.QInsert 3644
IBCustomDataSet.TIBCustomDataSet.QModify 3644
IBCustomDataSet.TIBCustomDataSet.QRefresh 3644
IBCustomDataSet.TIBCustomDataSet.QSelect 3644
IBCustomDataSet.TIBCustomDataSet.RecordModified 3629
IBCustomDataSet.TIBCustomDataSet.RefreshSQL 3645
IBCustomDataSet.TIBCustomDataSet.ReQuery 3629
IBCustomDataSet.TIBCustomDataSet.RevertRecord 3629
IBCustomDataSet.TIBCustomDataSet.RowsAffected 3645
IBCustomDataSet.TIBCustomDataSet.SelectSQL 3645
IBCustomDataSet.TIBCustomDataSet.SelectStmtHandle 3645
IBCustomDataSet.TIBCustomDataSet.SetActive 3629
IBCustomDataSet.TIBCustomDataSet.SetBookmarkData 3630
IBCustomDataSet.TIBCustomDataSet.SetBookmarkFlag 3630
IBCustomDataSet.TIBCustomDataSet.SetCachedUpdates 3630
IBCustomDataSet.TIBCustomDataSet.SetDataSource 3630
IBCustomDataSet.TIBCustomDataSet.SetFieldData 3631
IBCustomDataSet.TIBCustomDataSet.SetRecNo 3631
IBCustomDataSet.TIBCustomDataSet.SQLParams 3646
IBCustomDataSet.TIBCustomDataSet.SQLType 3631
IBCustomDataSet.TIBCustomDataSet.StatementType 3646
IBCustomDataSet.TIBCustomDataSet.Transaction 3646
IBCustomDataSet.TIBCustomDataSet.TransactionFree 3651
IBCustomDataSet.TIBCustomDataSet.TRHandle 3646
IBCustomDataSet.TIBCustomDataSet.Undelete 3632
IBCustomDataSet.TIBCustomDataSet.UniDirectional 3647
IBCustomDataSet.TIBCustomDataSet.UpdateMode 3647
IBCustomDataSet.TIBCustomDataSet.UpdateObject 3647
IBCustomDataSet.TIBCustomDataSet.UpdateRecordTypes 3648
IBCustomDataSet.TIBCustomDataSet.UpdatesPending 3648
IBCustomDataSet.TIBCustomDataSet.UpdateStatus 3632
IBCustomDataSet.TIBDataLink 3651
IBCustomDataSet.TIBDataLink.ActiveChanged 3653
IBCustomDataSet.TIBDataLink.CheckBrowseMode 3653
IBCustomDataSet.TIBDataLink.Create 3652
IBCustomDataSet.TIBDataLink.Destroy 3652
IBCustomDataSet.TIBDataLink.GetDetailDataSet 3653
IBCustomDataSet.TIBDataLink.RecordChanged 3654
IBCustomDataSet.TIBDataSet 3654
IBCustomDataSet.TIBDataSet.Active 3662
IBCustomDataSet.TIBDataSet.AfterCancel 3662
IBCustomDataSet.TIBDataSet.AfterClose 3662
IBCustomDataSet.TIBDataSet.AfterDatabaseDisconnect 3673
IBCustomDataSet.TIBDataSet.AfterDelete 3663
IBCustomDataSet.TIBDataSet.AfterEdit 3663
IBCustomDataSet.TIBDataSet.AfterInsert 3663
IBCustomDataSet.TIBDataSet.AfterOpen 3663
IBCustomDataSet.TIBDataSet.AfterPost 3663
IBCustomDataSet.TIBDataSet.AfterScroll 3664
IBCustomDataSet.TIBDataSet.AfterTransactionEnd 3673
IBCustomDataSet.TIBDataSet.AutoCalcFields 3664
IBCustomDataSet.TIBDataSet.BatchInput 3659
IBCustomDataSet.TIBDataSet.BatchOutput 3659
IBCustomDataSet.TIBDataSet.BeforeCancel 3664
IBCustomDataSet.TIBDataSet.BeforeClose 3664
IBCustomDataSet.TIBDataSet.BeforeDatabaseDisconnect 3673
IBCustomDataSet.TIBDataSet.BeforeDelete 3665
IBCustomDataSet.TIBDataSet.BeforeEdit 3665
IBCustomDataSet.TIBDataSet.BeforeInsert 3665
IBCustomDataSet.TIBDataSet.BeforeOpen 3665
IBCustomDataSet.TIBDataSet.BeforePost 3665
IBCustomDataSet.TIBDataSet.BeforeScroll 3666
IBCustomDataSet.TIBDataSet.BeforeTransactionEnd 3674
IBCustomDataSet.TIBDataSet.BufferChunks 3666
IBCustomDataSet.TIBDataSet.CachedUpdates 3666
IBCustomDataSet.TIBDataSet.DatabaseFree 3674
IBCustomDataSet.TIBDataSet.DataSource 3667
IBCustomDataSet.TIBDataSet.DeleteSQL 3667
IBCustomDataSet.TIBDataSet.ExecSQL 3659

IBCustomDataSet.TIBDataSet.Filtered 3667
IBCustomDataSet.TIBDataSet.GeneratorField 3667
IBCustomDataSet.TIBDataSet.InsertSQL 3668
IBCustomDataSet.TIBDataSet.InternalOpen 3659
IBCustomDataSet.TIBDataSet.LiveMode 3668
IBCustomDataSet.TIBDataSet.ModifySQL 3668
IBCustomDataSet.TIBDataSet.OnCalcFields 3669
IBCustomDataSet.TIBDataSet.OnDeleteError 3669
IBCustomDataSet.TIBDataSet.OnEditError 3669
IBCustomDataSet.TIBDataSet.OnFilterRecord 3669
IBCustomDataSet.TIBDataSet.OnNewRecord 3669
IBCustomDataSet.TIBDataSet.OnPostError 3670
IBCustomDataSet.TIBDataSet.ParamByName 3660
IBCustomDataSet.TIBDataSet.ParamCheck 3670
IBCustomDataSet.TIBDataSet.Params 3670
IBCustomDataSet.TIBDataSet.Prepare 3660
IBCustomDataSet.TIBDataSet.Prepared 3671
IBCustomDataSet.TIBDataSet.PSSetCommandText 3660
IBCustomDataSet.TIBDataSet.RefreshSQL 3671
IBCustomDataSet.TIBDataSet.SelectSQL 3671
IBCustomDataSet.TIBDataSet.SelectStmtHandle 3671
IBCustomDataSet.TIBDataSet.SetFiltered 3660
IBCustomDataSet.TIBDataSet.StatementType 3672
IBCustomDataSet.TIBDataSet.TransactionFree 3674
IBCustomDataSet.TIBDataSet.UniDirectional 3672
IBCustomDataSet.TIBDataSet.UnPrepare 3661
IBCustomDataSet.TIBDataSet.UpdateObject 3672
IBCustomDataSet.TIBDataSetUpdateObject 3674
IBCustomDataSet.TIBDataSetUpdateObject.Apply 3676
IBCustomDataSet.TIBDataSetUpdateObject.Create 3675
IBCustomDataSet.TIBDataSetUpdateObject.DataSet 3677
IBCustomDataSet.TIBDataSetUpdateObject.Destroy 3676
IBCustomDataSet.TIBDataSetUpdateObject.GetDataSet 3676
IBCustomDataSet.TIBDataSetUpdateObject.GetSQL 3677
IBCustomDataSet.TIBDataSetUpdateObject.RefreshSQL 3677
IBCustomDataSet.TIBDataSetUpdateObject.SetDataSet 3677
IBCustomDataSet.TIBDBKey 3690
IBCustomDataSet.TIBDBKey record 3690
IBCustomDataSet.TIBDSBlobStream 3678
IBCustomDataSet.TIBDSBlobStream.Create 3679
IBCustomDataSet.TIBDSBlobStream.Destroy 3679
IBCustomDataSet.TIBDSBlobStream.FBlobStream 3679
IBCustomDataSet.TIBDSBlobStream.FField 3680
IBCustomDataSet.TIBDSBlobStream.FModified 3680
IBCustomDataSet.TIBDSBlobStream.Read 3680
IBCustomDataSet.TIBDSBlobStream.Seek 3681
IBCustomDataSet.TIBDSBlobStream.SetSize 3681
IBCustomDataSet.TIBDSBlobStream.Write 3681
IBCustomDataSet.TIBGeneratorApplyEvent 3691
IBCustomDataSet.TIBGeneratorApplyEvent enumeration 3691
IBCustomDataSet.TIBGeneratorField 3682
IBCustomDataSet.TIBGeneratorField.Apply 3683
IBCustomDataSet.TIBGeneratorField.ApplyEvent 3685
IBCustomDataSet.TIBGeneratorField.Assign 3684
IBCustomDataSet.TIBGeneratorField.Create 3683
IBCustomDataSet.TIBGeneratorField.Field 3685
IBCustomDataSet.TIBGeneratorField.Generator 3685
IBCustomDataSet.TIBGeneratorField.IncrementBy 3686
IBCustomDataSet.TIBGeneratorField.ValueName 3684
IBCustomDataSet.TIBStringField 3686
IBCustomDataSet.TIBStringField.CheckTypeSize 3687
IBCustomDataSet.TIBStringField.Create 3687
IBCustomDataSet.TIBStringField.GetAsString 3688
IBCustomDataSet.TIBStringField.GetAsVariant 3688
IBCustomDataSet.TIBStringField.GetValue 3688
IBCustomDataSet.TIBStringField.SetAsString 3688
IBCustomDataSet.TIBUpdateAction 3691
IBCustomDataSet.TIBUpdateAction enumeration 3691
IBCustomDataSet.TIBUpdateErrorEvent 3696
IBCustomDataSet.TIBUpdateErrorEvent type 3696
IBCustomDataSet.TIBUpdateRecordEvent 3696
IBCustomDataSet.TIBUpdateRecordEvent type 3696
IBCustomDataSet.TIBUpdateRecordTypes 3697
IBCustomDataSet.TIBUpdateRecordTypes type 3697
IBCustomDataSet.TLiveMode 3692
IBCustomDataSet.TLiveMode enumeration 3692
IBCustomDataSet.TLiveModes 3697
IBCustomDataSet.TLiveModes type 3697

IBCustomDataSet.TRecordData 3693
IBCustomDataSet.TRecordData record 3693
IBCustomDataSet.UniCache 3701
IBCustomDataSet.UniCache constant 3701
IBDatabase 3703
IBDatabase namespace 3703
Classes 3705
Constants 3768
Functions 3764
Interfaces 3761
Structs, Records, Enums 3765
Types 3767
IBDatabase.DPBConstantNames 3768
IBDatabase.DPBConstantNames constant 3768
IBDatabase.DPBPrefix 3769
IBDatabase.DPBPrefix constant 3769
IBDatabase.GenerateDPB 3764
IBDatabase.GenerateDPB function 3764
IBDatabase.GenerateTPB 3764
IBDatabase.GenerateTPB function 3764
IBDatabase.IIBEventNotifier 3761
IBDatabase.IIBEventNotifier.GetAutoRegister 3762
IBDatabase.IIBEventNotifier.RegisterEvents 3762
IBDatabase.IIBEventNotifier.UnRegisterEvents 3762
IBDatabase.TAutoStopAction 3765
IBDatabase.TAutoStopAction enumeration 3765
IBDatabase.TIBBase 3705
IBDatabase.TIBBase.AfterDatabaseDisconnect 3715
IBDatabase.TIBBase.AfterTransactionEnd 3715
IBDatabase.TIBBase.BeforeDatabaseDisconnect 3715
IBDatabase.TIBBase.BeforeTransactionEnd 3715
IBDatabase.TIBBase.CheckDatabase 3710
IBDatabase.TIBBase.CheckTransaction 3710
IBDatabase.TIBBase.Create 3706
IBDatabase.TIBBase.Database 3713
IBDatabase.TIBBase.DBHandle 3713
IBDatabase.TIBBase.Destroy 3707
IBDatabase.TIBBase.DoAfterDatabaseDisconnect 3711
IBDatabase.TIBBase.DoAfterTransactionEnd 3711
IBDatabase.TIBBase.DoBeforeDatabaseDisconnect 3711
IBDatabase.TIBBase.DoBeforeTransactionEnd 3711
IBDatabase.TIBBase.DoDatabaseFree 3711
IBDatabase.TIBBase.DoTransactionFree 3712
IBDatabase.TIBBase.FAfterDatabaseDisconnect 3707
IBDatabase.TIBBase.FAfterTransactionEnd 3707
IBDatabase.TIBBase.FBeforeDatabaseDisconnect 3708
IBDatabase.TIBBase.FBeforeTransactionEnd 3708
IBDatabase.TIBBase.FDatabase 3708
IBDatabase.TIBBase.FIndexInDatabase 3708
IBDatabase.TIBBase.FIndexInTransaction 3709
IBDatabase.TIBBase.FOnDatabaseFree 3709
IBDatabase.TIBBase.FOnTransactionFree 3709
IBDatabase.TIBBase.FOwner 3709
IBDatabase.TIBBase.FTransaction 3709
IBDatabase.TIBBase.GetDBHandle 3712
IBDatabase.TIBBase.GetTRHandle 3712
IBDatabase.TIBBase.OnDatabaseFree 3716
IBDatabase.TIBBase.OnTransactionFree 3716
IBDatabase.TIBBase.Owner 3714
IBDatabase.TIBBase.SetDatabase 3712
IBDatabase.TIBBase.SetTransaction 3713
IBDatabase.TIBBase.Transaction 3714
IBDatabase.TIBBase.TRHandle 3714
IBDatabase.TIBDataBase 3716
IBDatabase.TIBDataBase.AddEventNotifier 3720
IBDatabase.TIBDataBase.AddTransaction 3720
IBDatabase.TIBDataBase.AfterConnect 3738
IBDatabase.TIBDataBase.AfterDisconnect 3739
IBDatabase.TIBDataBase.AllowStreamedConnected 3729
IBDatabase.TIBDataBase.ApplyUpdates 3720
IBDatabase.TIBDataBase.BeforeConnect 3739
IBDatabase.TIBDataBase.BeforeDisconnect 3739
IBDatabase.TIBDataBase.Call 3721
IBDatabase.TIBDataBase.CheckActive 3721
IBDatabase.TIBDataBase.CheckDatabaseName 3721
IBDatabase.TIBDataBase.CheckInactive 3721
IBDatabase.TIBDataBase.CloseDataSets 3722
IBDatabase.TIBDataBase.Connected 3729

IBDatabase.TIBDataBase.Create 3718
IBDatabase.TIBDataBase.CreateDatabase 3722
IBDatabase.TIBDataBase.DatabaseName 3730
IBDatabase.TIBDataBase.DBParamByDPB 3730
IBDatabase.TIBDataBase.DBSQLDialect 3730
IBDatabase.TIBDataBase.DefaultTransaction 3731
IBDatabase.TIBDataBase.Destroy 3718
IBDatabase.TIBDataBase.DoConnect 3722
IBDatabase.TIBDataBase.DoDisconnect 3723
IBDatabase.TIBDataBase.DropDatabase 3723
IBDatabase.TIBDataBase.FindDefaultTransaction 3723
IBDatabase.TIBDataBase.FindTransaction 3724
IBDatabase.TIBDataBase.FlushSchema 3724
IBDatabase.TIBDataBase.ForceClose 3724
IBDatabase.TIBDataBase.GetConnected 3724
IBDatabase.TIBDataBase.GetFieldNames 3725
IBDatabase.TIBDataBase.GetTableNames 3725
IBDatabase.TIBDataBase.Handle 3731
IBDatabase.TIBDataBase.HandleIsShared 3731
IBDatabase.TIBDataBase.Has_COMPUTED_BLR 3725
IBDatabase.TIBDataBase.Has_DEFAULT_VALUE 3726
IBDatabase.TIBDataBase.IdleTimer 3731
IBDatabase.TIBDataBase.IndexOfDBConst 3726
IBDatabase.TIBDataBase.InternalTransaction 3732
IBDatabase.TIBDataBase.IsReadOnly 3732
IBDatabase.TIBDataBase.Loaded 3726
IBDatabase.TIBDataBase.LoginPrompt 3732
IBDatabase.TIBDataBase.Notification 3726
IBDatabase.TIBDataBase.OnDialectDowngradeWarning 3740
IBDatabase.TIBDataBase.OnIdleTimer 3740
IBDatabase.TIBDataBase.OnLogin 3741
IBDatabase.TIBDataBase.Params 3736
IBDatabase.TIBDataBase.RemoveEventNotifier 3727
IBDatabase.TIBDataBase.RemoveTransaction 3727
IBDatabase.TIBDataBase.RemoveTransactions 3728
IBDatabase.TIBDataBase.SetHandle 3728
IBDatabase.TIBDataBase.SQLDialect 3736
IBDatabase.TIBDataBase.SQLObjectCount 3737
IBDatabase.TIBDataBase.SQLObjects 3737
IBDatabase.TIBDataBase.TestConnected 3728
IBDatabase.TIBDataBase.TraceFlags 3737
IBDatabase.TIBDataBase.TransactionCount 3738
IBDatabase.TIBDataBase.Transactions 3738
IBDatabase.TIBDatabaseLoginEvent 3767
IBDatabase.TIBDatabaseLoginEvent type 3767
IBDatabase.TIBFileName 3767
IBDatabase.TIBFileName type 3767
IBDatabase.TIBSchema 3741
IBDatabase.TIBSchema.FreeNodes 3742
IBDatabase.TIBSchema.Has_COMPUTED_BLR 3742
IBDatabase.TIBSchema.Has_DEFAULT_VALUE 3742
IBDatabase.TIBTimer 3742
IBDatabase.TIBTimer.Create 3743
IBDatabase.TIBTimer.Destroy 3744
IBDatabase.TIBTimer.Enabled 3745
IBDatabase.TIBTimer.Interval 3746
IBDatabase.TIBTimer.OnTimer 3746
IBDatabase.TIBTimer.Timer 3745
IBDatabase.TIBTransaction 3746
IBDatabase.TIBTransaction.Active 3755
IBDatabase.TIBTransaction.AddDatabase 3749
IBDatabase.TIBTransaction.AutoStopAction 3755
IBDatabase.TIBTransaction.Call 3749
IBDatabase.TIBTransaction.CheckAutoStop 3750
IBDatabase.TIBTransaction.CheckDatabasesInList 3750
IBDatabase.TIBTransaction.CheckInTransaction 3750
IBDatabase.TIBTransaction.CheckNotInTransaction 3750
IBDatabase.TIBTransaction.Commit 3751
IBDatabase.TIBTransaction.CommitRetaining 3751
IBDatabase.TIBTransaction.Create 3748
IBDatabase.TIBTransaction.DatabaseCount 3756
IBDatabase.TIBTransaction.Databases 3756
IBDatabase.TIBTransaction.DefaultAction 3756
IBDatabase.TIBTransaction.DefaultDatabase 3757
IBDatabase.TIBTransaction.Destroy 3748
IBDatabase.TIBTransaction.FindDatabase 3751
IBDatabase.TIBTransaction.FindDefaultDatabase 3752
IBDatabase.TIBTransaction.Handle 3757

IBDatabase.TIBTransaction.HandleIsShared 3757
IBDatabase.TIBTransaction.IdleTimer 3757
IBDatabase.TIBTransaction.InTransaction 3758
IBDatabase.TIBTransaction.Loaded 3752
IBDatabase.TIBTransaction.Notification 3752
IBDatabase.TIBTransaction.OnIdleTimer 3759
IBDatabase.TIBTransaction.Params 3758
IBDatabase.TIBTransaction.RemoveDatabase 3753
IBDatabase.TIBTransaction.RemoveDatabases 3753
IBDatabase.TIBTransaction.Rollback 3753
IBDatabase.TIBTransaction.RollbackRetaining 3754
IBDatabase.TIBTransaction.SetHandle 3754
IBDatabase.TIBTransaction.SQLObjectCount 3758
IBDatabase.TIBTransaction.SQLObjects 3758
IBDatabase.TIBTransaction.StartTransaction 3754
IBDatabase.TIBTransaction.TPB 3759
IBDatabase.TIBTransaction.TPBLength 3759
IBDatabase.TPBConstantNames 3769
IBDatabase.TPBConstantNames constant 3769
IBDatabase.TPBPrefix 3769
IBDatabase.TPBPrefix constant 3769
IBDatabase.TTransactionAction 3766
IBDatabase.TTransactionAction enumeration 3766
IBDatabaseInfo 3771
IBDatabaseInfo namespace 3771
Classes 3772
IBDatabaseInfo.TIBDatabaseInfo 3772
IBDatabaseInfo.TIBDatabaseInfo.Allocation 3787
IBDatabaseInfo.TIBDatabaseInfo.BackoutCount 3787
IBDatabaseInfo.TIBDatabaseInfo.BaseLevel 3788
IBDatabaseInfo.TIBDatabaseInfo.Call 3778
IBDatabaseInfo.TIBDatabaseInfo.Create 3774
IBDatabaseInfo.TIBDatabaseInfo.CurrentMemory 3788
IBDatabaseInfo.TIBDatabaseInfo.Database 3788
IBDatabaseInfo.TIBDatabaseInfo.DBFileName 3788
IBDatabaseInfo.TIBDatabaseInfo.DBImplementationClass 3789
IBDatabaseInfo.TIBDatabaseInfo.DBImplementationNo 3789
IBDatabaseInfo.TIBDatabaseInfo.DBSiteName 3789
IBDatabaseInfo.TIBDatabaseInfo.DBSQLDialect 3789
IBDatabaseInfo.TIBDatabaseInfo.DeleteCount 3790
IBDatabaseInfo.TIBDatabaseInfo.Destroy 3774
IBDatabaseInfo.TIBDatabaseInfo.ExpungeCount 3790
IBDatabaseInfo.TIBDatabaseInfo.FBackoutCount 3775
IBDatabaseInfo.TIBDatabaseInfo.FDatabase 3775
IBDatabaseInfo.TIBDatabaseInfo.FDeleteCount 3776
IBDatabaseInfo.TIBDatabaseInfo.Fetches 3790
IBDatabaseInfo.TIBDatabaseInfo.FExpungeCount 3776
IBDatabaseInfo.TIBDatabaseInfo.FIBLoaded 3776
IBDatabaseInfo.TIBDatabaseInfo.FInsertCount 3776
IBDatabaseInfo.TIBDatabaseInfo.ForcedWrites 3790
IBDatabaseInfo.TIBDatabaseInfo.FPurgeCount 3776
IBDatabaseInfo.TIBDatabaseInfo.FReadIdxCount 3777
IBDatabaseInfo.TIBDatabaseInfo.FReadSeqCount 3777
IBDatabaseInfo.TIBDatabaseInfo.FUpdateCount 3777
IBDatabaseInfo.TIBDatabaseInfo.FUserNames 3777
IBDatabaseInfo.TIBDatabaseInfo.GetAllocation 3779
IBDatabaseInfo.TIBDatabaseInfo.GetBackoutCount 3779
IBDatabaseInfo.TIBDatabaseInfo.GetBaseLevel 3779
IBDatabaseInfo.TIBDatabaseInfo.GetCurrentMemory 3779
IBDatabaseInfo.TIBDatabaseInfo.GetDBFileName 3779
IBDatabaseInfo.TIBDatabaseInfo.GetDBImplementationClass 3780
IBDatabaseInfo.TIBDatabaseInfo.GetDBImplementationNo 3780
IBDatabaseInfo.TIBDatabaseInfo.GetDBSiteName 3780
IBDatabaseInfo.TIBDatabaseInfo.GetDBSQLDialect 3780
IBDatabaseInfo.TIBDatabaseInfo.GetDeleteCount 3781
IBDatabaseInfo.TIBDatabaseInfo.GetExpungeCount 3781
IBDatabaseInfo.TIBDatabaseInfo.GetFetches 3781
IBDatabaseInfo.TIBDatabaseInfo.GetForcedWrites 3781
IBDatabaseInfo.TIBDatabaseInfo.GetInsertCount 3781
IBDatabaseInfo.TIBDatabaseInfo.GetLongDatabaseInfo 3782
IBDatabaseInfo.TIBDatabaseInfo.GetMarks 3782
IBDatabaseInfo.TIBDatabaseInfo.GetMaxMemory 3782
IBDatabaseInfo.TIBDatabaseInfo.GetNoReserve 3782
IBDatabaseInfo.TIBDatabaseInfo.GetNumBuffers 3783
IBDatabaseInfo.TIBDatabaseInfo.GetODSMajorVersion 3783
IBDatabaseInfo.TIBDatabaseInfo.GetODSMinorVersion 3783
IBDatabaseInfo.TIBDatabaseInfo.GetOperationCounts 3783
IBDatabaseInfo.TIBDatabaseInfo.GetPageSize 3784

IBDatabaseInfo.TIBDatabaseInfo.GetPurgeCount 3784
IBDatabaseInfo.TIBDatabaseInfo.GetReadIdxCount 3784
IBDatabaseInfo.TIBDatabaseInfo.GetReadOnly 3784
IBDatabaseInfo.TIBDatabaseInfo.GetReads 3784
IBDatabaseInfo.TIBDatabaseInfo.GetReadSeqCount 3785
IBDatabaseInfo.TIBDatabaseInfo.GetStringDatabaseInfo 3785
IBDatabaseInfo.TIBDatabaseInfo.GetSweepInterval 3785
IBDatabaseInfo.TIBDatabaseInfo.GetUpdateCount 3785
IBDatabaseInfo.TIBDatabaseInfo.GetUserNames 3786
IBDatabaseInfo.TIBDatabaseInfo.GetVersion 3786
IBDatabaseInfo.TIBDatabaseInfo.GetWrites 3786
IBDatabaseInfo.TIBDatabaseInfo.InsertCount 3791
IBDatabaseInfo.TIBDatabaseInfo.Marks 3791
IBDatabaseInfo.TIBDatabaseInfo.MaxMemory 3791
IBDatabaseInfo.TIBDatabaseInfo.NoReserve 3791
IBDatabaseInfo.TIBDatabaseInfo.NumBuffers 3792
IBDatabaseInfo.TIBDatabaseInfo.ODSMajorVersion 3792
IBDatabaseInfo.TIBDatabaseInfo.ODSMinorVersion 3792
IBDatabaseInfo.TIBDatabaseInfo.PageSize 3792
IBDatabaseInfo.TIBDatabaseInfo.PurgeCount 3793
IBDatabaseInfo.TIBDatabaseInfo.ReadIdxCount 3793
IBDatabaseInfo.TIBDatabaseInfo.ReadOnly 3793
IBDatabaseInfo.TIBDatabaseInfo.Reads 3793
IBDatabaseInfo.TIBDatabaseInfo.ReadSeqCount 3794
IBDatabaseInfo.TIBDatabaseInfo.SweepInterval 3794
IBDatabaseInfo.TIBDatabaseInfo.UpdateCount 3794
IBDatabaseInfo.TIBDatabaseInfo.UserNames 3795
IBDatabaseInfo.TIBDatabaseInfo.Version 3795
IBDatabaseInfo.TIBDatabaseInfo.Writes 3795
IBEvents 3797
IBEvents namespace 3797
    Classes 3798
    Types 3806
IBEvents.TErrorEvent 3806
IBEvents.TErrorEvent type 3806
IBEvents.TEventAlert 3806
IBEvents.TEventAlert type 3806
IBEvents.TIBEvents 3798
IBEvents.TIBEvents.AutoRegister 3803
IBEvents.TIBEvents.Create 3799
IBEvents.TIBEvents.Database 3803
IBEvents.TIBEvents.Destroy 3800
IBEvents.TIBEvents.EventChange 3800
IBEvents.TIBEvents.Events 3804
IBEvents.TIBEvents.GetAutoRegister 3800
IBEvents.TIBEvents.GetNativeHandle 3801
IBEvents.TIBEvents.NativeHandle 3804
IBEvents.TIBEvents.Notification 3801
IBEvents.TIBEvents.OnError 3805
IBEvents.TIBEvents.OnEventAlert 3805
IBEvents.TIBEvents.Registered 3804
IBEvents.TIBEvents.RegisterEvents 3801
IBEvents.TIBEvents.SetAutoRegister 3802
IBEvents.TIBEvents.ThreadEnded 3802
IBEvents.TIBEvents.UnRegisterEvents 3802
IBEvents.TIBEvents.ValidateDatabase 3803
IBExtract 3809
IBExtract namespace 3809
    Classes 3811
    Constants 3829
    Structs, Records, Enums 3823
    Types 3827
IBExtract.ColumnTypes 3830
IBExtract.ColumnTypes constant 3830
IBExtract.FILE_cache 3830
IBExtract.FILE_cache constant 3830
IBExtract.FILE_conditional 3830
IBExtract.FILE_conditional constant 3830
IBExtract.FILE_inactive 3831
IBExtract.FILE_inactive constant 3831
IBExtract.FILE_manual 3831
IBExtract.FILE_manual constant 3831
IBExtract.FILE_shadow 3831
IBExtract.FILE_shadow constant 3831
IBExtract.IntegralSubtypes 3832
IBExtract.IntegralSubtypes constant 3832
IBExtract.LOG_default 3832
IBExtract.LOG_default constant 3832

IBExtract.LOG_overflow 3832
IBExtract.LOG_overflow constant 3832
IBExtract.LOG_raw 3833
IBExtract.LOG_raw constant 3833
IBExtract.LOG_serial 3833
IBExtract.LOG_serial constant 3833
IBExtract.MAX_INTSUBTYPES 3833
IBExtract.MAX_INTSUBTYPES constant 3833
IBExtract.MAXSUBTYPES 3834
IBExtract.MAXSUBTYPES constant 3834
IBExtract.obj_computed 3834
IBExtract.obj_computed constant 3834
IBExtract.obj_count 3834
IBExtract.obj_count constant 3834
IBExtract.obj_exception 3835
IBExtract.obj_exception constant 3835
IBExtract.obj_expression_index 3835
IBExtract.obj_expression_index constant 3835
IBExtract.obj_field 3835
IBExtract.obj_field constant 3835
IBExtract.obj_index 3836
IBExtract.obj_index constant 3836
IBExtract.obj_procedure 3836
IBExtract.obj_procedure constant 3836
IBExtract.obj_relation 3836
IBExtract.obj_relation constant 3836
IBExtract.obj_sql_role 3837
IBExtract.obj_sql_role constant 3837
IBExtract.obj_trigger 3837
IBExtract.obj_trigger constant 3837
IBExtract.obj_user 3837
IBExtract.obj_user constant 3837
IBExtract.obj_user_group 3838
IBExtract.obj_user_group constant 3838
IBExtract.obj_validation 3838
IBExtract.obj_validation constant 3838
IBExtract.obj_view 3838
IBExtract.obj_view constant 3838
IBExtract.ODS_VERSION10 3839
IBExtract.ODS_VERSION10 constant 3839
IBExtract.ODS_VERSION6 3839
IBExtract.ODS_VERSION6 constant 3839
IBExtract.ODS_VERSION7 3839
IBExtract.ODS_VERSION7 constant 3839
IBExtract.ODS_VERSION8 3840
IBExtract.ODS_VERSION8 constant 3840
IBExtract.ODS_VERSION9 3840
IBExtract.ODS_VERSION9 constant 3840
IBExtract.priv_DELETE 3840
IBExtract.priv_DELETE constant 3840
IBExtract.priv_EXECUTE 3841
IBExtract.priv_EXECUTE constant 3841
IBExtract.priv_INSERT 3841
IBExtract.priv_INSERT constant 3841
IBExtract.priv_REFERENCES 3841
IBExtract.priv_REFERENCES constant 3841
IBExtract.priv_SELECT 3842
IBExtract.priv_SELECT constant 3842
IBExtract.priv_UNKNOWN 3842
IBExtract.priv_UNKNOWN constant 3842
IBExtract.priv_UPDATE 3842
IBExtract.priv_UPDATE constant 3842
IBExtract.PrivTypes 3843
IBExtract.PrivTypes constant 3843
IBExtract.SubTypes 3843
IBExtract.SubTypes constant 3843
IBExtract.TExtractObjectTypes 3823
IBExtract.TExtractObjectTypes enumeration 3823
IBExtract.TExtractType 3824
IBExtract.TExtractType enumeration 3824
IBExtract.TExtractTypes 3827
IBExtract.TExtractTypes type 3827
IBExtract.TIBExtract 3811
IBExtract.TIBExtract.Create 3812
IBExtract.TIBExtract.Database 3821
IBExtract.TIBExtract.DatabaseInfo 3821
IBExtract.TIBExtract.Destroy 3813
IBExtract.TIBExtract.ExtractDDL 3814

IBExtract.TIBExtract.ExtractListTable 3814
IBExtract.TIBExtract.ExtractListView 3814
IBExtract.TIBExtract.ExtractObject 3814
IBExtract.TIBExtract.GetArrayField 3816
IBExtract.TIBExtract.GetCharacterSets 3816
IBExtract.TIBExtract.GetFieldType 3816
IBExtract.TIBExtract.Items 3821
IBExtract.TIBExtract.ListAllTables 3816
IBExtract.TIBExtract.ListCheck 3817
IBExtract.TIBExtract.ListCreateDb 3817
IBExtract.TIBExtract.ListData 3817
IBExtract.TIBExtract.ListDomains 3817
IBExtract.TIBExtract.ListException 3818
IBExtract.TIBExtract.ListFilters 3818
IBExtract.TIBExtract.ListForeign 3818
IBExtract.TIBExtract.ListFunctions 3818
IBExtract.TIBExtract.ListGenerators 3818
IBExtract.TIBExtract.ListGrants 3819
IBExtract.TIBExtract.ListIndex 3819
IBExtract.TIBExtract.ListProcs 3819
IBExtract.TIBExtract.ListRoles 3819
IBExtract.TIBExtract.ListTriggers 3820
IBExtract.TIBExtract.ListViews 3820
IBExtract.TIBExtract.Notification 3820
IBExtract.TIBExtract.PrintSet 3820
IBExtract.TIBExtract.ShowSystem 3822
IBExtract.TIBExtract.Transaction 3822
IBExtract.TPrivTypes 3825
IBExtract.TPrivTypes record 3825
IBExtract.TriggerTypes 3843
IBExtract.TriggerTypes constant 3843
IBExtract.TSQLType 3825
IBExtract.TSQLType record 3825
IBExtract.TSQLTypes 3828
IBExtract.TSQLTypes type 3828
IBInstall 3845
IBInstall namespace 3845
Classes 3847
Structs, Records, Enums 3868
Types 3873
IBInstall.EIBInstall 3847
IBInstall.EIBInstall.Create 3848
IBInstall.EIBInstall.InstallerError 3848
IBInstall.EIBInstall.InstallError 3849
IBInstall.EIBInstallerError 3849
IBInstall.EIBInstallError 3850
IBInstall.TCmdOption 3868
IBInstall.TCmdOption enumeration 3868
IBInstall.TCmdOptions 3873
IBInstall.TCmdOptions type 3873
IBInstall.TConnectivityOption 3869
IBInstall.TConnectivityOption enumeration 3869
IBInstall.TConnectivityOptions 3873
IBInstall.TConnectivityOptions type 3873
IBInstall.TErrorResult 3869
IBInstall.TErrorResult enumeration 3869
IBInstall.TExamplesOption 3870
IBInstall.TExamplesOption enumeration 3870
IBInstall.TExamplesOptions 3874
IBInstall.TExamplesOptions type 3874
IBInstall.TIBInstall 3850
IBInstall.TIBInstall.Create 3851
IBInstall.TIBInstall.DestinationDirectory 3855
IBInstall.TIBInstall.Destroy 3852
IBInstall.TIBInstall.GetOptionDescription 3852
IBInstall.TIBInstall.GetOptionName 3853
IBInstall.TIBInstall.GetOptionSpaceRequired 3854
IBInstall.TIBInstall.InstallCheck 3854
IBInstall.TIBInstall.InstallExecute 3854
IBInstall.TIBInstall.InstallOptions 3855
IBInstall.TIBInstall.SourceDirectory 3856
IBInstall.TIBInstall.SuggestedDestination 3856
IBInstall.TIBInstall.UnInstallFile 3856
IBInstall.TIBInstallerError 3870
IBInstall.TIBInstallerError enumeration 3870
IBInstall.TIBSetup 3857
IBInstall.TIBSetup.Call 3858
IBInstall.TIBSetup.Create 3858

IBInstall.TIBSetup.ErrorContext 3859
IBInstall.TIBSetup.ErrorInternal 3858
IBInstall.TIBSetup.GetInstallMessage 3859
IBInstall.TIBSetup.IBInstallError 3859
IBInstall.TIBSetup.MsgFilePath 3860
IBInstall.TIBSetup.OnError 3861
IBInstall.TIBSetup.OnStatusChange 3862
IBInstall.TIBSetup.OnWarning 3862
IBInstall.TIBSetup.Progress 3860
IBInstall.TIBSetup.RebootToComplete 3860
IBInstall.TIBSetup.StatusContext 3861
IBInstall.TIBSetup.StatusInternal 3859
IBInstall.TIBSetupOnError 3874
IBInstall.TIBSetupOnError type 3874
IBInstall.TIBSetupOnStatus 3874
IBInstall.TIBSetupOnStatus type 3874
IBInstall.TIBSetupOnWarning 3875
IBInstall.TIBSetupOnWarning type 3875
IBInstall.TIBUnInstall 3862
IBInstall.TIBUnInstall.UnInstallCheck 3864
IBInstall.TIBUnInstall.UnInstallExecute 3864
IBInstall.TIBUnInstall.UnInstallFile 3864
IBInstall.TInstallOptions 3865
IBInstall.TInstallOptions.CmdLineTools 3865
IBInstall.TInstallOptions.ConnectivityClients 3866
IBInstall.TInstallOptions.Examples 3866
IBInstall.TInstallOptions.MainComponents 3866
IBInstall.TIscError 3875
IBInstall.TIscError type 3875
IBInstall.TMainOption 3871
IBInstall.TMainOption enumeration 3871
IBInstall.TMainOptions 3876
IBInstall.TMainOptions type 3876
IBInstall.TStatusResult 3872
IBInstall.TStatusResult enumeration 3872
IBInstall.TWarningResult 3872
IBInstall.TWarningResult enumeration 3872
IBIntf 3877
IBIntf namespace 3877
Functions 3880
Variables 3891
IBIntf.BLOB_get 3892
IBIntf.BLOB_get variable 3892
IBIntf.BLOB_put 3892
IBIntf.BLOB_put variable 3892
IBIntf.CheckIBInstallLoaded 3880
IBIntf.CheckIBInstallLoaded function 3880
IBIntf.CheckIBLoaded 3881
IBIntf.CheckIBLoaded function 3881
IBIntf.FreeIBInstallLibrary 3881
IBIntf.FreeIBInstallLibrary function 3881
IBIntf.FreeIBLibrary 3881
IBIntf.FreeIBLibrary function 3881
IBIntf.GetIBClientVersion 3882
IBIntf.GetIBClientVersion function 3882
IBIntf.isc_add_user 3893
IBIntf.isc_add_user variable 3893
IBIntf.isc_attach_database 3893
IBIntf.isc_attach_database variable 3893
IBIntf.isc_blob_info 3893
IBIntf.isc_blob_info variable 3893
IBIntf.isc_cancel_events 3894
IBIntf.isc_cancel_events variable 3894
IBIntf.isc_close_blob 3894
IBIntf.isc_close_blob variable 3894
IBIntf.isc_commit_retaining 3894
IBIntf.isc_commit_retaining variable 3894
IBIntf.isc_commit_transaction 3895
IBIntf.isc_commit_transaction variable 3895
IBIntf.isc_create_blob2 3895
IBIntf.isc_create_blob2 variable 3895
IBIntf.isc_database_info 3895
IBIntf.isc_database_info variable 3895
IBIntf.isc_decode_date 3896
IBIntf.isc_decode_date variable 3896
IBIntf.isc_decode_sql_date 3896
IBIntf.isc_decode_sql_date variable 3896
IBIntf.isc_decode_sql_date_stub 3882

IBIntf.isc_decode_sql_date_stub function 3882
IBIntf.isc_decode_sql_time 3896
IBIntf.isc_decode_sql_time variable 3896
IBIntf.isc_decode_sql_time_stub 3882
IBIntf.isc_decode_sql_time_stub function 3882
IBIntf.isc_decode_timestamp 3897
IBIntf.isc_decode_timestamp variable 3897
IBIntf.isc_decode_timestamp_stub 3883
IBIntf.isc_decode_timestamp_stub function 3883
IBIntf.isc_delete_user 3897
IBIntf.isc_delete_user variable 3897
IBIntf.isc_detach_database 3897
IBIntf.isc_detach_database variable 3897
IBIntf.isc_drop_database 3898
IBIntf.isc_drop_database variable 3898
IBIntf.isc_dsql_alloc_statement2 3898
IBIntf.isc_dsql_alloc_statement2 variable 3898
IBIntf.isc_dsql_describe 3898
IBIntf.isc_dsql_describe variable 3898
IBIntf.isc_dsql_describe_bind 3899
IBIntf.isc_dsql_describe_bind variable 3899
IBIntf.isc_dsql_execute 3899
IBIntf.isc_dsql_execute variable 3899
IBIntf.isc_dsql_execute_immediate 3899
IBIntf.isc_dsql_execute_immediate variable 3899
IBIntf.isc_dsql_execute2 3900
IBIntf.isc_dsql_execute2 variable 3900
IBIntf.isc_dsql_fetch 3900
IBIntf.isc_dsql_fetch variable 3900
IBIntf.isc_dsql_free_statement 3900
IBIntf.isc_dsql_free_statement variable 3900
IBIntf.isc_dsql_prepare 3901
IBIntf.isc_dsql_prepare variable 3901
IBIntf.isc_dsql_set_cursor_name 3901
IBIntf.isc_dsql_set_cursor_name variable 3901
IBIntf.isc_dsql_sql_info 3901
IBIntf.isc_dsql_sql_info variable 3901
IBIntf.isc_encode_date 3902
IBIntf.isc_encode_date variable 3902
IBIntf.isc_encode_sql_date 3902
IBIntf.isc_encode_sql_date variable 3902
IBIntf.isc_encode_sql_date_stub 3883
IBIntf.isc_encode_sql_date_stub function 3883
IBIntf.isc_encode_sql_time 3902
IBIntf.isc_encode_sql_time variable 3902
IBIntf.isc_encode_sql_time_stub 3883
IBIntf.isc_encode_sql_time_stub function 3883
IBIntf.isc_encode_timestamp 3903
IBIntf.isc_encode_timestamp variable 3903
IBIntf.isc_encode_timestamp_stub 3884
IBIntf.isc_encode_timestamp_stub function 3884
IBIntf.isc_event_block 3903
IBIntf.isc_event_block variable 3903
IBIntf.isc_event_counts 3903
IBIntf.isc_event_counts variable 3903
IBIntf.isc_free 3904
IBIntf.isc_free variable 3904
IBIntf.isc_get_segment 3904
IBIntf.isc_get_segment variable 3904
IBIntf.isc_install_clear_options 3904
IBIntf.isc_install_clear_options variable 3904
IBIntf.isc_install_clear_options_stub 3884
IBIntf.isc_install_clear_options_stub function 3884
IBIntf.isc_install_execute 3905
IBIntf.isc_install_execute variable 3905
IBIntf.isc_install_execute_stub 3884
IBIntf.isc_install_execute_stub function 3884
IBIntf.isc_install_get_info 3905
IBIntf.isc_install_get_info variable 3905
IBIntf.isc_install_get_info_stub 3885
IBIntf.isc_install_get_info_stub function 3885
IBIntf.isc_install_get_message 3905
IBIntf.isc_install_get_message variable 3905
IBIntf.isc_install_get_message_stub 3885
IBIntf.isc_install_get_message_stub function 3885
IBIntf.isc_install_load_external_text 3906
IBIntf.isc_install_load_external_text variable 3906
IBIntf.isc_install_load_external_text_stub 3885

IBIntf.isc_install_load_external_text_stub function 3885
IBIntf.isc_install_precheck 3906
IBIntf.isc_install_precheck variable 3906
IBIntf.isc_install_precheck_stub 3886
IBIntf.isc_install_precheck_stub function 3886
IBIntf.isc_install_set_option 3906
IBIntf.isc_install_set_option variable 3906
IBIntf.isc_install_set_option_stub 3886
IBIntf.isc_install_set_option_stub function 3886
IBIntf.isc_install_unset_option 3907
IBIntf.isc_install_unset_option variable 3907
IBIntf.isc_install_unset_option_stub 3886
IBIntf.isc_install_unset_option_stub function 3886
IBIntf.isc_interprete 3907
IBIntf.isc_interprete variable 3907
IBIntf.isc_modify_user 3907
IBIntf.isc_modify_user variable 3907
IBIntf.isc_open_blob2 3908
IBIntf.isc_open_blob2 variable 3908
IBIntf.isc_prepare_transaction 3908
IBIntf.isc_prepare_transaction variable 3908
IBIntf.isc_prepare_transaction2 3908
IBIntf.isc_prepare_transaction2 variable 3908
IBIntf.isc_put_segment 3909
IBIntf.isc_put_segment variable 3909
IBIntf.isc_que_events 3909
IBIntf.isc_que_events variable 3909
IBIntf.isc_rollback_retaining 3909
IBIntf.isc_rollback_retaining variable 3909
IBIntf.isc_rollback_retaining_stub 3887
IBIntf.isc_rollback_retaining_stub function 3887
IBIntf.isc_rollback_transaction 3910
IBIntf.isc_rollback_transaction variable 3910
IBIntf.isc_service_attach 3910
IBIntf.isc_service_attach variable 3910
IBIntf.isc_service_attach_stub 3887
IBIntf.isc_service_attach_stub function 3887
IBIntf.isc_service_detach 3910
IBIntf.isc_service_detach variable 3910
IBIntf.isc_service_detach_stub 3887
IBIntf.isc_service_detach_stub function 3887
IBIntf.isc_service_query 3911
IBIntf.isc_service_query variable 3911
IBIntf.isc_service_query_stub 3888
IBIntf.isc_service_query_stub function 3888
IBIntf.isc_service_start 3911
IBIntf.isc_service_start variable 3911
IBIntf.isc_service_start_stub 3888
IBIntf.isc_service_start_stub function 3888
IBIntf.isc_sql_interprete 3911
IBIntf.isc_sql_interprete variable 3911
IBIntf.isc_sqlcode 3912
IBIntf.isc_sqlcode variable 3912
IBIntf.isc_start_multiple 3912
IBIntf.isc_start_multiple variable 3912
IBIntf.isc_uninstall_execute 3912
IBIntf.isc_uninstall_execute variable 3912
IBIntf.isc_uninstall_execute_stub 3888
IBIntf.isc_uninstall_execute_stub function 3888
IBIntf.isc_uninstall_precheck 3913
IBIntf.isc_uninstall_precheck variable 3913
IBIntf.isc_uninstall_precheck_stub 3889
IBIntf.isc_uninstall_precheck_stub function 3889
IBIntf.isc_vax_integer 3913
IBIntf.isc_vax_integer variable 3913
IBIntf.LoadIBInstallLibrary 3889
IBIntf.LoadIBInstallLibrary function 3889
IBIntf.LoadIBLibrary 3889
IBIntf.LoadIBLibrary function 3889
IBIntf.TryIBLoad 3890
IBIntf.TryIBLoad function 3890
IBOA interface 6213
    about IBOA interface 6213
    Deactivate 6214
    GetPrincipal 6214
    GetPrincipalLength 6214
    IBOA Members 6213
    IBOA methods 6214

ImplIsReady 6214
ObjIsReady 6215
IBQuery 3915
IBQuery namespace 3915
Classes 3916
IBQuery.TIBQuery 3916
IBQuery.TIBQuery.Active 3927
IBQuery.TIBQuery.AfterDatabaseDisconnect 3934
IBQuery.TIBQuery.AfterTransactionEnd 3934
IBQuery.TIBQuery.BatchInput 3922
IBQuery.TIBQuery.BatchOutput 3922
IBQuery.TIBQuery.BeforeDatabaseDisconnect 3934
IBQuery.TIBQuery.BeforeTransactionEnd 3935
IBQuery.TIBQuery.BufferChunks 3927
IBQuery.TIBQuery.CachedUpdates 3928
IBQuery.TIBQuery.Constraints 3928
IBQuery.TIBQuery.Create 3921
IBQuery.TIBQuery.DatabaseFree 3935
IBQuery.TIBQuery.DataSource 3928
IBQuery.TIBQuery.DefineProperties 3922
IBQuery.TIBQuery.Destroy 3921
IBQuery.TIBQuery.Disconnect 3923
IBQuery.TIBQuery.ExecSQL 3923
IBQuery.TIBQuery.Filtered 3929
IBQuery.TIBQuery.GenerateParamNames 3929
IBQuery.TIBQuery.GenerateQueryForLiveUpdate 3923
IBQuery.TIBQuery.GeneratorField 3929
IBQuery.TIBQuery.GetDetailLinkFields 3923
IBQuery.TIBQuery.GetParamsCount 3924
IBQuery.TIBQuery.InitFieldDefs 3924
IBQuery.TIBQuery.InternalOpen 3924
IBQuery.TIBQuery.LiveMode 3929
IBQuery.TIBQuery.OnFilterRecord 3930
IBQuery.TIBQuery.ParamByName 3924
IBQuery.TIBQuery.ParamCheck 3930
IBQuery.TIBQuery.ParamCount 3930
IBQuery.TIBQuery.Params 3931
IBQuery.TIBQuery.Prepare 3925
IBQuery.TIBQuery.Prepared 3931
IBQuery.TIBQuery.PSExecute 3925
IBQuery.TIBQuery.PSGetParams 3925
IBQuery.TIBQuery.PSGetTableName 3925
IBQuery.TIBQuery.PSSetCommandText 3926
IBQuery.TIBQuery.PSSetParams 3926
IBQuery.TIBQuery.RowsAffected 3931
IBQuery.TIBQuery.SetFiltered 3926
IBQuery.TIBQuery.SQL 3932
IBQuery.TIBQuery.StatementType 3932
IBQuery.TIBQuery.StmtHandle 3932
IBQuery.TIBQuery.Text 3932
IBQuery.TIBQuery.TransactionFree 3935
IBQuery.TIBQuery.UniDirectional 3933
IBQuery.TIBQuery.UnPrepare 3926
IBQuery.TIBQuery.UpdateObject 3933
IBServices 3937
IBServices namespace 3937
Classes 3939
Constants 4032
Structs, Records, Enums 4018
Types 4027
IBServices.DefaultBufferSize 4032
IBServices.DefaultBufferSize constant 4032
IBServices.SPBConstantNames 4032
IBServices.SPBConstantNames constant 4032
IBServices.SPBConstantValues 4033
IBServices.SPBConstantValues constant 4033
IBServices.SPBPrefix 4033
IBServices.SPBPrefix constant 4033
IBServices.TBackupOption 4018
IBServices.TBackupOption enumeration 4018
IBServices.TBackupOptions 4027
IBServices.TBackupOptions type 4027
IBServices.TConfigFileData 3939
IBServices.TConfigFileData.ConfigFileKey 3941
IBServices.TConfigFileData.ConfigFileValue 3941
IBServices.TConfigFileData.Create 3940
IBServices.TConfigFileData.Destroy 3940
IBServices.TConfigParams 3941

IBServices.TConfigParams.BaseLocation 3942
IBServices.TConfigParams.ConfigFileData 3943
IBServices.TConfigParams.ConfigFileParams 3943
IBServices.TConfigParams.Create 3942
IBServices.TConfigParams.Destroy 3942
IBServices.TConfigParams.LockFileLocation 3943
IBServices.TConfigParams.MessageFileLocation 3943
IBServices.TConfigParams.SecurityDatabaseLocation 3944
IBServices.TDatabaseInfo 3944
IBServices.TDatabaseInfo.Create 3945
IBServices.TDatabaseInfo.DbName 3945
IBServices.TDatabaseInfo.Destroy 3945
IBServices.TDatabaseInfo.NoOfAttachments 3945
IBServices.TDatabaseInfo.NoOfDatabases 3946
IBServices.TIBBackupRestoreService 3946
IBServices.TIBBackupRestoreService.Verbose 3948
IBServices.TIBBackupService 3948
IBServices.TIBBackupService.BackupFile 3951
IBServices.TIBBackupService.BlockingFactor 3951
IBServices.TIBBackupService.Create 3950
IBServices.TIBBackupService.DatabaseName 3951
IBServices.TIBBackupService.Destroy 3950
IBServices.TIBBackupService.Options 3951
IBServices.TIBBackupService.SetServiceStartOptions 3950
IBServices.TIBConfigService 3952
IBServices.TIBConfigService.ActivateShadow 3954
IBServices.TIBConfigService.BringDatabaseOnline 3954
IBServices.TIBConfigService.DatabaseName 3957
IBServices.TIBConfigService.ServiceStart 3954
IBServices.TIBConfigService.SetAsyncMode 3955
IBServices.TIBConfigService.SetDBSqlDialect 3955
IBServices.TIBConfigService.SetPageBuffers 3955
IBServices.TIBConfigService.SetReadOnly 3956
IBServices.TIBConfigService.SetReserveSpace 3956
IBServices.TIBConfigService.SetSweepInterval 3956
IBServices.TIBConfigService.ShutdownDatabase 3957
IBServices.TIBControlAndQueryService 3957
IBServices.TIBControlAndQueryService.Action 3960
IBServices.TIBControlAndQueryService.BufferSize 3961
IBServices.TIBControlAndQueryService.Create 3959
IBServices.TIBControlAndQueryService.Eof 3961
IBServices.TIBControlAndQueryService.GetNextChunk 3959
IBServices.TIBControlAndQueryService.GetNextLine 3960
IBServices.TIBControlService 3961
IBServices.TIBControlService.Create 3963
IBServices.TIBControlService.InternalServiceStart 3963
IBServices.TIBControlService.IsServiceRunning 3965
IBServices.TIBControlService.ServiceStart 3963
IBServices.TIBControlService.ServiceStartAddParam 3964
IBServices.TIBControlService.ServiceStartParams 3965
IBServices.TIBControlService.SetServiceStartOptions 3964
IBServices.TIBCustomService 3965
IBServices.TIBCustomService.Active 3970
IBServices.TIBCustomService.Attach 3967
IBServices.TIBCustomService.BufferSize 3970
IBServices.TIBCustomService.CheckActive 3968
IBServices.TIBCustomService.CheckInactive 3968
IBServices.TIBCustomService.Create 3966
IBServices.TIBCustomService.Destroy 3967
IBServices.TIBCustomService.Detach 3968
IBServices.TIBCustomService.Handle 3970
IBServices.TIBCustomService.InternalServiceQuery 3968
IBServices.TIBCustomService.Loaded 3968
IBServices.TIBCustomService.Login 3969
IBServices.TIBCustomService.LoginPrompt 3971
IBServices.TIBCustomService.OnAttach 3973
IBServices.TIBCustomService.OnLogin 3973
IBServices.TIBCustomService.OutputBuffer 3971
IBServices.TIBCustomService.OutputBufferOption 3971
IBServices.TIBCustomService.Params 3971
IBServices.TIBCustomService.Protocol 3971
IBServices.TIBCustomService.ServerName 3972
IBServices.TIBCustomService.ServiceParamBySPB 3972
IBServices.TIBCustomService.ServiceQueryParams 3972
IBServices.TIBCustomService.TraceFlags 3972
IBServices.TIBLicensingService 3973
IBServices.TIBLicensingService.Action 3976
IBServices.TIBLicensingService.AddLicense 3975

IBServices.TIBLicensingService.ID 3977
IBServices.TIBLicensingService.Key 3977
IBServices.TIBLicensingService.RemoveLicense 3976
IBServices.TIBLicensingService.SetServiceStartOptions 3976
IBServices.TIBLogService 3977
IBServices.TIBLogService.SetServiceStartOptions 3979
IBServices.TIBRestoreService 3979
IBServices.TIBRestoreService.BackupFile 3982
IBServices.TIBRestoreService.Create 3981
IBServices.TIBRestoreService.DatabaseName 3982
IBServices.TIBRestoreService.Destroy 3981
IBServices.TIBRestoreService.Options 3983
IBServices.TIBRestoreService.PageBuffers 3983
IBServices.TIBRestoreService.PageSize 3983
IBServices.TIBRestoreService.SetServiceStartOptions 3982
IBServices.TIBSecurityService 3983
IBServices.TIBSecurityService.AddUser 3986
IBServices.TIBSecurityService.Create 3985
IBServices.TIBSecurityService.DeleteUser 3986
IBServices.TIBSecurityService.Destroy 3986
IBServices.TIBSecurityService.DisplayUser 3987
IBServices.TIBSecurityService.DisplayUsers 3987
IBServices.TIBSecurityService.FirstName 3989
IBServices.TIBSecurityService.GroupID 3989
IBServices.TIBSecurityService.LastName 3989
IBServices.TIBSecurityService.Loaded 3987
IBServices.TIBSecurityService.MiddleName 3990
IBServices.TIBSecurityService.ModifyUser 3988
IBServices.TIBSecurityService.Password 3990
IBServices.TIBSecurityService.SecurityAction 3990
IBServices.TIBSecurityService.SetServiceStartOptions 3988
IBServices.TIBSecurityService.SQlRole 3991
IBServices.TIBSecurityService.UserID 3991
IBServices.TIBSecurityService.UserInfo 3991
IBServices.TIBSecurityService.UserInfoCount 3991
IBServices.TIBSecurityService.UserName 3992
IBServices.TIBServerProperties 3992
IBServices.TIBServerProperties.ConfigParams 3996
IBServices.TIBServerProperties.Create 3993
IBServices.TIBServerProperties.DatabaseInfo 3997
IBServices.TIBServerProperties.Destroy 3994
IBServices.TIBServerProperties.Fetch 3994
IBServices.TIBServerProperties.FetchConfigParams 3995
IBServices.TIBServerProperties.FetchDatabaseInfo 3995
IBServices.TIBServerProperties.FetchLicenseInfo 3995
IBServices.TIBServerProperties.FetchLicenseMaskInfo 3995
IBServices.TIBServerProperties.FetchVersionInfo 3996
IBServices.TIBServerProperties.LicenseInfo 3997
IBServices.TIBServerProperties.LicenseMaskInfo 3997
IBServices.TIBServerProperties.Options 3997
IBServices.TIBServerProperties.VersionInfo 3998
IBServices.TIBStatisticalService 3998
IBServices.TIBStatisticalService.DatabaseName 4000
IBServices.TIBStatisticalService.Options 4001
IBServices.TIBStatisticalService.SetServiceStartOptions 4000
IBServices.TIBValidationService 4001
IBServices.TIBValidationService.Create 4003
IBServices.TIBValidationService.DatabaseName 4005
IBServices.TIBValidationService.Destroy 4003
IBServices.TIBValidationService.FetchLimboTransactionInfo 4003
IBServices.TIBValidationService.FixLimboTransactionErrors 4004
IBServices.TIBValidationService.GlobalAction 4005
IBServices.TIBValidationService.LimboTransactionInfo 4005
IBServices.TIBValidationService.LimboTransactionInfoCount 4005
IBServices.TIBValidationService.Options 4006
IBServices.TIBValidationService.SetServiceStartOptions 4004
IBServices.TLicenseInfo 4006
IBServices.TLicenseInfo.Create 4007
IBServices.TLicenseInfo.Desc 4007
IBServices.TLicenseInfo.Destroy 4007
IBServices.TLicenseInfo.Id 4008
IBServices.TLicenseInfo.Key 4008
IBServices.TLicenseInfo.LicensedUsers 4008
IBServices.TLicenseMaskInfo 4008
IBServices.TLicenseMaskInfo.CapabilityMask 4009
IBServices.TLicenseMaskInfo.LicenseMask 4009

IBServices.TLicensingAction 4019
IBServices.TLicensingAction enumeration 4019
IBServices.TLimboTransactionInfo 4010
IBServices.TLimboTransactionInfo.Action 4011
IBServices.TLimboTransactionInfo.Advise 4011
IBServices.TLimboTransactionInfo.HostSite 4011
IBServices.TLimboTransactionInfo.ID 4012
IBServices.TLimboTransactionInfo.MultiDatabase 4012
IBServices.TLimboTransactionInfo.RemoteDatabasePath 4012
IBServices.TLimboTransactionInfo.RemoteSite 4012
IBServices.TLimboTransactionInfo.State 4012
IBServices.TLoginEvent 4028
IBServices.TLoginEvent type 4028
IBServices.TOutputBufferOption 4019
IBServices.TOutputBufferOption enumeration 4019
IBServices.TPropertyOption 4020
IBServices.TPropertyOption enumeration 4020
IBServices.TPropertyOptions 4028
IBServices.TPropertyOptions type 4028
IBServices.TProtocol 4020
IBServices.TProtocol enumeration 4020
IBServices.TRestoreOption 4021
IBServices.TRestoreOption enumeration 4021
IBServices.TRestoreOptions 4029
IBServices.TRestoreOptions type 4029
IBServices.TSecurityAction 4021
IBServices.TSecurityAction enumeration 4021
IBServices.TSecurityModifyParam 4022
IBServices.TSecurityModifyParam enumeration 4022
IBServices.TSecurityModifyParams 4030
IBServices.TSecurityModifyParams type 4030
IBServices.TShutdownMode 4022
IBServices.TShutdownMode enumeration 4022
IBServices.TStatOption 4023
IBServices.TStatOption enumeration 4023
IBServices.TStatOptions 4030
IBServices.TStatOptions type 4030
IBServices.TTransactionAction 4023
IBServices.TTransactionAction enumeration 4023
IBServices.TTransactionAdvise 4024
IBServices.TTransactionAdvise enumeration 4024
IBServices.TTransactionGlobalAction 4024
IBServices.TTransactionGlobalAction enumeration 4024
IBServices.TTransactionState 4025
IBServices.TTransactionState enumeration 4025
IBServices.TUserInfo 4013
IBServices.TUserInfo.FirstName 4014
IBServices.TUserInfo.GroupID 4014
IBServices.TUserInfo.LastName 4014
IBServices.TUserInfo.MiddleName 4014
IBServices.TUserInfo.UserID 4015
IBServices.TUserInfo.UserName 4015
IBServices.TValidateOption 4026
IBServices.TValidateOption enumeration 4026
IBServices.TValidateOptions 4031
IBServices.TValidateOptions type 4031
IBServices.TVersionInfo 4015
IBServices.TVersionInfo.ServerImplementation 4016
IBServices.TVersionInfo.ServerVersion 4016
IBServices.TVersionInfo.ServiceVersion 4016
IBSQL 4035
IBSQL namespace 4035
    Classes 4036
    Structs, Records, Enums 4098
    Types 4100
IBSQL.TIBBatch 4036
IBSQL.TIBBatch.Columns 4038
IBSQL.TIBBatch.FColumns 4037
IBSQL.TIBBatch.FFilename 4037
IBSQL.TIBBatch.Filename 4039
IBSQL.TIBBatch.FParams 4038
IBSQL.TIBBatch.Params 4039
IBSQL.TIBBatch.ReadyFile 4038
IBSQL.TIBBatchInput 4039
IBSQL.TIBBatchInput.ReadParameters 4040
IBSQL.TIBBatchOutput 4040
IBSQL.TIBBatchOutput.WriteColumns 4041
IBSQL.TIBInputDelimitedFile 4042

IBSQL.TIBInputDelimitedFile.ColDelimiter 4046
IBSQL.TIBInputDelimitedFile.Destroy 4043
IBSQL.TIBInputDelimitedFile.FColDelimiter 4043
IBSQL.TIBInputDelimitedFile.FEOF 4043
IBSQL.TIBInputDelimitedFile.FFile 4044
IBSQL.TIBInputDelimitedFile.FLookAhead 4044
IBSQL.TIBInputDelimitedFile.FReadBlanksAsNull 4044
IBSQL.TIBInputDelimitedFile.FRowDelimiter 4044
IBSQL.TIBInputDelimitedFile.FSkipTitles 4045
IBSQL.TIBInputDelimitedFile.GetColumn 4045
IBSQL.TIBInputDelimitedFile.ReadBlanksAsNull 4046
IBSQL.TIBInputDelimitedFile.ReadParameters 4045
IBSQL.TIBInputDelimitedFile.ReadyFile 4045
IBSQL.TIBInputDelimitedFile.RowDelimiter 4046
IBSQL.TIBInputDelimitedFile.SkipTitles 4047
IBSQL.TIBInputRawFile 4047
IBSQL.TIBInputRawFile.Destroy 4048
IBSQL.TIBInputRawFile.FFile 4048
IBSQL.TIBInputRawFile.ReadParameters 4049
IBSQL.TIBInputRawFile.ReadyFile 4049
IBSQL.TIBOutputDelimitedFile 4049
IBSQL.TIBOutputDelimitedFile.ColDelimiter 4052
IBSQL.TIBOutputDelimitedFile.Destroy 4050
IBSQL.TIBOutputDelimitedFile.FColDelimiter 4051
IBSQL.TIBOutputDelimitedFile.FFile 4051
IBSQL.TIBOutputDelimitedFile.FOutputTitles 4051
IBSQL.TIBOutputDelimitedFile.FRowDelimiter 4051
IBSQL.TIBOutputDelimitedFile.OutputTitles 4053
IBSQL.TIBOutputDelimitedFile.ReadyFile 4052
IBSQL.TIBOutputDelimitedFile.RowDelimiter 4053
IBSQL.TIBOutputDelimitedFile.WriteColumns 4052
IBSQL.TIBOutputRawFile 4053
IBSQL.TIBOutputRawFile.Destroy 4054
IBSQL.TIBOutputRawFile.FFile 4054
IBSQL.TIBOutputRawFile.ReadyFile 4055
IBSQL.TIBOutputRawFile.WriteColumns 4055
IBSQL.TIBSQL 4055
IBSQL.TIBSQL.BatchInput 4063
IBSQL.TIBSQL.BatchOutput 4064
IBSQL.TIBSQL.BeforeTransactionEnd 4064
IBSQL.TIBSQL.Bof 4072
IBSQL.TIBSQL.Call 4064
IBSQL.TIBSQL.CheckClosed 4064
IBSQL.TIBSQL.CheckOpen 4065
IBSQL.TIBSQL.CheckValidStatement 4065
IBSQL.TIBSQL.Close 4065
IBSQL.TIBSQL.Create 4058
IBSQL.TIBSQL.Current 4065
IBSQL.TIBSQL.Database 4072
IBSQL.TIBSQL.DBHandle 4072
IBSQL.TIBSQL.Destroy 4058
IBSQL.TIBSQL.DoBeforeDatabaseDisconnect 4066
IBSQL.TIBSQL.Eof 4072
IBSQL.TIBSQL.ExecQuery 4066
IBSQL.TIBSQL.FBase 4059
IBSQL.TIBSQL.FBOF 4059
IBSQL.TIBSQL.FCursor 4059
IBSQL.TIBSQL.FEOF 4059
IBSQL.TIBSQL.FGenerateParamNames 4060
IBSQL.TIBSQL.FGoToFirstRecordOnExecute 4060
IBSQL.TIBSQL.FHandle 4060
IBSQL.TIBSQL.FieldByName 4066
IBSQL.TIBSQL.FieldIndex 4073
IBSQL.TIBSQL.Fields 4073
IBSQL.TIBSQL.FOnSQLChanging 4060
IBSQL.TIBSQL.FOpen 4061
IBSQL.TIBSQL.FParamCheck 4061
IBSQL.TIBSQL.FPrepared 4061
IBSQL.TIBSQL.FProcessedSQL 4061
IBSQL.TIBSQL.FRecordCount 4061
IBSQL.TIBSQL.FreeHandle 4066
IBSQL.TIBSQL.FSQL 4062
IBSQL.TIBSQL.FSQLParams 4062
IBSQL.TIBSQL.FSQLRecord 4062
IBSQL.TIBSQL.FSQLType 4062
IBSQL.TIBSQL.GenerateParamNames 4073
IBSQL.TIBSQL.GetDatabase 4067
IBSQL.TIBSQL.GetDBHandle 4067

IBSQL.TIBSQL.GetEOF 4067
IBSQL.TIBSQL.GetFieldIndex 4067
IBSQL.TIBSQL.GetFields 4067
IBSQL.TIBSQL.GetPlan 4068
IBSQL.TIBSQL.GetRecordCount 4068
IBSQL.TIBSQL.GetRowsAffected 4068
IBSQL.TIBSQL.GetSQLParams 4068
IBSQL.TIBSQL.GetTransaction 4069
IBSQL.TIBSQL.GetTRHandle 4069
IBSQL.TIBSQL.GetUniqueRelationName 4069
IBSQL.TIBSQL.GoToFirstRecordOnExecute 4074
IBSQL.TIBSQL.Handle 4074
IBSQL.TIBSQL.Next 4069
IBSQL.TIBSQL.OnSQLChanging 4077
IBSQL.TIBSQL.Open 4074
IBSQL.TIBSQL.ParamByName 4069
IBSQL.TIBSQL.ParamCheck 4074
IBSQL.TIBSQL.Params 4075
IBSQL.TIBSQL.Plan 4075
IBSQL.TIBSQL.Prepare 4070
IBSQL.TIBSQL.Prepared 4075
IBSQL.TIBSQL.PreprocessSQL 4070
IBSQL.TIBSQL.RecordCount 4075
IBSQL.TIBSQL.RowsAffected 4076
IBSQL.TIBSQL.SetDatabase 4070
IBSQL.TIBSQL.SetSQL 4070
IBSQL.TIBSQL.SetTransaction 4071
IBSQL.TIBSQL.SQL 4076
IBSQL.TIBSQL.SQLChanging 4071
IBSQL.TIBSQL.SQLType 4076
IBSQL.TIBSQL.Transaction 4077
IBSQL.TIBSQL.TRHandle 4077
IBSQL.TIBSQL.UniqueRelationName 4077
IBSQL.TIBSQLTypes 4098
IBSQL.TIBSQLTypes enumeration 4098
IBSQL.TIBXSQLDA 4078
IBSQL.TIBXSQLDA.AddName 4082
IBSQL.TIBXSQLDA.AsXSQLDA 4084
IBSQL.TIBXSQLDA.ByName 4082
IBSQL.TIBXSQLDA.Count 4085
IBSQL.TIBXSQLDA.Create 4079
IBSQL.TIBXSQLDA.Destroy 4079
IBSQL.TIBXSQLDA.FCount 4080
IBSQL.TIBXSQLDA.FNames 4080
IBSQL.TIBXSQLDA.FSize 4080
IBSQL.TIBXSQLDA.FSQL 4081
IBSQL.TIBXSQLDA.FUniqueRelationName 4081
IBSQL.TIBXSQLDA.FXSQLDA 4081
IBSQL.TIBXSQLDA.FXSQLVARs 4081
IBSQL.TIBXSQLDA.GetModified 4082
IBSQL.TIBXSQLDA.GetNames 4083
IBSQL.TIBXSQLDA.GetRecordSize 4083
IBSQL.TIBXSQLDA.GetXSQLDA 4083
IBSQL.TIBXSQLDA.GetXSQLVAR 4083
IBSQL.TIBXSQLDA.GetXSQLVARByName 4083
IBSQL.TIBXSQLDA.Initialize 4084
IBSQL.TIBXSQLDA.Modified 4085
IBSQL.TIBXSQLDA.Names 4085
IBSQL.TIBXSQLDA.RecordSize 4085
IBSQL.TIBXSQLDA.SetCount 4084
IBSQL.TIBXSQLDA.UniqueRelationName 4086
IBSQL.TIBXSQLDA.Vars 4086
IBSQL.TIBXSQLVAR 4086
IBSQL.TIBXSQLVAR.AsCurrency 4090
IBSQL.TIBXSQLVAR.AsDate 4090
IBSQL.TIBXSQLVAR.AsDateTime 4091
IBSQL.TIBXSQLVAR.AsDouble 4091
IBSQL.TIBXSQLVAR.AsFloat 4091
IBSQL.TIBXSQLVAR.AsInt64 4092
IBSQL.TIBXSQLVAR.AsInteger 4092
IBSQL.TIBXSQLVAR.AsLong 4092
IBSQL.TIBXSQLVAR.AsPointer 4092
IBSQL.TIBXSQLVAR.AsQuad 4093
IBSQL.TIBXSQLVAR.AsShort 4093
IBSQL.TIBXSQLVAR.Assign 4088
IBSQL.TIBXSQLVAR.AsString 4093
IBSQL.TIBXSQLVAR.AsTime 4093
IBSQL.TIBXSQLVAR.AsTrimString 4094

IBSQL.TIBXSQLVAR.AsVariant 4094
IBSQL.TIBXSQLVAR.AsXSQLVAR 4094
IBSQL.TIBXSQLVAR.Clear 4088
IBSQL.TIBXSQLVAR.Create 4087
IBSQL.TIBXSQLVAR.Data 4095
IBSQL.TIBXSQLVAR.Index 4095
IBSQL.TIBXSQLVAR.IsNull 4095
IBSQL.TIBXSQLVAR.IsNullable 4095
IBSQL.TIBXSQLVAR.LoadFromFile 4089
IBSQL.TIBXSQLVAR.LoadFromStream 4089
IBSQL.TIBXSQLVAR.Modified 4096
IBSQL.TIBXSQLVAR.Name 4096
IBSQL.TIBXSQLVAR.SaveToFile 4089
IBSQL.TIBXSQLVAR.SaveToStream 4089
IBSQL.TIBXSQLVAR.Size 4096
IBSQL.TIBXSQLVAR.SQLType 4096
IBSQL.TIBXSQLVAR.Value 4097
IBSQL.TIBXSQLVARArray 4100
IBSQL.TIBXSQLVARArray type 4100
IBSQLMonitor 4101
IBSQLMonitor namespace 4101
Classes 4103
Constants 4120
Functions 4117
Interfaces 4109
Types 4119
IBSQLMonitor.DisableMonitoring 4117
IBSQLMonitor.DisableMonitoring function 4117
IBSQLMonitor.EnableMonitoring 4117
IBSQLMonitor.EnableMonitoring function 4117
IBSQLMonitor.IIBSQLMonitorHook 4109
IBSQLMonitor.IIBSQLMonitorHook.DBConnect 4110
IBSQLMonitor.IIBSQLMonitorHook.DBDisconnect 4111
IBSQLMonitor.IIBSQLMonitorHook.Enabled 4116
IBSQLMonitor.IIBSQLMonitorHook.GetEnabled 4111
IBSQLMonitor.IIBSQLMonitorHook.GetMonitorCount 4111
IBSQLMonitor.IIBSQLMonitorHook.GetTraceFlags 4111
IBSQLMonitor.IIBSQLMonitorHook.RegisterMonitor 4112
IBSQLMonitor.IIBSQLMonitorHook.ReleaseMonitor 4112
IBSQLMonitor.IIBSQLMonitorHook.SendMisc 4112
IBSQLMonitor.IIBSQLMonitorHook.ServiceAttach 4112
IBSQLMonitor.IIBSQLMonitorHook.ServiceDetach 4112
IBSQLMonitor.IIBSQLMonitorHook.ServiceQuery 4113
IBSQLMonitor.IIBSQLMonitorHook.ServiceStart 4113
IBSQLMonitor.IIBSQLMonitorHook.SetEnabled 4113
IBSQLMonitor.IIBSQLMonitorHook.SetTraceFlags 4113
IBSQLMonitor.IIBSQLMonitorHook.SQLExecute 4114
IBSQLMonitor.IIBSQLMonitorHook.SQLFetch 4114
IBSQLMonitor.IIBSQLMonitorHook.SQLPrepare 4114
IBSQLMonitor.IIBSQLMonitorHook.TraceFlags 4116
IBSQLMonitor.IIBSQLMonitorHook.TRCommit 4114
IBSQLMonitor.IIBSQLMonitorHook.TRCommitRetaining 4114
IBSQLMonitor.IIBSQLMonitorHook.TRRollback 4115
IBSQLMonitor.IIBSQLMonitorHook.TRRollbackRetaining 4115
IBSQLMonitor.IIBSQLMonitorHook.TRStart 4115
IBSQLMonitor.IIBSQLMonitorHook.UnregisterMonitor 4115
IBSQLMonitor.MonitorHook 4118
IBSQLMonitor.MonitorHook function 4118
IBSQLMonitor.MonitoringEnabled 4118
IBSQLMonitor.MonitoringEnabled function 4118
IBSQLMonitor.TIBCustomSQLMonitor 4103
IBSQLMonitor.TIBCustomSQLMonitor.Create 4104
IBSQLMonitor.TIBCustomSQLMonitor.Destroy 4104
IBSQLMonitor.TIBCustomSQLMonitor.Enabled 4105
IBSQLMonitor.TIBCustomSQLMonitor.Handle 4105
IBSQLMonitor.TIBCustomSQLMonitor.OnSQL 4106
IBSQLMonitor.TIBCustomSQLMonitor.Release 4104
IBSQLMonitor.TIBCustomSQLMonitor.TraceFlags 4106
IBSQLMonitor.TIBSQLMonitor 4106
IBSQLMonitor.TIBSQLMonitor.Enabled 4107
IBSQLMonitor.TIBSQLMonitor.OnSQL 4108
IBSQLMonitor.TIBSQLMonitor.TraceFlags 4108
IBSQLMonitor.TSQLEvent 4119
IBSQLMonitor.TSQLEvent type 4119
IBSQLMonitor.WM_IBSQL_SQL_EVENT 4120
IBSQLMonitor.WM_IBSQL_SQL_EVENT constant 4120
IBSQLMonitor.WM_MAX_IBSQL_MONITOR 4120
IBSQLMonitor.WM_MAX_IBSQL_MONITOR constant 4120

IBSQLMonitor.WM_MIN_IBSQL_MONITOR 4120
IBSQLMonitor.WM_MIN_IBSQL_MONITOR constant 4120
IBStoredProc 4123
IBStoredProc namespace 4123
Classes 4124
IBStoredProc.TIBStoredProc 4124
IBStoredProc.TIBStoredProc.AfterDatabaseDisconnect 4137
IBStoredProc.TIBStoredProc.AfterTransactionEnd 4137
IBStoredProc.TIBStoredProc.BeforeDatabaseDisconnect 4138
IBStoredProc.TIBStoredProc.BeforeTransactionEnd 4138
IBStoredProc.TIBStoredProc.CopyParams 4130
IBStoredProc.TIBStoredProc.Create 4128
IBStoredProc.TIBStoredProc.DatabaseFree 4138
IBStoredProc.TIBStoredProc.DefineProperties 4130
IBStoredProc.TIBStoredProc.Destroy 4129
IBStoredProc.TIBStoredProc.Disconnect 4130
IBStoredProc.TIBStoredProc.ExecProc 4130
IBStoredProc.TIBStoredProc.Filtered 4134
IBStoredProc.TIBStoredProc.GetParamsCount 4131
IBStoredProc.TIBStoredProc.InternalOpen 4131
IBStoredProc.TIBStoredProc.OnFilterRecord 4135
IBStoredProc.TIBStoredProc.ParamByName 4131
IBStoredProc.TIBStoredProc.ParamCount 4135
IBStoredProc.TIBStoredProc.Params 4135
IBStoredProc.TIBStoredProc.Prepare 4132
IBStoredProc.TIBStoredProc.Prepared 4136
IBStoredProc.TIBStoredProc.PSExecute 4132
IBStoredProc.TIBStoredProc.PSGetParams 4132
IBStoredProc.TIBStoredProc.PSGetTableName 4132
IBStoredProc.TIBStoredProc.PSSetCommandText 4133
IBStoredProc.TIBStoredProc.PSSetParams 4133
IBStoredProc.TIBStoredProc.SetFiltered 4133
IBStoredProc.TIBStoredProc.SetPrepare 4133
IBStoredProc.TIBStoredProc.SetPrepared 4133
IBStoredProc.TIBStoredProc.SetProcName 4134
IBStoredProc.TIBStoredProc.StmtHandle 4136
IBStoredProc.TIBStoredProc.StoredProcedureNames 4136
IBStoredProc.TIBStoredProc.StoredProcName 4137
IBStoredProc.TIBStoredProc.TransactionFree 4138
IBStoredProc.TIBStoredProc.UnPrepare 4134
IBTable 4141
IBTable namespace 4141
Classes 4142
Structs, Records, Enums 4168
Types 4169
IBTable.TIBTable 4142
IBTable.TIBTable.Active 4157
IBTable.TIBTable.AddIndex 4148
IBTable.TIBTable.AfterDatabaseDisconnect 4165
IBTable.TIBTable.AfterTransactionEnd 4165
IBTable.TIBTable.BeforeDatabaseDisconnect 4166
IBTable.TIBTable.BeforeTransactionEnd 4166
IBTable.TIBTable.BufferChunks 4157
IBTable.TIBTable.CachedUpdates 4157
IBTable.TIBTable.Constraints 4158
IBTable.TIBTable.Create 4147
IBTable.TIBTable.CreateTable 4148
IBTable.TIBTable.CurrentDBKey 4158
IBTable.TIBTable.DatabaseFree 4166
IBTable.TIBTable.DataEvent 4149
IBTable.TIBTable.DefaultIndex 4158
IBTable.TIBTable.DefChanged 4149
IBTable.TIBTable.DeleteIndex 4149
IBTable.TIBTable.DeleteTable 4150
IBTable.TIBTable.Destroy 4147
IBTable.TIBTable.DoOnNewRecord 4150
IBTable.TIBTable.EmptyTable 4150
IBTable.TIBTable.Exists 4158
IBTable.TIBTable.FieldDefs 4159
IBTable.TIBTable.Filter 4159
IBTable.TIBTable.Filtered 4159
IBTable.TIBTable.GetCanModify 4150
IBTable.TIBTable.GetDataSource 4151
IBTable.TIBTable.GetDetailLinkFields 4151
IBTable.TIBTable.GetIndexNames 4151
IBTable.TIBTable.GetIndexParams 4151
IBTable.TIBTable.GotoCurrent 4152
IBTable.TIBTable.IndexDefs 4160

IBTable.TIBTable.IndexFieldCount 4160
IBTable.TIBTable.IndexFieldNames 4160
IBTable.TIBTable.IndexFields 4161
IBTable.TIBTable.IndexName 4161
IBTable.TIBTable.InitFieldDefs 4152
IBTable.TIBTable.InternalClose 4152
IBTable.TIBTable.InternalOpen 4152
IBTable.TIBTable.InternalRefresh 4153
IBTable.TIBTable.InternalRefreshRow 4153
IBTable.TIBTable.LiveMode 4161
IBTable.TIBTable.Locate 4153
IBTable.TIBTable.MasterFields 4162
IBTable.TIBTable.MasterSource 4162
IBTable.TIBTable.OnFilterRecord 4162
IBTable.TIBTable.PSGetDefaultOrder 4154
IBTable.TIBTable.PSGetIndexDefs 4154
IBTable.TIBTable.PSGetKeyFields 4154
IBTable.TIBTable.PSGetTableName 4154
IBTable.TIBTable.PSSetCommandText 4155
IBTable.TIBTable.PSSetParams 4155
IBTable.TIBTable.ReadOnly 4163
IBTable.TIBTable.SetFiltered 4155
IBTable.TIBTable.SetFilterOptions 4155
IBTable.TIBTable.SetFilterText 4156
IBTable.TIBTable.StoreDefs 4163
IBTable.TIBTable.TableName 4163
IBTable.TIBTable.TableNames 4164
IBTable.TIBTable.TableTypes 4164
IBTable.TIBTable.TransactionFree 4166
IBTable.TIBTable.UniDirectional 4164
IBTable.TIBTable.UpdateIndexDefs 4156
IBTable.TIBTable.UpdateObject 4165
IBTable.TIBTableType 4168
IBTable.TIBTableType enumeration 4168
IBTable.TIBTableTypes 4169
IBTable.TIBTableTypes type 4169
IBTable.TIndexName 4169
IBTable.TIndexName type 4169
IBUpdateSQL 4171
IBUpdateSQL namespace 4171
Classes 4172
IBUpdateSQL.TIBUpdateSQL 4172
IBUpdateSQL.TIBUpdateSQL.Apply 4174
IBUpdateSQL.TIBUpdateSQL.Create 4173
IBUpdateSQL.TIBUpdateSQL.DataSet 4177
IBUpdateSQL.TIBUpdateSQL.DeleteSQL 4177
IBUpdateSQL.TIBUpdateSQL.Destroy 4173
IBUpdateSQL.TIBUpdateSQL.ExecSQL 4175
IBUpdateSQL.TIBUpdateSQL.GetDataSet 4175
IBUpdateSQL.TIBUpdateSQL.GetSQL 4175
IBUpdateSQL.TIBUpdateSQL.InsertSQL 4178
IBUpdateSQL.TIBUpdateSQL.ModifySQL 4178
IBUpdateSQL.TIBUpdateSQL.Query 4178
IBUpdateSQL.TIBUpdateSQL.SetDataSet 4176
IBUpdateSQL.TIBUpdateSQL.SetParams 4176
IBUpdateSQL.TIBUpdateSQL.SQL 4179
IBUpdateSQL.TIBUpdateSQL.SQLChanged 4176
IBUtils 4181
IBUtils namespace 4181
Constants 4186
Functions 4182
IBUtils.AddIBParamSQLForDetail 4182
IBUtils.AddIBParamSQLForDetail function 4182
IBUtils.CR 4186
IBUtils.CR constant 4186
IBUtils.CRLF 4186
IBUtils.CRLF constant 4186
IBUtils.ExtractIdentifier 4182
IBUtils.ExtractIdentifier function 4182
IBUtils.FormatIdentifier 4183
IBUtils.FormatIdentifier function 4183
IBUtils.FormatIdentifierValue 4183
IBUtils.FormatIdentifierValue function 4183
IBUtils.LF 4187
IBUtils.LF constant 4187
IBUtils.Max 4183
IBUtils.Max function 4183
IBUtils.Min 4184

IBUtils.Min function 4184
IBUtils.NULL_TERMINATOR 4187
IBUtils.NULL_TERMINATOR constant 4187
IBUtils.QuoteIdentifier 4184
IBUtils.QuoteIdentifier function 4184
IBUtils.RandomInteger 4184
IBUtils.RandomInteger function 4184
IBUtils.RandomString 4185
IBUtils.RandomString function 4185
IBUtils.StripString 4185
IBUtils.StripString function 4185
IBUtils.TAB 4187
IBUtils.TAB constant 4187
ibxeBlobCannotBeRead enumeration member 3551
ibxeBlobCannotBeWritten enumeration member 3551
ibxeBOF enumeration member 3551
ibxeBufferNotSet enumeration member 3551
ibxeCannotCreatePrimaryIndex enumeration member 3551
ibxeCannotCreateSharedResource enumeration member 3551
ibxeCannotDelete enumeration member 3551
ibxeCannotDropSystemIndex enumeration member 3551
ibxeCannotInsert enumeration member 3551
ibxeCannotOpenNonSQLSelect enumeration member 3551
ibxeCannotPost enumeration member 3551
ibxeCannotRefresh enumeration member 3551
ibxeCannotSetDatabase enumeration member 3551
ibxeCannotSetTransaction enumeration member 3551
ibxeCannotUpdate enumeration member 3551
ibxeCantEndSharedTransaction enumeration member 3551
ibxeCantPrintValue enumeration member 3551
ibxeCircularDataLink enumeration member 3551
ibxeCircularReference enumeration member 3551
ibxeColumnIsNotNullable enumeration member 3551
ibxeColumnListsDontMatch enumeration member 3551
ibxeColumnTypesDontMatch enumeration member 3551
ibxeConnectionTimeout enumeration member 3551
ibxeDatabaseClosed enumeration member 3551
ibxeDatabaseNameMissing enumeration member 3551
ibxeDatabaseNotAssigned enumeration member 3551
ibxeDatabaseOpen enumeration member 3551
ibxeDatasetClosed enumeration member 3551
ibxeDatasetOpen enumeration member 3551
ibxeDataSetUniDirectional enumeration member 3551
ibxeDelphiException enumeration member 3551
ibxeDPBConstantNotSupported enumeration member 3551
ibxeDPBConstantUnknown enumeration member 3551
ibxeDPBConstantUnknownEx enumeration member 3551
ibxeEmptyQuery enumeration member 3551
ibxeEmptySQLStatement enumeration member 3551
ibxeEOF enumeration member 3551
ibxeEOFInComment enumeration member 3551
ibxeEOFInString enumeration member 3551
ibxeEOFReached enumeration member 3551
ibxeEventAlreadyRegistered enumeration member 3551
ibxeFieldNotFound enumeration member 3551
ibxeFieldReadOnly enumeration member 3551
ibxeFieldSizeMismatch enumeration member 3551
ibxeFieldUnsupportedType enumeration member 3551
ibxeFileAccessError enumeration member 3551
ibxeIB60feature enumeration member 3551
ibxeIndexFieldMissing enumeration member 3551
ibxeInterBaseInstallMissing enumeration member 3551
ibxeInterBaseMissing enumeration member 3551
ibxeInTransaction enumeration member 3551
ibxeInvalidBatchMove enumeration member 3551
ibxeInvalidCancellation enumeration member 3551
ibxeInvalidColumnIndex enumeration member 3551
ibxeInvalidDataConversion enumeration member 3551
ibxeInvalidEvent enumeration member 3551
ibxeInvalidOnErrorResult enumeration member 3551
ibxeInvalidOnStatusResult enumeration member 3551
ibxeInvalidOption enumeration member 3551
ibxeInvalidParamColumnIndex enumeration member 3551
ibxeInvalidQueueing enumeration member 3551
ibxeInvalidRegistration enumeration member 3551
ibxeInvalidStatementHandle enumeration member 3551
ibxeIsAExecuteProcedure enumeration member 3551
ibxeIsASelectStatement enumeration member 3551

ibxeMaximumEvents enumeration member 3551
ibxeNoDatabasesInTransaction enumeration member 3551
ibxeNoDestinationDirectory enumeration member 3551
ibxeNoEventsRegistered enumeration member 3551
ibxeNoFieldAccess enumeration member 3551
ibxeNoOptionsSet enumeration member 3551
ibxeNoProvider enumeration member 3551
ibxeNoRecordsAffected enumeration member 3551
ibxeNosourceDirectory enumeration member 3551
ibxeNoStoredProcName enumeration member 3551
ibxeNoTableName enumeration member 3551
ibxeNotCachedUpdates enumeration member 3551
ibxeNotEditing enumeration member 3551
ibxeNotInTransaction enumeration member 3551
ibxeNotLiveRequest enumeration member 3551
ibxeNotPermitted enumeration member 3551
ibxeNotSupported enumeration member 3551
ibxeNoUninstallFile enumeration member 3551
ibxeOperationCancelled enumeration member 3551
ibxeOptionNeedsClient enumeration member 3551
ibxeOptionNeedsServer enumeration member 3551
ibxeOutputParsingError enumeration member 3551
ibxeParamNameExpected enumeration member 3551
ibxeQueryParamsError enumeration member 3551
ibxeRequiredParamNotSet enumeration member 3551
ibxeServerNameMissing enumeration member 3551
ibxeServiceActive enumeration member 3551
ibxeServiceInActive enumeration member 3551
ibxeSPBConstantNotSupported enumeration member 3551
ibxeSPBConstantUnknown enumeration member 3551
ibxeSQLClosed enumeration member 3551
ibxeSQLDialectInvalid enumeration member 3551
ibxeSQLMonitorAlreadyPresent enumeration member 3551
ibxeSQLOpen enumeration member 3551
ibxeSQLParseError enumeration member 3551
ibxeStartParamsError enumeration member 3551
ibxeStringTooLarge enumeration member 3551
ibxeSuccess enumeration member 3551
ibxeTableNameMismatch enumeration member 3551
ibxeTimeoutNegative enumeration member 3551
ibxeTPBConstantNotSupported enumeration member 3551
ibxeTPBConstantUnknown enumeration member 3551
ibxeTPBConstantUnknownEx enumeration member 3551
ibxeTransactionNotAssigned enumeration member 3551
ibxeUnknownError enumeration member 3551
ibxeUnknownPlan enumeration member 3551
ibxeUnknownSQLDataType enumeration member 3551
ibxeUpdateFailed enumeration member 3551
ibxeUpdateWrongDB enumeration member 3551
ibxeUpdateWrongTR enumeration member 3551
ibxeUserAbort enumeration member 3551
ibxeUseSpecificProcedures enumeration member 3551
ibxeWindowsAPIError enumeration member 3551
ibxeXSQLDAIndexOutOfRange enumeration member 3551
ibxeXSQLDANameDoesNotExist enumeration member 3551
IChangeNotifier interface 2834
- about IChangeNotifier interface 2834
- Changed 2835
- IChangeNotifier Members 2834
- IChangeNotifier methods 2834

ICorbaObj interface 6215
- about ICorbaObj interface 6215
- CorbaObject 6216
- ICorbaObj Members 6215
- ICorbaObj methods 6215
- IsLocal 6216

ICustomHelpViewer interface 3215
- about ICustomHelpViewer interface 3215
- CanShowTableOfContents 3216
- GetHelpStrings 3217
- GetViewerName 3217
- ICustomHelpViewer Members 3216
- ICustomHelpViewer methods 3216
- NotifyID 3218
- ShowHelp 3218
- ShowTableOfContents 3218
- ShutDown 3218
- SoftShutDown 3219

UnderstandsKeyword 3219
IDataSetComponent interface 5644
about IDataSetComponent interface 5644
DataSet 5646
GetDataSet 5645
GetXMLBroker 5645
GetXMLDataSet 5646
IDataSetComponent Members 5645
IDataSetComponent methods 5645
IDataSetComponent properties 5646
XMLBroker 5647
XMLDataSet 5647
IDataSetField interface 5648
about IDataSetField interface 5648
FieldName 5649
GetFieldName 5648
GetRowSetFieldAttributes 5649
IDataSetField Members 5648
IDataSetField methods 5648
IDataSetField properties 5649
SetFieldName 5649
IDataSetFields interface 5650
about IDataSetFields interface 5650
GetFieldsList 5651
GetVisibleFields 5651
HasStatusField 5651
IDataSetFields Members 5650
IDataSetFields methods 5650
IDataSetFields properties 5652
IsFieldInUse 5651
ValidateDataSet 5651
VisibleFields 5652
IDesignerFileManager interface 3502
about IDesignerFileManager interface 3502
GetStream 3503
IDesignerFileManager Members 3502
IDesignerFileManager methods 3502
QualifyFileName 3503
IDesignerHook interface 2566
about IDesignerHook interface 2566
Form 2571
GetCustomForm 2567
GetIsControl 2568
GetRoot 2568
IDesignerHook Members 2567
IDesignerHook methods 2567
IDesignerHook properties 2571
IsControl 2571
IsDesignMsg 2568
PaintGrid 2569
PaintMenu 2569
SetCustomForm 2569
SetIsControl 2570
UniqueName 2570
ValidateRename 2570
ieDelphiException enumeration member 3870
ieInvalidOnErrorResult enumeration member 3870
ieInvalidOnStatusResult enumeration member 3870
ieInvalidOption enumeration member 3870
ieNoDestinationDirectory enumeration member 3870
ieNoOptionsSet enumeration member 3870
ieNosourceDirectory enumeration member 3870
ieNoUninstallFile enumeration member 3870
IEnumNames interface 5859
about IEnumNames interface 5859
Clone 5860
IEnumNames Members 5859
IEnumNames methods 5860
Next 5860
Reset 5860
Skip 5860
ieOptionNeedsClient enumeration member 3870
ieOptionNeedsServer enumeration member 3870
ieSuccess enumeration member 3870
IExtendedHelpViewer interface 3219
about IExtendedHelpViewer interface 3219
DisplayHelpByContext 3221
DisplayTopic 3221

IExtendedHelpViewer Members 3220
IExtendedHelpViewer methods 3220
UnderstandsContext 3221
UnderstandsTopic 3222
IExtendedHelpViewer2 interface 3222
about IExtendedHelpViewer2 interface 3222
DisplayIndex 3223
DisplaySearch 3223
DisplayTopic 3224
IExtendedHelpViewer2 Members 3222
IExtendedHelpViewer2 methods 3223
IFilteredHelpViewer interface 3224
about IFilteredHelpViewer interface 3224
GetFilter 3225
IFilteredHelpViewer Members 3225
IFilteredHelpViewer methods 3225
SetFilter 3226
IFormatColumn interface 5652
about IFormatColumn interface 5652
FormatColumnData 5653
FormatColumnHeading 5653
IFormatColumn Members 5652
IFormatColumn methods 5653
IGetAppDispatcher interface 3416
about IGetAppDispatcher interface 3416
GetAppDispatcher 3417
IGetAppDispatcher Members 3416
IGetAppDispatcher methods 3417
IGetContextProperties interface 5861
about IGetContextProperties interface 5861
Count 5861
EnumNames 5862
GetProperty 5862
IGetContextProperties Members 5861
IGetContextProperties methods 5861
IGetLocateFileService interface 3503
about IGetLocateFileService interface 3503
GetLocateFileService 3504
IGetLocateFileService Members 3503
IGetLocateFileService methods 3504
IGetProducerTemplate interface 3504
about IGetProducerTemplate interface 3504
GetProducerTemplateFileName 3505
GetProducerTemplateStream 3505
IGetProducerTemplate Members 3504
IGetProducerTemplate methods 3505
IGetWebAppServices interface 3417
about IGetWebAppServices interface 3417
GetWebAppServices 3418
IGetWebAppServices Members 3417
IGetWebAppServices methods 3418
IGetWebRequestHandler interface 3418
about IGetWebRequestHandler interface 3418
GetWebRequestHandler 3419
IGetWebRequestHandler Members 3418
IGetWebRequestHandler methods 3418
IgnoreChecksum enumeration member 4026
IgnoreChecksums enumeration member 4018
IgnoreLimbo enumeration member 4018
IHeadersSetter interface 4460
about IHeadersSetter interface 4460
IHeadersSetter Members 4460
IHeadersSetter methods 4460
SetHeadersInOut 4461
IHelpManager interface 3226
about IHelpManager interface 3226
GetHandle 3227
GetHelpFile 3227
IHelpManager Members 3226
IHelpManager methods 3227
Release 3227
IHelpSelector interface 3228
about IHelpSelector interface 3228
IHelpSelector Members 3228
IHelpSelector methods 3229
SelectKeyword 3229
TableOfContents 3229
IHelpSelector2 interface 3230

about IHelpSelector2 interface 3230
IHelpSelector2 Members 3230
IHelpSelector2 methods 3230
SelectContext 3230
IHelpSystem interface 3231
about IHelpSystem interface 3231
AssignHelpSelector 3232
Hook 3232
IHelpSystem Members 3231
IHelpSystem methods 3231
ShowContextHelp 3232
ShowHelp 3233
ShowTableOfContents 3233
ShowTopicHelp 3233
IHelpSystem2 interface 3234
about IHelpSystem2 interface 3234
IHelpSystem2 Members 3234
IHelpSystem2 methods 3234
UnderstandsKeyword 3235
IHelpSystem3 interface 3235
about IHelpSystem3 interface 3235
GetFilter 3236
IHelpSystem3 Members 3235
IHelpSystem3 methods 3236
SetFilter 3236
ShowIndex 3237
ShowSearch 3237
ShowTopicHelp 3237
IHelpSystemFlags interface 3238
about IHelpSystemFlags interface 3238
GetUseDefaultTopic 3239
IHelpSystemFlags Members 3238
IHelpSystemFlags methods 3239
SetUseDefaultTopic 3239
IHTMLField interface 5653
about IHTMLField interface 5653
GetHTMLControlName 5654
HTMLControlName 5655
IHTMLField Members 5654
IHTMLField methods 5654
IHTMLField properties 5654
IHTMLForm interface 5655
about IHTMLForm interface 5655
GetHTMLFormName 5656
GetHTMLFormVarName 5656
HTMLFormName 5657
HTMLFormVarName 5657
IHTMLForm Members 5655
IHTMLForm methods 5656
IHTMLForm properties 5657
IHtmlHelpTester interface 3246
about IHtmlHelpTester interface 3246
CanShowALink 3247
CanShowContext 3247
CanShowTopic 3247
ConvertWinHelp 3247
GetDefaultHelpFile 3248
GetHelpPath 3248
GetHelpStrings 3248
IHtmlHelpTester Members 3246
IHtmlHelpTester methods 3246
IIBEventNotifier interface 3761
about IIBEventNotifier interface 3761
GetAutoRegister 3762
IIBEventNotifier Members 3761
IIBEventNotifier methods 3762
RegisterEvents 3762
UnRegisterEvents 3762
IIBSQLMonitorHook interface 4109
about IIBSQLMonitorHook interface 4109
DBConnect 4110
DBDisconnect 4111
Enabled 4116
GetEnabled 4111
GetMonitorCount 4111
GetTraceFlags 4111
IIBSQLMonitorHook Members 4109
IIBSQLMonitorHook methods 4110

IIBSQLMonitorHook properties 4115
RegisterMonitor 4112
ReleaseMonitor 4112
SendMisc 4112
ServiceAttach 4112
ServiceDetach 4112
ServiceQuery 4113
ServiceStart 4113
SetEnabled 4113
SetTraceFlags 4113
SQLExecute 4114
SQLFetch 4114
SQLPrepare 4114
TraceFlags 4116
TRCommit 4114
TRCommitRetaining 4114
TRRollback 4115
TRRollbackRetaining 4115
TRStart 4115
UnregisterMonitor 4115
IInetXWebDataSet interface 5657
about IInetXWebDataSet interface 5657
AddOnDataChange 5659
FetchParams 5659
FieldCount 5659
Fields 5660
IInetXWebDataSet Members 5658
IInetXWebDataSet methods 5658
ParamCount 5660
Params 5660
RemoveOnDataChange 5661
ILocateFileService interface 3505
about ILocateFileService interface 3505
GetTemplateStream 3506
ILocateFileService Members 3506
ILocateFileService methods 3506
IMarshalInBuffer interface 6216
about IMarshalInBuffer interface 6216
Buffer 6217
GetAny 6218
GetChar 6218
GetDouble 6218
GetFloat 6218
GetLong 6218
GetObject 6219
GetShort 6219
GetText 6219
GetUnsignedChar 6219
GetUnsignedLong 6220
GetUnsignedShort 6220
GetWidechar 6220
GetWideText 6220
IMarshalInBuffer Members 6216
IMarshalInBuffer methods 6217
IMarshalOutBuffer interface 6220
about IMarshalOutBuffer interface 6220
Buffer 6222
IMarshalOutBuffer Members 6221
IMarshalOutBuffer methods 6221
PutAny 6222
PutChar 6222
PutDouble 6222
PutFloat 6222
PutLong 6223
PutObject 6223
PutShort 6223
PutText 6223
PutUnsignedChar 6224
PutUnsignedLong 6224
PutUnsignedShort 6224
PutWidechar 6224
PutWideText 6224
IMethodHandler interface 5893
about IMethodHandler interface 5893
Execute 5894
IMethodHandler Members 5893
IMethodHandler methods 5893
InstanceToVariant 5894

ImgList 4189
ImgList namespace 4189
Classes 4190
Structs, Records, Enums 4227
Types 4231
ImgList.TChangeLink 4190
ImgList.TChangeLink.Change 4191
ImgList.TChangeLink.Destroy 4191
ImgList.TChangeLink.OnChange 4192
ImgList.TChangeLink.Sender 4192
ImgList.TColorDepth 4227
ImgList.TColorDepth enumeration 4227
ImgList.TCustomImageList 4193
ImgList.TCustomImageList.Add 4197
ImgList.TCustomImageList.AddIcon 4197
ImgList.TCustomImageList.AddImage 4198
ImgList.TCustomImageList.AddImages 4198
ImgList.TCustomImageList.AddMasked 4199
ImgList.TCustomImageList.AllocBy 4217
ImgList.TCustomImageList.Assign 4199
ImgList.TCustomImageList.AssignTo 4200
ImgList.TCustomImageList.BkColor 4218
ImgList.TCustomImageList.BlendColor 4218
ImgList.TCustomImageList.Change 4200
ImgList.TCustomImageList.Clear 4200
ImgList.TCustomImageList.ColorDepth 4218
ImgList.TCustomImageList.Count 4219
ImgList.TCustomImageList.Create 4195
ImgList.TCustomImageList.CreateSize 4195
ImgList.TCustomImageList.DefineProperties 4201
ImgList.TCustomImageList.Delete 4201
ImgList.TCustomImageList.Destroy 4196
ImgList.TCustomImageList.DoDraw 4201
ImgList.TCustomImageList.Draw 4202
ImgList.TCustomImageList.DrawingStyle 4223
ImgList.TCustomImageList.DrawOverlay 4205
ImgList.TCustomImageList.FileLoad 4206
ImgList.TCustomImageList.GetBitmap 4206
ImgList.TCustomImageList.GetHotSpot 4207
ImgList.TCustomImageList.GetIcon 4207
ImgList.TCustomImageList.GetImageBitmap 4208
ImgList.TCustomImageList.GetImages 4208
ImgList.TCustomImageList.GetInstRes 4209
ImgList.TCustomImageList.GetMaskBitmap 4210
ImgList.TCustomImageList.GetResource 4210
ImgList.TCustomImageList.Handle 4224
ImgList.TCustomImageList.HandleAllocated 4211
ImgList.TCustomImageList.HandleNeeded 4211
ImgList.TCustomImageList.Height 4224
ImgList.TCustomImageList.ImageType 4224
ImgList.TCustomImageList.Initialize 4211
ImgList.TCustomImageList.Insert 4212
ImgList.TCustomImageList.InsertIcon 4212
ImgList.TCustomImageList.InsertMasked 4212
ImgList.TCustomImageList.Masked 4225
ImgList.TCustomImageList.Move 4213
ImgList.TCustomImageList.OnChange 4226
ImgList.TCustomImageList.Overlay 4213
ImgList.TCustomImageList.ReadData 4213
ImgList.TCustomImageList.RegisterChanges 4214
ImgList.TCustomImageList.Replace 4214
ImgList.TCustomImageList.ReplaceIcon 4214
ImgList.TCustomImageList.ReplaceMasked 4215
ImgList.TCustomImageList.ResInstLoad 4215
ImgList.TCustomImageList.ResourceLoad 4216
ImgList.TCustomImageList.SetSize 4216
ImgList.TCustomImageList.ShareImages 4225
ImgList.TCustomImageList.UnRegisterChanges 4216
ImgList.TCustomImageList.Width 4225
ImgList.TCustomImageList.WriteData 4217
ImgList.TDrawingStyle 4228
ImgList.TDrawingStyle enumeration 4228
ImgList.TImageIndex 4231
ImgList.TImageIndex type 4231
ImgList.TImageListHandle 4231
ImgList.TImageListHandle type 4231
ImgList.TImageType 4229
ImgList.TImageType enumeration 4229

ImgList.TLoadResource 4229
ImgList.TLoadResource enumeration 4229
ImgList.TLoadResources 4232
ImgList.TLoadResources type 4232
ImgList.TOverlay 4232
ImgList.TOverlay type 4232
ImgList.TResType 4230
ImgList.TResType enumeration 4230
IMouse 4233
IMouse namespace 4233
Classes 4235
Constants 4252
Functions 4247
Structs, Records, Enums 4248
Types 4250
IMouse.crPanAll 4252
IMouse.crPanAll constant 4252
IMouse.crPanDown 4252
IMouse.crPanDown constant 4252
IMouse.crPanDownLeft 4253
IMouse.crPanDownLeft constant 4253
IMouse.crPanDownRight 4253
IMouse.crPanDownRight constant 4253
IMouse.crPanLeft 4253
IMouse.crPanLeft constant 4253
IMouse.crPanLeftRight 4254
IMouse.crPanLeftRight constant 4254
IMouse.crPanRight 4254
IMouse.crPanRight constant 4254
IMouse.crPanUp 4254
IMouse.crPanUp constant 4254
IMouse.crPanUpDown 4255
IMouse.crPanUpDown constant 4255
IMouse.crPanUpLeft 4255
IMouse.crPanUpLeft constant 4255
IMouse.crPanUpRight 4255
IMouse.crPanUpRight constant 4255
IMouse.StartPan 4247
IMouse.StartPan function 4247
IMouse.TPanDirection 4248
IMouse.TPanDirection enumeration 4248
IMouse.TPanDirections 4250
IMouse.TPanDirections type 4250
IMouse.TPanningWindow 4235
IMouse.TPanningWindow.Create 4236
IMouse.TPanningWindow.CreateParams 4237
IMouse.TPanningWindow.Destroy 4236
IMouse.TPanningWindow.DoUpdate 4238
IMouse.TPanningWindow.GetIsPanning 4238
IMouse.TPanningWindow.GetPanOptions 4238
IMouse.TPanningWindow.IsPanning 4245
IMouse.TPanningWindow.MouseMove 4239
IMouse.TPanningWindow.OnUpdate 4246
IMouse.TPanningWindow.Paint 4239
IMouse.TPanningWindow.PanControl 4245
IMouse.TPanningWindow.PanInterval 4246
IMouse.TPanningWindow.PanOptions 4246
IMouse.TPanningWindow.SetPanCursor 4240
IMouse.TPanningWindow.StartPanning 4240
IMouse.TPanningWindow.StopPanning 4240
IMouse.TPanningWindow.UpdateScroll 4240
IMouse.TPanningWindow.WndProc 4241
IMouse.TPanOption 4248
IMouse.TPanOption enumeration 4248
IMouse.TPanOptions 4250
IMouse.TPanOptions type 4250
IMouse.TUpdateEvent 4250
IMouse.TUpdateEvent type 4250
IMultiModuleSupport interface 3419
about IMultiModuleSupport interface 3419
FinishContext 3420
IMultiModuleSupport Members 3419
IMultiModuleSupport methods 3419
InitContext 3420
InitModule 3420
IndexPages enumeration member 4023
IniFiles 4257
IniFiles namespace 4257

Classes 4258
Structs, Records, Enums 4328
Types 4329
IniFiles.EIniFileException 4258
IniFiles.PHashItem 4329
IniFiles.PHashItem type 4329
IniFiles.PPHashItem 4329
IniFiles.PPHashItem type 4329
IniFiles.TCustomIniFile 4258
IniFiles.TCustomIniFile.Create 4262
IniFiles.TCustomIniFile.DeleteKey 4263
IniFiles.TCustomIniFile.EraseSection 4264
IniFiles.TCustomIniFile.FileName 4291
IniFiles.TCustomIniFile.ReadBinaryStream 4264
IniFiles.TCustomIniFile.ReadBool 4265
IniFiles.TCustomIniFile.ReadDate 4268
IniFiles.TCustomIniFile.ReadDateTime 4268
IniFiles.TCustomIniFile.ReadFloat 4269
IniFiles.TCustomIniFile.ReadInteger 4269
IniFiles.TCustomIniFile.ReadSection 4272
IniFiles.TCustomIniFile.ReadSections 4273
IniFiles.TCustomIniFile.ReadSectionValues 4274
IniFiles.TCustomIniFile.ReadString 4274
IniFiles.TCustomIniFile.ReadTime 4277
IniFiles.TCustomIniFile.SectionExists 4278
IniFiles.TCustomIniFile.UpdateFile 4278
IniFiles.TCustomIniFile.ValueExists 4278
IniFiles.TCustomIniFile.WriteBinaryStream 4279
IniFiles.TCustomIniFile.WriteBool 4279
IniFiles.TCustomIniFile.WriteDate 4282
IniFiles.TCustomIniFile.WriteDateTime 4283
IniFiles.TCustomIniFile.WriteFloat 4283
IniFiles.TCustomIniFile.WriteInteger 4284
IniFiles.TCustomIniFile.WriteString 4287
IniFiles.TCustomIniFile.WriteTime 4290
IniFiles.THashedStringList 4291
IniFiles.THashedStringList.Changed 4292
IniFiles.THashedStringList.Destroy 4292
IniFiles.THashedStringList.IndexOf 4293
IniFiles.THashedStringList.IndexOfName 4293
IniFiles.THashItem 4328
IniFiles.THashItem record 4328
IniFiles.TIniFile 4294
IniFiles.TIniFile.DeleteKey 4299
IniFiles.TIniFile.Destroy 4298
IniFiles.TIniFile.EraseSection 4299
IniFiles.TIniFile.ReadSection 4299
IniFiles.TIniFile.ReadSections 4300
IniFiles.TIniFile.ReadSectionValues 4300
IniFiles.TIniFile.ReadString 4301
IniFiles.TIniFile.UpdateFile 4304
IniFiles.TIniFile.WriteString 4304
IniFiles.TMemIniFile 4307
IniFiles.TMemIniFile.CaseSensitive 4324
IniFiles.TMemIniFile.Clear 4313
IniFiles.TMemIniFile.Create 4311
IniFiles.TMemIniFile.DeleteKey 4313
IniFiles.TMemIniFile.Destroy 4312
IniFiles.TMemIniFile.Encoding 4324
IniFiles.TMemIniFile.EraseSection 4314
IniFiles.TMemIniFile.GetStrings 4314
IniFiles.TMemIniFile.ReadSection 4314
IniFiles.TMemIniFile.ReadSections 4315
IniFiles.TMemIniFile.ReadSectionValues 4315
IniFiles.TMemIniFile.ReadString 4316
IniFiles.TMemIniFile.Rename 4319
IniFiles.TMemIniFile.SetStrings 4319
IniFiles.TMemIniFile.UpdateFile 4320
IniFiles.TMemIniFile.WriteString 4320
IniFiles.TStringHash 4324
IniFiles.TStringHash.Add 4326
IniFiles.TStringHash.Clear 4326
IniFiles.TStringHash.Create 4325
IniFiles.TStringHash.Destroy 4325
IniFiles.TStringHash.Find 4326
IniFiles.TStringHash.HashOf 4327
IniFiles.TStringHash.Modify 4327
IniFiles.TStringHash.Remove 4327

IniFiles.TStringHash.ValueOf 4327
IntfInfo 4331
IntfInfo namespace 4331
Classes 4333
Constants 4345
Functions 4334
Structs, Records, Enums 4339
Types 4342
IntfInfo.CallingConventionName 4345
IntfInfo.CallingConventionName constant 4345
IntfInfo.EInterfaceRTTIException 4333
IntfInfo.GetClsMemberTypeInfo 4334
IntfInfo.GetClsMemberTypeInfo function 4334
IntfInfo.GetDynArrayElTypeInfo 4334
IntfInfo.GetDynArrayElTypeInfo function 4334
IntfInfo.GetDynArrayNextInfo 4335
IntfInfo.GetDynArrayNextInfo function 4335
IntfInfo.GetDynArrayNextInfo2 4335
IntfInfo.GetDynArrayNextInfo2 function 4335
IntfInfo.GetIntfMetaData 4335, 4336
IntfInfo.GetIntfMetaData function 4335, 4336
IntfInfo.GetMethNum 4336
IntfInfo.GetMethNum function 4336
IntfInfo.GetPropListFlat 4336
IntfInfo.GetPropListFlat function 4336
IntfInfo.IsStoredPropConst 4337
IntfInfo.IsStoredPropConst function 4337
IntfInfo.OtherTypeName 4337
IntfInfo.OtherTypeName function 4337
IntfInfo.PIntfMetaData 4342
IntfInfo.PIntfMetaData type 4342
IntfInfo.PIntfMethEntry 4342
IntfInfo.PIntfMethEntry type 4342
IntfInfo.PIntfParamEntry 4343
IntfInfo.PIntfParamEntry type 4343
IntfInfo.SameTypeInfo 4337
IntfInfo.SameTypeInfo function 4337
IntfInfo.TCallConv 4339
IntfInfo.TCallConv enumeration 4339
IntfInfo.TFillMethodArrayOpt 4339
IntfInfo.TFillMethodArrayOpt enumeration 4339
IntfInfo.TIntfMetaData 4340
IntfInfo.TIntfMetaData record 4340
IntfInfo.TIntfMethEntry 4340
IntfInfo.TIntfMethEntry record 4340
IntfInfo.TIntfMethEntryArray 4343
IntfInfo.TIntfMethEntryArray type 4343
IntfInfo.TIntfParamEntry 4341
IntfInfo.TIntfParamEntry record 4341
IntfInfo.TIntfParamEntryArray 4343
IntfInfo.TIntfParamEntryArray type 4343
IntfInfo.TPIntfMethEntryArray 4344
IntfInfo.TPIntfMethEntryArray type 4344
IntfInfo.TypeInfoNames 4345
IntfInfo.TypeInfoNames constant 4345
IntfInfo.TypeNamesMatch 4338
IntfInfo.TypeNamesMatch function 4338
InvokeRegistry 4347
InvokeRegistry namespace 4347
Classes 4350
Constants 4497
Functions 4470
Interfaces 4460
Structs, Records, Enums 4475
Types 4487
Variables 4496
InvokeRegistry.AppNameSpacePrefix 4496
InvokeRegistry.AppNameSpacePrefix variable 4496
InvokeRegistry.AS_ATTRIBUTE 4497
InvokeRegistry.AS_ATTRIBUTE constant 4497
InvokeRegistry.eHeaderMethodType 4475
InvokeRegistry.eHeaderMethodType enumeration 4475
InvokeRegistry.ERemotableException 4350
InvokeRegistry.ERemotableException.Create 4351
InvokeRegistry.ERemotableException.FaultActor 4352
InvokeRegistry.ERemotableException.FaultCode 4352
InvokeRegistry.ERemotableException.FaultDetail 4353
InvokeRegistry.ERemotableExceptionClass 4487

InvokeRegistry.ERemotableExceptionClass type 4487
InvokeRegistry.ETypeRegistryException 4353
InvokeRegistry.ExtNameMapItem 4476
InvokeRegistry.ExtNameMapItem record 4476
InvokeRegistry.GetRemotableDataContext 4470
InvokeRegistry.GetRemotableDataContext function 4470
InvokeRegistry.IHeadersSetter 4460
InvokeRegistry.IHeadersSetter.SetHeadersInOut 4461
InvokeRegistry.IntfExceptionItem 4476
InvokeRegistry.IntfExceptionItem record 4476
InvokeRegistry.IntfHeaderItem 4477
InvokeRegistry.IntfHeaderItem record 4477
InvokeRegistry.InvRegClassEntry 4478
InvokeRegistry.InvRegClassEntry record 4478
InvokeRegistry.InvRegIntfEntry 4478
InvokeRegistry.InvRegIntfEntry record 4478
InvokeRegistry.InvRegistry 4471
InvokeRegistry.InvRegistry function 4471
InvokeRegistry.InvString 4488
InvokeRegistry.InvString type 4488
InvokeRegistry.InvStringDynArray 4488
InvokeRegistry.InvStringDynArray type 4488
InvokeRegistry.IObjConverter 4461
InvokeRegistry.IObjConverter.AddAttachment 4462
InvokeRegistry.IObjConverter.FindAttachment 4463
InvokeRegistry.IObjConverter.InitObjectFromSOAP 4463
InvokeRegistry.IObjConverter.ObjInstanceToSOAP 4463
InvokeRegistry.ISOAPHeaders 4464
InvokeRegistry.ISOAPHeaders.Get 4466
InvokeRegistry.ISOAPHeaders.GetOwnsSentHeaders 4466
InvokeRegistry.ISOAPHeaders.OwnsSentHeaders 4468
InvokeRegistry.ISOAPHeaders.Send 4467
InvokeRegistry.ISOAPHeaders.SendAt 4467
InvokeRegistry.ISOAPHeaders.SendCount 4468
InvokeRegistry.ISOAPHeaders.SetOwnsSentHeaders 4468
InvokeRegistry.MethParamNameMapItem 4480
InvokeRegistry.MethParamNameMapItem record 4480
InvokeRegistry.ObjectConvertOptions 4480
InvokeRegistry.ObjectConvertOptions enumeration 4480
InvokeRegistry.PTRemotable 4488
InvokeRegistry.PTRemotable type 4488
InvokeRegistry.RemClassRegistry 4471
InvokeRegistry.RemClassRegistry function 4471
InvokeRegistry.RemTypeRegistry 4472
InvokeRegistry.RemTypeRegistry function 4472
InvokeRegistry.SerializationOptions 4481
InvokeRegistry.SerializationOptions enumeration 4481
InvokeRegistry.SetRemotableDataContext 4473
InvokeRegistry.SetRemotableDataContext function 4473
InvokeRegistry.SubstituteStrings 4474
InvokeRegistry.SubstituteStrings function 4474
InvokeRegistry.TCreateInstanceProc 4489
InvokeRegistry.TCreateInstanceProc type 4489
InvokeRegistry.TDataContext 4354
InvokeRegistry.TDataContext.AddDynArrayToClear 4358
InvokeRegistry.TDataContext.AddObjectToDestroy 4359
InvokeRegistry.TDataContext.AddStrToClear 4359
InvokeRegistry.TDataContext.AddUStrToClear 4360
InvokeRegistry.TDataContext.AddVariantToClear 4360
InvokeRegistry.TDataContext.AddWStrToClear 4361
InvokeRegistry.TDataContext.AllocData 4361
InvokeRegistry.TDataContext.Create 4355
InvokeRegistry.TDataContext.Data 4356
InvokeRegistry.TDataContext.DataOffset 4356
InvokeRegistry.TDataContext.DataP 4357
InvokeRegistry.TDataContext.Destroy 4355
InvokeRegistry.TDataContext.DynArrayToClear 4357
InvokeRegistry.TDataContext.FObjsToDestroy 4357
InvokeRegistry.TDataContext.GetDataPointer 4362
InvokeRegistry.TDataContext.RemoveObjectToDestroy 4362
InvokeRegistry.TDataContext.SetDataPointer 4362
InvokeRegistry.TDataContext.StrToClear 4357
InvokeRegistry.TDataContext.UStrToClear 4357
InvokeRegistry.TDataContext.VarToClear 4358
InvokeRegistry.TDataContext.WStrToClear 4358
InvokeRegistry.TDynToClear 4482
InvokeRegistry.TDynToClear record 4482
InvokeRegistry.TExceptionItemArray 4489

InvokeRegistry.TExceptionItemArray type 4489
InvokeRegistry.THeaderItemArray 4489
InvokeRegistry.THeaderItemArray type 4489
InvokeRegistry.THeaderList 4363
InvokeRegistry.THeaderList.Add 4365
InvokeRegistry.THeaderList.Clear 4366
InvokeRegistry.THeaderList.Count 4368
InvokeRegistry.THeaderList.Create 4364
InvokeRegistry.THeaderList.Destroy 4364
InvokeRegistry.THeaderList.Extract 4366
InvokeRegistry.THeaderList.GetCount 4366
InvokeRegistry.THeaderList.GetHeader 4367
InvokeRegistry.THeaderList.GetOwnsObjects 4367
InvokeRegistry.THeaderList.Headers 4368
InvokeRegistry.THeaderList.IndexOf 4367
InvokeRegistry.THeaderList.OwnsObjects 4369
InvokeRegistry.THeaderList.SetOwnsObjects 4367
InvokeRegistry.THeaderMethodTypeArray 4490
InvokeRegistry.THeaderMethodTypeArray type 4490
InvokeRegistry.TIntfInvokeOption 4483
InvokeRegistry.TIntfInvokeOption enumeration 4483
InvokeRegistry.TIntfInvokeOptions 4490
InvokeRegistry.TIntfInvokeOptions type 4490
InvokeRegistry.TInvContext 4369
InvokeRegistry.TInvContext.AllocServerData 4371
InvokeRegistry.TInvContext.GetParamPointer 4371
InvokeRegistry.TInvContext.GetResultPointer 4372
InvokeRegistry.TInvContext.SetMethodInfo 4372
InvokeRegistry.TInvContext.SetParamPointer 4373
InvokeRegistry.TInvContext.SetResultPointer 4373
InvokeRegistry.TInvokableClass 4373
InvokeRegistry.TInvokableClass._AddRef 4376
InvokeRegistry.TInvokableClass._Release 4377
InvokeRegistry.TInvokableClass.AfterConstruction 4377
InvokeRegistry.TInvokableClass.BeforeDestruction 4377
InvokeRegistry.TInvokableClass.Create 4375
InvokeRegistry.TInvokableClass.FRefCount 4375
InvokeRegistry.TInvokableClass.FSOAPHeaders 4376
InvokeRegistry.TInvokableClass.NewInstance 4378
InvokeRegistry.TInvokableClass.QueryInterface 4378
InvokeRegistry.TInvokableClass.RefCount 4379
InvokeRegistry.TInvokableClassClass 4490
InvokeRegistry.TInvokableClassClass type 4490
InvokeRegistry.TInvokableClassRegistry 4379
InvokeRegistry.TInvokableClassRegistry.Create 4381
InvokeRegistry.TInvokableClassRegistry.Destroy 4382
InvokeRegistry.TInvokableClassRegistry.GetActionURIOfIID 4383
InvokeRegistry.TInvokableClassRegistry.GetActionURIOfInfo 4384
InvokeRegistry.TInvokableClassRegistry.GetClassFromIntfInfo 4384
InvokeRegistry.TInvokableClassRegistry.GetExceptionInfoForInterface 4385
InvokeRegistry.TInvokableClassRegistry.GetHeaderClass 4385
InvokeRegistry.TInvokableClassRegistry.GetHeaderInfoForInterface 4386
InvokeRegistry.TInvokableClassRegistry.GetHeaderName 4386
InvokeRegistry.TInvokableClassRegistry.GetHeaderNamespace 4387
InvokeRegistry.TInvokableClassRegistry.GetInfoForURI 4388
InvokeRegistry.TInvokableClassRegistry.GetInterfaceCount 4388
InvokeRegistry.TInvokableClassRegistry.GetInterfaceExternalName 4389
InvokeRegistry.TInvokableClassRegistry.GetInterfaceInfoFromName 4390
InvokeRegistry.TInvokableClassRegistry.GetInterfaceTypeInfo 4390
InvokeRegistry.TInvokableClassRegistry.GetIntfInvokeOptions 4391
InvokeRegistry.TInvokableClassRegistry.GetInvokableObjectFromClass 4391
InvokeRegistry.TInvokableClassRegistry.GetMethExternalName 4392
InvokeRegistry.TInvokableClassRegistry.GetMethInternalName 4392
InvokeRegistry.TInvokableClassRegistry.GetNamespaceByGUID

4393
InvokeRegistry.TInvokableClassRegistry.GetParamExternalName 4393
InvokeRegistry.TInvokableClassRegistry.GetParamInternalName 4393
InvokeRegistry.TInvokableClassRegistry.GetRegInterfaceEntry 4394
InvokeRegistry.TInvokableClassRegistry.GetRequestHeaderInfoForInterface 4394
InvokeRegistry.TInvokableClassRegistry.GetResponseHeaderInfoForInterface 4395
InvokeRegistry.TInvokableClassRegistry.GetReturnParamNames 4396
InvokeRegistry.TInvokableClassRegistry.GetUDDIInfo 4396
InvokeRegistry.TInvokableClassRegistry.GetWSDLEncoding 4397
InvokeRegistry.TInvokableClassRegistry.HasRegInterfaceImpl 4398
InvokeRegistry.TInvokableClassRegistry.RegisterAllSOAPActions 4398
InvokeRegistry.TInvokableClassRegistry.RegisterDefaultSOAPAction 4399
InvokeRegistry.TInvokableClassRegistry.RegisterException 4399
InvokeRegistry.TInvokableClassRegistry.RegisterExceptionMethod 4400
InvokeRegistry.TInvokableClassRegistry.RegisterExternalMethName 4400
InvokeRegistry.TInvokableClassRegistry.RegisterExternalParamName 4401
InvokeRegistry.TInvokableClassRegistry.RegisterHeaderClass 4402
InvokeRegistry.TInvokableClassRegistry.RegisterHeaderMethod 4403
InvokeRegistry.TInvokableClassRegistry.RegisterInterface 4404
InvokeRegistry.TInvokableClassRegistry.RegisterInvokableClass 4405
InvokeRegistry.TInvokableClassRegistry.RegisterInvokeOptions 4406
InvokeRegistry.TInvokableClassRegistry.RegisterReturnParamNames 4406
InvokeRegistry.TInvokableClassRegistry.RegisterUDDIInfo 4407
InvokeRegistry.TInvokableClassRegistry.UnRegisterInterface 4407
InvokeRegistry.TInvokableClassRegistry.UnRegisterInvokableClass 4408
InvokeRegistry.TInvRegIntfEntryArray 4491
InvokeRegistry.TInvRegIntfEntryArray type 4491
InvokeRegistry.TObjectConvertOptions 4491
InvokeRegistry.TObjectConvertOptions type 4491
InvokeRegistry.TObjMultiOptions 4484
InvokeRegistry.TObjMultiOptions enumeration 4484
InvokeRegistry.TPascalRemotableClassRegistry 4491
InvokeRegistry.TPascalRemotableClassRegistry type 4491
InvokeRegistry.TPascalRemotableTypeRegistry 4492
InvokeRegistry.TPascalRemotableTypeRegistry type 4492
InvokeRegistry.TRemHeaderEntry 4485
InvokeRegistry.TRemHeaderEntry record 4485
InvokeRegistry.TRemotable 4408
InvokeRegistry.TRemotable.Create 4410
InvokeRegistry.TRemotable.DataContext 4413
InvokeRegistry.TRemotable.Destroy 4410
InvokeRegistry.TRemotable.FSerializationOptions 4411
InvokeRegistry.TRemotable.ObjectToSOAP 4411
InvokeRegistry.TRemotable.SerializationOptions 4413
InvokeRegistry.TRemotable.SOAPToObject 4412
InvokeRegistry.TRemotableClass 4492
InvokeRegistry.TRemotableClass type 4492
InvokeRegistry.TRemotableClassRegistry 4492
InvokeRegistry.TRemotableClassRegistry type 4492
InvokeRegistry.TRemotableTypeRegistry 4414
InvokeRegistry.TRemotableTypeRegistry.AutoRegisterNativeTypes 4431
InvokeRegistry.TRemotableTypeRegistry.ClassOptions 4417
InvokeRegistry.TRemotableTypeRegistry.ClassToURI 4418
InvokeRegistry.TRemotableTypeRegistry.Create 4415
InvokeRegistry.TRemotableTypeRegistry.Destroy 4416

InvokeRegistry.TRemotableTypeRegistry.GetExternalPropName 4418
InvokeRegistry.TRemotableTypeRegistry.GetInternalPropName 4419
InvokeRegistry.TRemotableTypeRegistry.GetURICount 4419
InvokeRegistry.TRemotableTypeRegistry.GetURIMap 4420
InvokeRegistry.TRemotableTypeRegistry.GetVarTypeFromXSD 4421
InvokeRegistry.TRemotableTypeRegistry.GetXSDInfoForClass 4421
InvokeRegistry.TRemotableTypeRegistry.InfoToURI 4421
InvokeRegistry.TRemotableTypeRegistry.IsClassScalar 4422
InvokeRegistry.TRemotableTypeRegistry.Lock 4422
InvokeRegistry.TRemotableTypeRegistry.RegisterExternalPropName 4423
InvokeRegistry.TRemotableTypeRegistry.RegisterHolderClsMember 4423
InvokeRegistry.TRemotableTypeRegistry.RegisterSerializeOptions 4424
InvokeRegistry.TRemotableTypeRegistry.RegisterXSClass 4425
InvokeRegistry.TRemotableTypeRegistry.RegisterXSInfo 4426
InvokeRegistry.TRemotableTypeRegistry.SerializeOptions 4427
InvokeRegistry.TRemotableTypeRegistry.TypeInfoToXSD 4427
InvokeRegistry.TRemotableTypeRegistry.UnLock 4428
InvokeRegistry.TRemotableTypeRegistry.UnRegisterXSClass 4428
InvokeRegistry.TRemotableTypeRegistry.UnRegisterXSInfo 4429
InvokeRegistry.TRemotableTypeRegistry.URIToClass 4429
InvokeRegistry.TRemotableTypeRegistry.URIToInfo 4430
InvokeRegistry.TRemotableTypeRegistry.VariantToInfo 4430
InvokeRegistry.TRemotableTypeRegistry.XSDToTypeInfo 4431
InvokeRegistry.TRemotableXS 4432
InvokeRegistry.TRemotableXS.NativeToXS 4433
InvokeRegistry.TRemotableXS.ObjectToSOAP 4434
InvokeRegistry.TRemotableXS.SOAPToObject 4434
InvokeRegistry.TRemotableXS.XSToNative 4435
InvokeRegistry.TRemotableXSClass 4493
InvokeRegistry.TRemotableXSClass type 4493
InvokeRegistry.TRemRegEntry 4485
InvokeRegistry.TRemRegEntry record 4485
InvokeRegistry.TRemRegEntryArray 4493
InvokeRegistry.TRemRegEntryArray type 4493
InvokeRegistry.TRequiredArray 4493
InvokeRegistry.TRequiredArray type 4493
InvokeRegistry.TSerializationOptions 4494
InvokeRegistry.TSerializationOptions type 4494
InvokeRegistry.TSOAPAttachment 4435
InvokeRegistry.TSOAPAttachment.CacheFile 4444
InvokeRegistry.TSOAPAttachment.CacheFilePersist 4445
InvokeRegistry.TSOAPAttachment.ContentType 4445
InvokeRegistry.TSOAPAttachment.Create 4437
InvokeRegistry.TSOAPAttachment.Destroy 4438
InvokeRegistry.TSOAPAttachment.Encoding 4446
InvokeRegistry.TSOAPAttachment.GetSourceStream 4439
InvokeRegistry.TSOAPAttachment.Headers 4446
InvokeRegistry.TSOAPAttachment.Init 4439
InvokeRegistry.TSOAPAttachment.InternalSetCacheFile 4440
InvokeRegistry.TSOAPAttachment.InternalSetSourceStream 4440
InvokeRegistry.TSOAPAttachment.ObjectToSOAP 4440
InvokeRegistry.TSOAPAttachment.Ownership 4447
InvokeRegistry.TSOAPAttachment.SaveToFile 4441
InvokeRegistry.TSOAPAttachment.SaveToStream 4441
InvokeRegistry.TSOAPAttachment.SetOwnership 4442
InvokeRegistry.TSOAPAttachment.SetSourceFile 4442
InvokeRegistry.TSOAPAttachment.SetSourceStream 4443
InvokeRegistry.TSOAPAttachment.SetSourceString 4443
InvokeRegistry.TSOAPAttachment.SOAPToObject 4443
InvokeRegistry.TSOAPAttachment.SourceStream 4447
InvokeRegistry.TSOAPAttachment.SourceString 4448
InvokeRegistry.TSOAPAttachmentClass 4494
InvokeRegistry.TSOAPAttachmentClass type 4494
InvokeRegistry.TSOAPHeader 4448
InvokeRegistry.TSOAPHeader.Actor 4451
InvokeRegistry.TSOAPHeader.MustUnderstand 4452
InvokeRegistry.TSOAPHeader.ObjectToSOAP 4450
InvokeRegistry.TSOAPHeader.SOAPToObject 4451
InvokeRegistry.TSOAPHeaderClass 4494
InvokeRegistry.TSOAPHeaderClass type 4494

InvokeRegistry.TSOAPHeaders 4452
InvokeRegistry.TSOAPHeaders.Get 4454
InvokeRegistry.TSOAPHeaders.GetOwnsSentHeaders 4455
InvokeRegistry.TSOAPHeaders.Send 4455
InvokeRegistry.TSOAPHeaders.SendAt 4455
InvokeRegistry.TSOAPHeaders.SendCount 4456
InvokeRegistry.TSOAPHeaders.SetOwnsSentHeaders 4456
InvokeRegistry.TSOAPHeadersBase 4457
InvokeRegistry.TSOAPHeadersBase.FHeadersInbound 4458
InvokeRegistry.TSOAPHeadersBase.FHeadersOutBound 4458
InvokeRegistry.TSOAPHeadersBase.SetHeadersInOut 4458
InvokeRegistry.XMLBase64Types 4497
InvokeRegistry.XMLBase64Types constant 4497
InvokeRegistry.XMLSchemaInstNamepspaces 4498
InvokeRegistry.XMLSchemaInstNamepspaces constant 4498
InvokeRegistry.XMLSchemaNamepspaces 4498
InvokeRegistry.XMLSchemaNamepspaces constant 4498
IObjConverter interface 4461
about IObjConverter interface 4461
AddAttachment 4462
FindAttachment 4463
InitObjectFromSOAP 4463
IObjConverter Members 4462
IObjConverter methods 4462
ObjInstanceToSOAP 4463
IObjectContext interface 5862
about IObjectContext interface 5862
CreateInstance 5863
DisableCommit 5863
EnableCommit 5864
IObjectContext Members 5862
IObjectContext methods 5863
IsCallerInRole 5864
IsInTransaction 5864
IsSecurityEnabled 5864
SetAbort 5864
SetComplete 5865
IObjectContextActivity interface 5865
about IObjectContextActivity interface 5865
GetActivityId 5866
IObjectContextActivity Members 5865
IObjectContextActivity methods 5865
IObjectControl interface 5866
about IObjectControl interface 5866
Activate 5867
CanBePooled 5867
Deactivate 5867
IObjectControl Members 5866
IObjectControl methods 5866
ioDefault enumeration member 4483
ioDocument enumeration member 4483
ioHasAllSOAPActions enumeration member 4483
ioHasDefaultSOAPAction enumeration member 4483
ioHasNamespace enumeration member 4483
ioHasReturnParamNames enumeration member 4483
ioHasUDDIInfo enumeration member 4483
ioIsAppServerSOAP enumeration member 4483
IOleForm interface 2572
about IOleForm interface 2572
IOleForm Members 2572
IOleForm methods 2572
OnDestroy 2572
OnResize 2573
ioLiteral enumeration member 4483
IOPConvert interface 6152
about IOPConvert interface 6152
Attachments 6160
Encoding 6160
GetAttachments 6154
GetEncoding 6154
GetOptions 6154
GetTempDir 6154
InvContextToMsg 6155
IOPConvert Members 6153
IOPConvert methods 6153
IOPConvert properties 6160
MakeFault 6155
MakeResponse 6156

MsgToInvContext 6157
MsgToInvContext method 6156
Options 6161
ProcessResponse 6158
ProcessResponse method 6157
SetAttachments 6158
SetEncoding 6159
SetOptions 6159
SetTempDir 6159
TempDir 6161
IORB interface 6225
about IORB interface 6225
BOAInit 6226
CreateAliasTC 6226
CreateArrayTC 6226
CreateObjRefTC 6227
CreateSequenceTC 6227
CreateStructTC 6227
CreateTC 6227
DispatchStruct 6227
FindRepositoryTC 6228
IORB Members 6225
IORB methods 6225
MakeAny 6228
ObjectToString 6228
Orb 6228
Shutdown 6229
StringToObject 6229
IProduceContent interface 3420
about IProduceContent interface 3420
IProduceContent Members 3421
IProduceContent methods 3421
ProduceContent 3421
IProduceContentFrom interface 3422
about IProduceContentFrom interface 3422
IProduceContentFrom Members 3422
IProduceContentFrom methods 3422
ProduceContentFromStream 3422
ProduceContentFromString 3423
IProviderContainer interface 6580
about IProviderContainer interface 6580
IProviderContainer Members 6580
IProviderContainer methods 6580
RegisterProvider 6581
UnRegisterProvider 6581
IQueryField interface 5661
about IQueryField interface 5661
GetParamName 5662
GetText 5662
IQueryField Members 5661
IQueryField methods 5662
IQueryField properties 5663
ParamName 5663
SetParamName 5662
SetText 5663
Text 5663
IQueryFields interface 5664
about IQueryFields interface 5664
GetParamsList 5664
IQueryFields Members 5664
IQueryFields methods 5664
IsParamInUse 5665
ValidateDataSet 5665
IQueryForm interface 5665
about IQueryForm interface 5665
IRestoreDefaults interface 5665
about IRestoreDefaults interface 5665
IRestoreDefaults Members 5666
IRestoreDefaults methods 5666
RestoreDefaults 5666
IRibbon interface 6931
about IRibbon interface 6931
GetRibbon 6932
IRibbon Members 6931
IRibbon methods 6932
IRibbon properties 6932
Ribbon 6932
SetRibbon 6932

IRibbonBarKeyTip interface 7272
about IRibbonBarKeyTip interface 7272
BarHandleKeyTips 7273
BarHideKeyTips 7273
BarShowKeyTips 7274
IRibbonBarKeyTip Members 7272
IRibbonBarKeyTip methods 7273
IRibbonContainedControl interface 7274
about IRibbonContainedControl interface 7274
ContainedControl 7275
GetContainedControl 7275
IRibbonContainedControl Members 7274
IRibbonContainedControl methods 7275
IRibbonContainedControl properties 7275
SetContainedControl 7275
IRibbonControl interface 7276
about IRibbonControl interface 7276
CanAddToQuickAccessToolbar 7276
IRibbonControl Members 7276
IRibbonControl methods 7276
IRibbonGroup interface 6933
about IRibbonGroup interface 6933
AlignGroupControls 6934
CreateControl 6934
IRibbonGroup Members 6933
IRibbonGroup methods 6933
StyleChanged 6934
IRibbonGroupDropdown interface 7277
about IRibbonGroupDropdown interface 7277
CloseUp 7277
GetParentControl 7278
GetTopPosition 7278
IRibbonGroupDropdown Members 7277
IRibbonGroupDropdown methods 7277
IRibbonKeyTip interface 7278
about IRibbonKeyTip interface 7278
GetKeyTip 7279
HandleKeyTip 7280
HideKeyTip 7280
IRibbonKeyTip Members 7279
IRibbonKeyTip methods 7279
IRibbonKeyTip properties 7282
KeyTip 7282
KeyTipClick 7281
SetKeyTip 7281
ShowKeyTip 7281
IRibbonQuickAccessToolbar interface 6934
about IRibbonQuickAccessToolbar interface 6934
DoUpdateQATWidth 6935
IRibbonQuickAccessToolbar Members 6935
IRibbonQuickAccessToolbar methods 6935
IRibbonSplitButton interface 7282
about IRibbonSplitButton interface 7282
IRibbonSplitButton Members 7283
IRibbonSplitButton methods 7283
UpdateSplitBounds 7283
IRIOAccess interface 7420
about IRIOAccess interface 7420
GetRIO 7421
IRIOAccess Members 7420
IRIOAccess methods 7421
IRIOAccess properties 7421
RIO 7421
ISAPIApp 4499
ISAPIApp namespace 4499
Classes 4500
Functions 4505
Variables 4507
ISAPIApp.DispatchThread 4507
ISAPIApp.DispatchThread variable 4507
ISAPIApp.GetExtensionVersion 4505
ISAPIApp.GetExtensionVersion function 4505
ISAPIApp.HttpExtensionProc 4505
ISAPIApp.HttpExtensionProc function 4505
ISAPIApp.TerminateExtension 4506
ISAPIApp.TerminateExtension function 4506
ISAPIApp.TISAPIApplication 4500
ISAPIApp.TISAPIApplication.Create 4501

ISAPIApp.TISAPIApplication.Destroy 4501
ISAPIApp.TISAPIApplication.GetExtensionVersion 4502
ISAPIApp.TISAPIApplication.HttpExtensionProc 4502
ISAPIApp.TISAPIApplication.TerminateExtension 4503
ISAPIApp.TISAPIFactory 4503
ISAPIApp.TISAPIFactory.Create 4503
ISAPIApp.TISAPIFactory.NewRequest 4504
ISAPIApp.TISAPIFactory.NewResponse 4504
IsapiHTTP 4509
IsapiHTTP namespace 4509
Classes 4510
IsapiHTTP.TISAPIRequest 4510
IsapiHTTP.TISAPIRequest.Create 4512
IsapiHTTP.TISAPIRequest.ECB 4518
IsapiHTTP.TISAPIRequest.GetDateVariable 4513
IsapiHTTP.TISAPIRequest.GetFieldByName 4513
IsapiHTTP.TISAPIRequest.GetIntegerVariable 4514
IsapiHTTP.TISAPIRequest.GetStringVariable 4514
IsapiHTTP.TISAPIRequest.ReadClient 4515
IsapiHTTP.TISAPIRequest.ReadString 4515
IsapiHTTP.TISAPIRequest.TranslateURI 4516
IsapiHTTP.TISAPIRequest.WriteClient 4516
IsapiHTTP.TISAPIRequest.WriteHeaders 4516
IsapiHTTP.TISAPIRequest.WriteString 4517
IsapiHTTP.TISAPIResponse 4518
IsapiHTTP.TISAPIResponse.Create 4520
IsapiHTTP.TISAPIResponse.GetContent 4521
IsapiHTTP.TISAPIResponse.GetDateVariable 4521
IsapiHTTP.TISAPIResponse.GetIntegerVariable 4522
IsapiHTTP.TISAPIResponse.GetLogMessage 4522
IsapiHTTP.TISAPIResponse.GetStatusCode 4522
IsapiHTTP.TISAPIResponse.GetStringVariable 4523
IsapiHTTP.TISAPIResponse.InitResponse 4523
IsapiHTTP.TISAPIResponse.SendRedirect 4524
IsapiHTTP.TISAPIResponse.SendResponse 4524
IsapiHTTP.TISAPIResponse.SendStream 4524
IsapiHTTP.TISAPIResponse.Sent 4525
IsapiHTTP.TISAPIResponse.SetContent 4525
IsapiHTTP.TISAPIResponse.SetDateVariable 4525
IsapiHTTP.TISAPIResponse.SetIntegerVariable 4526
IsapiHTTP.TISAPIResponse.SetLogMessage 4526
IsapiHTTP.TISAPIResponse.SetStatusCode 4526
IsapiHTTP.TISAPIResponse.SetStringVariable 4527
IScriptContext interface 3506
about IScriptContext interface 3506
GetWebModuleContext 3507
IScriptContext Members 3507
IScriptContext methods 3507
IScriptContext properties 3507
WebModuleContext 3507
IScriptProducer interface 3508
about IScriptProducer interface 3508
Content 3512
Errors 3512
Evaluate 3509
GetContent 3509
GetErrors 3509
GetHTMLBlock 3510
GetHTMLBlockCount 3510
HandleScriptError 3510
HTMLBlockCount 3513
HTMLBlocks 3513
IScriptProducer Members 3508
IScriptProducer methods 3508
IScriptProducer properties 3512
ParseStream 3510
ParseString 3510
ReplaceTags 3511
SetContent 3511
Write 3511
Write method 3511
WriteItem 3512
ISecurityProperty interface 5867
about ISecurityProperty interface 5867
GetDirectCallerSID 5868
GetDirectCreatorSID 5868
GetOriginalCallerSID 5868
GetOriginalCreatorSID 5869

ISecurityProperty Members 5868
ISecurityProperty methods 5868
ReleaseSID 5869
ISetAppDispatcher interface 3423
about ISetAppDispatcher interface 3423
ISetAppDispatcher Members 3423
ISetAppDispatcher methods 3423
SetAppDispatcher 3424
ISharedProperty interface 5869
about ISharedProperty interface 5869
Get_Value 5870
ISharedProperty Members 5869
ISharedProperty methods 5870
ISharedProperty properties 5870
Set_Value 5870
Value 5871
ISharedPropertyGroup interface 5871
about ISharedPropertyGroup interface 5871
CreateProperty 5872
CreatePropertyByPosition 5872
Get_PropertyByName 5872
Get_PropertyByPosition 5872
ISharedPropertyGroup Members 5871
ISharedPropertyGroup methods 5871
ISharedPropertyGroup properties 5873
PropertyByName 5873
PropertyByPosition 5873
ISharedPropertyGroupManager interface 5873
_NewEnum 5875
about ISharedPropertyGroupManager interface 5873
CreatePropertyGroup 5874
Get__NewEnum 5874
Get_Group 5875
Group 5875
ISharedPropertyGroupManager Members 5874
ISharedPropertyGroupManager methods 5874
ISharedPropertyGroupManager properties 5875
ISkeleton interface 6229
about ISkeleton interface 6229
GetImplementation 6230
GetReplyBuffer 6230
ISkeleton Members 6229
ISkeleton methods 6230
ISkeletonObject interface 6230
about ISkeletonObject interface 6230
Execute 6231
GetImplementation 6231
GetSkeleton 6231
ISkeletonObject Members 6231
ISkeletonObject methods 6231
isLarge enumeration member 6936
isMedium enumeration member 6936
ISOAPHeaders interface 4464
about ISOAPHeaders interface 4464
Get 4466
Get method 4465
GetOwnsSentHeaders 4466
ISOAPHeaders Members 4465
ISOAPHeaders methods 4465
ISOAPHeaders properties 4468
OwnsSentHeaders 4468
Send 4467
SendAt 4467
SendCount 4468
SetOwnsSentHeaders 4468
ISpecialWinHelpViewer interface 3239
about ISpecialWinHelpViewer interface 3239
CallWinHelp 3241
ISpecialWinHelpViewer Members 3240
ISpecialWinHelpViewer methods 3240
isSmall enumeration member 6936
IStatusField interface 5666
about IStatusField interface 5666
IStreamLoader interface 3528
about IStreamLoader interface 3528
GetPassword 3529
GetProxy 3530
GetTimeout 3530

GetUserName 3530
IStreamLoader Members 3528
IStreamLoader methods 3529
IStreamLoader properties 3532
Load 3530
Password 3532
Proxy 3533
SetPassword 3531
SetProxy 3531
SetTimeout 3532
SetUserName 3532
Timeout 3533
UserName 3533
IStringTokenizer interface 3534
about IStringTokenizer interface 3534
countTokens 3536
getTokenCounts 3535
hasMoreTokens 3535
IStringTokenizer Members 3534
IStringTokenizer methods 3535
IStringTokenizer properties 3536
nextToken 3535
IStub interface 6232
about IStub interface 6232
CreateRequest 6233
Dispatch 6233
GetInterface 6233
Hash 6233
Invoke 6234
IsA 6234
IStub Members 6232
IStub methods 6232
NonExistent 6234
RepositoryID 6234
SetPrincipal 6235
IStubObject interface 6235
about IStubObject interface 6235
GetStub 6236
IStubObject Members 6235
IStubObject methods 6235
itImage enumeration member 4229
itMask enumeration member 4229
ITransactionContext interface 5875
Abort 5876
about ITransactionContext interface 5875
Commit 5876
CreateInstance 5877
ITransactionContext Members 5876
ITransactionContext methods 5876
ITransactionContextEx interface 5877
Abort 5878
about ITransactionContextEx interface 5877
Commit 5878
CreateInstance 5878
ITransactionContextEx Members 5877
ITransactionContextEx methods 5877
ITypeCode interface 6236
about ITypeCode interface 6236
ContentType 6237
DefaultIndex 6237
DiscriminatorType 6237
Equal 6238
Id 6238
ITypeCode Members 6236
ITypeCode methods 6237
Kind 6238
Length 6238
MemberCount 6239
MemberLabel 6239
MemberName 6239
MemberType 6239
Name 6239
Parameter 6240
ParameterCount 6240
IValidateField interface 5667
about IValidateField interface 5667
IValidateField Members 5667
IValidateField methods 5667

ValidateField 5667
IValidateFields interface 5668
about IValidateFields interface 5668
EnableValidateFields 5669
GetEnableValidateFields 5669
IValidateFields Members 5668
IValidateFields methods 5668
IValidateFields properties 5669
SetEnableValidateFields 5669
ValidateFields 5669
IVCLFrameForm interface 6053
about IVCLFrameForm interface 6053
AddContainer 6054
ClearBorderSpace 6054
Form 6054
IVCLFrameForm Members 6053
IVCLFrameForm methods 6053
RemoveContainer 6054
IWebAppServices interface 3424
about IWebAppServices interface 3424
ExceptionHandler 3427
FinishContext 3425
GetExceptionHandler 3426
HandleRequest 3426
InitContext 3426
IWebAppServices Members 3425
IWebAppServices methods 3425
IWebAppServices properties 3427
IWebDispatch interface 3427
about IWebDispatch interface 3427
DispatchEnabled 3428
DispatchMask 3429
DispatchMethodType 3429
DispatchRequest 3429
Enabled 3429
IWebDispatch Members 3428
IWebDispatch methods 3428
IWebDispatch properties 3429
Mask 3430
MethodType 3430
IWebDispatchActions interface 3430
about IWebDispatchActions interface 3430
IWebDispatcherAccess interface 3431
about IWebDispatcherAccess interface 3431
IWebDispatcherAccess Members 3432
IWebDispatcherAccess methods 3432
Request 3432
Response 3432
IWebDisplay interface 5670
about IWebDisplay interface 5670
GetIsMultipleRecordView 5670
IsMultipleRecordView 5671
IWebDisplay Members 5670
IWebDisplay methods 5670
IWebDisplay properties 5671
IWebDisplayReference interface 5671
about IWebDisplayReference interface 5671
GetWebDisplayComponent 5672
IWebDisplayReference Members 5671
IWebDisplayReference methods 5672
IWebDisplayReference properties 5672
WebDisplayComponent 5672
IWebExceptionHandler interface 3433
about IWebExceptionHandler interface 3433
HandleException 3433
IWebExceptionHandler Members 3433
IWebExceptionHandler methods 3433
IWebRequestHandler interface 3434
about IWebRequestHandler interface 3434
HandleRequest 3435
IWebRequestHandler Members 3434
IWebRequestHandler methods 3434
IWebVariableName interface 3513
about IWebVariableName interface 3513
GetVariableName 3514
IWebVariableName Members 3513
IWebVariableName methods 3514
IWebVariableName properties 3514

VariableName 3514
IWebVariablesContainer interface 3514
about IWebVariablesContainer interface 3514
FindVariable 3515
FindVariableContainer 3516
GetVariable 3516
GetVariableCount 3516
IWebVariablesContainer Members 3515
IWebVariablesContainer methods 3515
IWebVariablesContainer properties 3516
VariableCount 3516
Variables 3517
IXMLDisplay interface 5673
about IXMLDisplay interface 5673
GetIsMultipleRecordView 5673
GetXMLDisplayName 5674
GetXMLRowSetName 5674
IsMultipleRecordView 5674
IXMLDisplay Members 5673
IXMLDisplay methods 5673
IXMLDisplay properties 5674
XMLDisplayName 5675
XMLRowSetName 5675

J

Java 4529
Java namespace 4529
Java.Sql 4530
Java.Sql namespace 4530
Classes 4530
Java.Sql.SQLException 4530
Java.Sql.SQLException.ErrorCode 4531
Java.Sql.SQLException.SQLException 4531
Java.Sql.SQLException.SQLState 4532
jsBind enumeration member 5742
jsDB enumeration member 5742
jsDisp enumeration member 5742
jsDom enumeration member 5742
jsErrDisp enumeration member 5742
jsShow enumeration member 5742

K

KillShadows enumeration member 4026
ktFull enumeration member 7286
ktNone enumeration member 7286
ktpApplicationButton enumeration member 7287
ktPartial enumeration member 7286
ktpGroupA enumeration member 7287
ktpGroupB enumeration member 7287
ktpGroupC enumeration member 7287
ktpGroupD enumeration member 7287
ktpQuickAccessToolbar enumeration member 7287
ktpTabs enumeration member 7287
ktsCollapsedGroup enumeration member 7288
ktsControls enumeration member 7288
ktsTabs enumeration member 7288

L

License enumeration member 4020
LicenseAdd enumeration member 4019
LicenseMask enumeration member 4020
LicenseRemove enumeration member 4019
LimboState enumeration member 4025
LimboTransactions enumeration member 4026
ListActns 4533
ListActns namespace 4533
Classes 4535
Structs, Records, Enums 4586
Types 4587
ListActns.TCustomListAction 4535
ListActns.TCustomListAction.Active 4540
ListActns.TCustomListAction.Count 4540
ListActns.TCustomListAction.Create 4536
ListActns.TCustomListAction.ExecuteTarget 4537
ListActns.TCustomListAction.GetCount 4537
ListActns.TCustomListAction.GetString 4538
ListActns.TCustomListAction.HandlesTarget 4538
ListActns.TCustomListAction.Images 4540

ListActns.TCustomListAction.ItemIndex 4541
ListActns.TCustomListAction.Loaded 4538
ListActns.TCustomListAction.Loading 4541
ListActns.TCustomListAction.OnGetItemCount 4542
ListActns.TCustomListAction.OnItemSelected 4542
ListActns.TCustomListAction.SetString 4539
ListActns.TCustomListAction.Strings 4541
ListActns.TCustomStaticListAction 4543
ListActns.TCustomStaticListAction.Clients 4547
ListActns.TCustomStaticListAction.Count 4547
ListActns.TCustomStaticListAction.Create 4544
ListActns.TCustomStaticListAction.Destroy 4544
ListActns.TCustomStaticListAction.GetCount 4545
ListActns.TCustomStaticListAction.GetItem 4545
ListActns.TCustomStaticListAction.GetItemClass 4545
ListActns.TCustomStaticListAction.GetString 4546
ListActns.TCustomStaticListAction.Items 4547
ListActns.TCustomStaticListAction.OnGetItem 4547
ListActns.TCustomStaticListAction.SetString 4546
ListActns.TCustomVirtualListAction 4548
ListActns.TCustomVirtualListAction.Count 4550
ListActns.TCustomVirtualListAction.GetItem 4549
ListActns.TCustomVirtualListAction.GetString 4549
ListActns.TCustomVirtualListAction.OnGetItem 4550
ListActns.TGetItemCountEvent 4587
ListActns.TGetItemCountEvent type 4587
ListActns.TGetItemEvent 4587
ListActns.TGetItemEvent type 4587
ListActns.TGetVirtualItemEvent 4588
ListActns.TGetVirtualItemEvent type 4588
ListActns.TItemSelectedEvent 4588
ListActns.TItemSelectedEvent type 4588
ListActns.TListActionLink 4550
ListActns.TListActionLink.AddItem 4552
ListActns.TListActionLink.IsActiveLinked 4552
ListActns.TListActionLink.IsImagesLinked 4552
ListActns.TListActionLink.RefreshControl 4553
ListActns.TListActionLink.SetAction 4553
ListActns.TListActionLink.SetActive 4553
ListActns.TListActionLink.SetImages 4553
ListActns.TListActionLink.SetItemIndex 4553
ListActns.TListCompareEvent 4588
ListActns.TListCompareEvent type 4588
ListActns.TListControlItem 4554
ListActns.TListControlItem.Assign 4556
ListActns.TListControlItem.Caption 4559
ListActns.TListControlItem.Changed 4557
ListActns.TListControlItem.Create 4555
ListActns.TListControlItem.Data 4560
ListActns.TListControlItem.FCaption 4555
ListActns.TListControlItem.FData 4556
ListActns.TListControlItem.FImageIndex 4556
ListActns.TListControlItem.GetDisplayName 4558
ListActns.TListControlItem.ImageIndex 4560
ListActns.TListControlItem.SetCaption 4558
ListActns.TListControlItem.SetData 4558
ListActns.TListControlItem.SetImageIndex 4559
ListActns.TListControlItemClass 4589
ListActns.TListControlItemClass type 4589
ListActns.TListControlItems 4560
ListActns.TListControlItems.Add 4562
ListActns.TListControlItems.CaseSensitive 4565
ListActns.TListControlItems.CompareItems 4563
ListActns.TListControlItems.Create 4562
ListActns.TListControlItems.CustomSort 4563
ListActns.TListControlItems.Items 4565
ListActns.TListControlItems.OnCompare 4567
ListActns.TListControlItems.Sort 4564
ListActns.TListControlItems.SortType 4566
ListActns.TListControlItems.Update 4564
ListActns.TListItemsCompare 4589
ListActns.TListItemsCompare type 4589
ListActns.TListItemsSortType 4586
ListActns.TListItemsSortType enumeration 4586
ListActns.TStaticListAction 4567
ListActns.TStaticListAction.Active 4569
ListActns.TStaticListAction.Caption 4570
ListActns.TStaticListAction.Enabled 4570

ListActns.TStaticListAction.HelpContext 4571
ListActns.TStaticListAction.Hint 4571
ListActns.TStaticListAction.Images 4571
ListActns.TStaticListAction.ItemIndex 4572
ListActns.TStaticListAction.Items 4572
ListActns.TStaticListAction.OnGetItem 4574
ListActns.TStaticListAction.OnHint 4574
ListActns.TStaticListAction.OnItemSelected 4574
ListActns.TStaticListAction.OnUpdate 4575
ListActns.TStaticListAction.SecondaryShortCuts 4572
ListActns.TStaticListAction.ShortCut 4573
ListActns.TStaticListAction.Visible 4573
ListActns.TStaticListItems 4576
ListActns.TStaticListItems.Notify 4577
ListActns.TStaticListItems.Update 4578
ListActns.TVirtualListAction 4578
ListActns.TVirtualListAction.Active 4580
ListActns.TVirtualListAction.Caption 4581
ListActns.TVirtualListAction.Enabled 4581
ListActns.TVirtualListAction.HelpContext 4581
ListActns.TVirtualListAction.Hint 4582
ListActns.TVirtualListAction.Images 4582
ListActns.TVirtualListAction.ItemIndex 4582
ListActns.TVirtualListAction.OnGetItem 4584
ListActns.TVirtualListAction.OnGetItemCount 4584
ListActns.TVirtualListAction.OnHint 4585
ListActns.TVirtualListAction.OnItemSelected 4585
ListActns.TVirtualListAction.SecondaryShortCuts 4583
ListActns.TVirtualListAction.ShortCut 4583
ListActns.TVirtualListAction.Visible 4584
lmDelete enumeration member 3692
lmInsert enumeration member 3692
lmModify enumeration member 3692
lmRefresh enumeration member 3692
Local enumeration member 4020
lpAbove enumeration member 1543
lpBelow enumeration member 1543
lpLeft enumeration member 1543
lpRight enumeration member 1543
lrDefaultColor enumeration member 4229
lrDefaultSize enumeration member 4229
lrFromFile enumeration member 4229
lrMap3DColors enumeration member 4229
lrMonoChrome enumeration member 4229
lrTransparent enumeration member 4229
lsButtons enumeration member 5682
lsFields enumeration member 5682
lsUnknown enumeration member 5682
ltFile enumeration member 7373
ltResource enumeration member 7373

M

maAutomatic enumeration member 4967
maBottomToTop enumeration member 4966
maLeftToRight enumeration member 4966
maManual enumeration member 4967
maNone enumeration member 4966
maParent enumeration member 4967
maRightToLeft enumeration member 4966
Mask 4591
Mask namespace 4591
    Classes 4592
    Types 4681
Mask.EDBEditError 4592
Mask.TCustomMaskEdit 4592
Mask.TCustomMaskEdit.CheckCursor 4595
Mask.TCustomMaskEdit.Clear 4595
Mask.TCustomMaskEdit.Create 4594
Mask.TCustomMaskEdit.EditCanModify 4596
Mask.TCustomMaskEdit.EditMask 4604
Mask.TCustomMaskEdit.EditText 4606
Mask.TCustomMaskEdit.GetFirstEditChar 4596
Mask.TCustomMaskEdit.GetLastEditChar 4597
Mask.TCustomMaskEdit.GetMaxChars 4597
Mask.TCustomMaskEdit.GetNextEditChar 4598
Mask.TCustomMaskEdit.GetPriorEditChar 4598
Mask.TCustomMaskEdit.GetSel 4599
Mask.TCustomMaskEdit.GetTextLen 4599

Mask.TCustomMaskEdit.IsMasked 4606
Mask.TCustomMaskEdit.KeyDown 4599
Mask.TCustomMaskEdit.KeyPress 4600
Mask.TCustomMaskEdit.KeyUp 4600
Mask.TCustomMaskEdit.MaskState 4607
Mask.TCustomMaskEdit.MaxLength 4607
Mask.TCustomMaskEdit.ReformatText 4601
Mask.TCustomMaskEdit.Reset 4601
Mask.TCustomMaskEdit.SetCursor 4602
Mask.TCustomMaskEdit.SetSel 4602
Mask.TCustomMaskEdit.Text 4608
Mask.TCustomMaskEdit.Validate 4603
Mask.TCustomMaskEdit.ValidateEdit 4603
Mask.TCustomMaskEdit.ValidateError 4603
Mask.TMaskEdit 4610
Mask.TMaskEdit.Align 4614
Mask.TMaskEdit.Alignment 4617
Mask.TMaskEdit.Anchors 4618
Mask.TMaskEdit.AutoSelect 4618
Mask.TMaskEdit.AutoSize 4619
Mask.TMaskEdit.BevelEdges 4620
Mask.TMaskEdit.BevelInner 4620
Mask.TMaskEdit.BevelKind 4620
Mask.TMaskEdit.BevelOuter 4621
Mask.TMaskEdit.BevelWidth 4621
Mask.TMaskEdit.BiDiMode 4622
Mask.TMaskEdit.BorderStyle 4622
Mask.TMaskEdit.CharCase 4622
Mask.TMaskEdit.Color 4623
Mask.TMaskEdit.Constraints 4624
Mask.TMaskEdit.Ctl3D 4624
Mask.TMaskEdit.DoubleBuffered 4625
Mask.TMaskEdit.DragCursor 4627
Mask.TMaskEdit.DragKind 4627
Mask.TMaskEdit.DragMode 4627
Mask.TMaskEdit.EditMask 4629
Mask.TMaskEdit.Enabled 4631
Mask.TMaskEdit.Font 4631
Mask.TMaskEdit.ImeMode 4632
Mask.TMaskEdit.ImeName 4633
Mask.TMaskEdit.MaxLength 4633
Mask.TMaskEdit.OnChange 4649
Mask.TMaskEdit.OnClick 4649
Mask.TMaskEdit.OnDblClick 4651
Mask.TMaskEdit.OnDragDrop 4652
Mask.TMaskEdit.OnDragOver 4655
Mask.TMaskEdit.OnEndDock 4658
Mask.TMaskEdit.OnEndDrag 4658
Mask.TMaskEdit.OnEnter 4659
Mask.TMaskEdit.OnExit 4662
Mask.TMaskEdit.OnKeyDown 4664
Mask.TMaskEdit.OnKeyPress 4668
Mask.TMaskEdit.OnKeyUp 4669
Mask.TMaskEdit.OnMouseActivate 4671
Mask.TMaskEdit.OnMouseDown 4672
Mask.TMaskEdit.OnMouseEnter 4674
Mask.TMaskEdit.OnMouseLeave 4674
Mask.TMaskEdit.OnMouseMove 4675
Mask.TMaskEdit.OnMouseUp 4677
Mask.TMaskEdit.OnStartDock 4679
Mask.TMaskEdit.OnStartDrag 4680
Mask.TMaskEdit.ParentBiDiMode 4634
Mask.TMaskEdit.ParentColor 4634
Mask.TMaskEdit.ParentCtl3D 4635
Mask.TMaskEdit.ParentDoubleBuffered 4635
Mask.TMaskEdit.ParentFont 4635
Mask.TMaskEdit.ParentShowHint 4636
Mask.TMaskEdit.PasswordChar 4636
Mask.TMaskEdit.PopupMenu 4637
Mask.TMaskEdit.ReadOnly 4642
Mask.TMaskEdit.ShowHint 4642
Mask.TMaskEdit.TabOrder 4644
Mask.TMaskEdit.TabStop 4644
Mask.TMaskEdit.Text 4645
Mask.TMaskEdit.TextHint 4647
Mask.TMaskEdit.Visible 4647
Mask.TMaskedState 4681
Mask.TMaskedState type 4681

Masks 4683
Masks namespace 4683
Classes 4684
Functions 4688
Masks.EMaskException 4684
Masks.MatchesMask 4688
Masks.MatchesMask function 4688
Masks.TMask 4684
Masks.TMask.Create 4685
Masks.TMask.Destroy 4686
Masks.TMask.Matches 4686
MaskUtils 4689
MaskUtils namespace 4689
Constants 4704
Functions 4691
Structs, Records, Enums 4698
Types 4699
Variables 4702
MaskUtils.DefaultBlank 4702
MaskUtils.DefaultBlank variable 4702
MaskUtils.FormatMaskText 4691
MaskUtils.FormatMaskText function 4691
MaskUtils.IsLiteralChar 4693
MaskUtils.IsLiteralChar function 4693
MaskUtils.MaskDoFormatText 4693
MaskUtils.MaskDoFormatText function 4693
MaskUtils.MaskFieldSeparator 4702
MaskUtils.MaskFieldSeparator variable 4702
MaskUtils.MaskGetCharType 4693
MaskUtils.MaskGetCharType function 4693
MaskUtils.MaskGetCurrentDirectives 4694
MaskUtils.MaskGetCurrentDirectives function 4694
MaskUtils.MaskGetFldSeparator 4694
MaskUtils.MaskGetFldSeparator function 4694
MaskUtils.MaskGetMaskBlank 4695
MaskUtils.MaskGetMaskBlank function 4695
MaskUtils.MaskGetMaskSave 4695
MaskUtils.MaskGetMaskSave function 4695
MaskUtils.MaskIntlLiteralToChar 4695
MaskUtils.MaskIntlLiteralToChar function 4695
MaskUtils.MaskNoSave 4703
MaskUtils.MaskNoSave variable 4703
MaskUtils.MaskOffsetToOffset 4696
MaskUtils.MaskOffsetToOffset function 4696
MaskUtils.MaskOffsetToWideOffset 4696
MaskUtils.MaskOffsetToWideOffset function 4696
MaskUtils.mDirLiteral 4704
MaskUtils.mDirLiteral constant 4704
MaskUtils.mDirLowerCase 4704
MaskUtils.mDirLowerCase constant 4704
MaskUtils.mDirReverse 4705
MaskUtils.mDirReverse constant 4705
MaskUtils.mDirUpperCase 4705
MaskUtils.mDirUpperCase constant 4705
MaskUtils.mMskAlpha 4706
MaskUtils.mMskAlpha constant 4706
MaskUtils.mMskAlphaNum 4706
MaskUtils.mMskAlphaNum constant 4706
MaskUtils.mMskAlphaNumOpt 4706
MaskUtils.mMskAlphaNumOpt constant 4706
MaskUtils.mMskAlphaOpt 4707
MaskUtils.mMskAlphaOpt constant 4707
MaskUtils.mMskAscii 4707
MaskUtils.mMskAscii constant 4707
MaskUtils.mMskAsciiOpt 4708
MaskUtils.mMskAsciiOpt constant 4708
MaskUtils.mMskDateSeparator 4708
MaskUtils.mMskDateSeparator constant 4708
MaskUtils.mMskNumeric 4708
MaskUtils.mMskNumeric constant 4708
MaskUtils.mMskNumericOpt 4709
MaskUtils.mMskNumericOpt constant 4709
MaskUtils.mMskNumSymOpt 4709
MaskUtils.mMskNumSymOpt constant 4709
MaskUtils.mMskTimeSeparator 4710
MaskUtils.mMskTimeSeparator constant 4710
MaskUtils.OffsetToMaskOffset 4697
MaskUtils.OffsetToMaskOffset function 4697

MaskUtils.PadInputLiterals 4697
MaskUtils.PadInputLiterals function 4697
MaskUtils.TEditMask 4699
MaskUtils.TEditMask type 4699
MaskUtils.TMaskCharType 4698
MaskUtils.TMaskCharType enumeration 4698
MaskUtils.TMaskDirectives 4700
MaskUtils.TMaskDirectives type 4700
MaskUtils.TMaskedText 4701
MaskUtils.TMaskedText type 4701
Math 4711
Math namespace 4711
Classes 4715
Constants 4776
Functions 4717
Structs, Records, Enums 4769
Types 4773
Math.ArcCos 4719
Math.ArcCos function 4719
Math.ArcCosh 4720
Math.ArcCosh function 4720
Math.ArcCot 4720
Math.ArcCot function 4720
Math.ArcCotH 4721
Math.ArcCotH function 4721
Math.ArcCsc 4721
Math.ArcCsc function 4721
Math.ArcCscH 4722
Math.ArcCscH function 4722
Math.ArcSec 4722
Math.ArcSec function 4722
Math.ArcSecH 4722
Math.ArcSecH function 4722
Math.ArcSin 4723
Math.ArcSin function 4723
Math.ArcSinh 4723
Math.ArcSinh function 4723
Math.ArcTan2 4724
Math.ArcTan2 function 4724
Math.ArcTanh 4724
Math.ArcTanh function 4724
Math.Ceil 4725
Math.Ceil function 4725
Math.ClearExceptions 4725
Math.ClearExceptions function 4725
Math.CompareValue 4726
Math.CompareValue function 4726
Math.Cosecant 4726
Math.Cosecant function 4726
Math.Cosh 4727
Math.Cosh function 4727
Math.Cot 4727
Math.Cot function 4727
Math.Cotan 4728
Math.Cotan function 4728
Math.CotH 4728
Math.CotH function 4728
Math.Csc 4729
Math.Csc function 4729
Math.CscH 4729
Math.CscH function 4729
Math.CycleToDeg 4730
Math.CycleToDeg function 4730
Math.CycleToGrad 4730
Math.CycleToGrad function 4730
Math.CycleToRad 4730
Math.CycleToRad function 4730
Math.DegToCycle 4731
Math.DegToCycle function 4731
Math.DegToGrad 4731
Math.DegToGrad function 4731
Math.DegToRad 4732
Math.DegToRad function 4732
Math.DivMod 4732
Math.DivMod function 4732
Math.DoubleDecliningBalance 4732
Math.DoubleDecliningBalance function 4732
Math.EInvalidArgument 4715

Math.EnsureRange 4733
Math.EnsureRange function 4733
Math.Floor 4733
Math.Floor function 4733
Math.Frexp 4734
Math.Frexp function 4734
Math.FutureValue 4734
Math.FutureValue function 4734
Math.GetExceptionMask 4735
Math.GetExceptionMask function 4735
Math.GetPrecisionMode 4735
Math.GetPrecisionMode function 4735
Math.GetRoundMode 4736
Math.GetRoundMode function 4736
Math.GradToCycle 4737
Math.GradToCycle function 4737
Math.GradToDeg 4737
Math.GradToDeg function 4737
Math.GradToRad 4737
Math.GradToRad function 4737
Math.Hypot 4738
Math.Hypot function 4738
Math.IfThen 4738
Math.IfThen function 4738
Math.Infinity 4776
Math.Infinity constant 4776
Math.InRange 4738
Math.InRange function 4738
Math.InterestPayment 4739
Math.InterestPayment function 4739
Math.InterestRate 4740
Math.InterestRate function 4740
Math.InternalRateOfReturn 4740
Math.InternalRateOfReturn function 4740
Math.IntPower 4741
Math.IntPower function 4741
Math.IsInfinite 4741
Math.IsInfinite function 4741
Math.IsNan 4742
Math.IsNan function 4742
Math.IsZero 4742
Math.IsZero function 4742
Math.Ldexp 4743
Math.Ldexp function 4743
Math.LnXP1 4743
Math.LnXP1 function 4743
Math.Log10 4743
Math.Log10 function 4743
Math.Log2 4744
Math.Log2 function 4744
Math.LogN 4744
Math.LogN function 4744
Math.Max 4745
Math.Max function 4745
Math.MaxComp 4777
Math.MaxComp constant 4777
Math.MaxDouble 4777
Math.MaxDouble constant 4777
Math.MaxExtended 4777
Math.MaxExtended constant 4777
Math.MaxIntValue 4745
Math.MaxIntValue function 4745
Math.MaxSingle 4778
Math.MaxSingle constant 4778
Math.MaxValue 4746
Math.MaxValue function 4746
Math.Mean 4746
Math.Mean function 4746
Math.MeanAndStdDev 4747
Math.MeanAndStdDev function 4747
Math.Min 4747
Math.Min function 4747
Math.MinComp 4778
Math.MinComp constant 4778
Math.MinDouble 4779
Math.MinDouble constant 4779
Math.MinExtended 4779
Math.MinExtended constant 4779

Math.MinIntValue 4748
Math.MinIntValue function 4748
Math.MinSingle 4779
Math.MinSingle constant 4779
Math.MinValue 4748
Math.MinValue function 4748
Math.MomentSkewKurtosis 4749
Math.MomentSkewKurtosis function 4749
Math.NaN 4780
Math.NaN constant 4780
Math.NegativeValue 4780
Math.NegativeValue constant 4780
Math.NegInfinity 4781
Math.NegInfinity constant 4781
Math.NetPresentValue 4749
Math.NetPresentValue function 4749
Math.Norm 4750
Math.Norm function 4750
Math.NumberOfPeriods 4750
Math.NumberOfPeriods function 4750
Math.Payment 4751
Math.Payment function 4751
Math.PeriodPayment 4751
Math.PeriodPayment function 4751
Math.Poly 4751
Math.Poly function 4751
Math.PopnStdDev 4752
Math.PopnStdDev function 4752
Math.PopnVariance 4752
Math.PopnVariance function 4752
Math.PositiveValue 4781
Math.PositiveValue constant 4781
Math.Power 4753
Math.Power function 4753
Math.PresentValue 4753
Math.PresentValue function 4753
Math.RadToCycle 4754
Math.RadToCycle function 4754
Math.RadToDeg 4754
Math.RadToDeg function 4754
Math.RadToGrad 4754
Math.RadToGrad function 4754
Math.RandG 4755
Math.RandG function 4755
Math.RandomFrom 4755
Math.RandomFrom function 4755
Math.RandomRange 4756
Math.RandomRange function 4756
Math.RoundTo 4756
Math.RoundTo function 4756
Math.SameValue 4757
Math.SameValue function 4757
Math.Sec 4758
Math.Sec function 4758
Math.Secant 4758
Math.Secant function 4758
Math.SecH 4759
Math.SecH function 4759
Math.SetExceptionMask 4759
Math.SetExceptionMask function 4759
Math.SetPrecisionMode 4760
Math.SetPrecisionMode function 4760
Math.SetRoundMode 4760
Math.SetRoundMode function 4760
Math.Sign 4761
Math.Sign function 4761
Math.SimpleRoundTo 4761
Math.SimpleRoundTo function 4761
Math.SinCos 4762
Math.SinCos function 4762
Math.Sinh 4762
Math.Sinh function 4762
Math.SLNDepreciation 4763
Math.SLNDepreciation function 4763
Math.StdDev 4763
Math.StdDev function 4763
Math.Sum 4764
Math.Sum function 4764

Math.SumInt 4764
Math.SumInt function 4764
Math.SumOfSquares 4765
Math.SumOfSquares function 4765
Math.SumsAndSquares 4765
Math.SumsAndSquares function 4765
Math.SYDDepreciation 4766
Math.SYDDepreciation function 4766
Math.Tan 4766
Math.Tan function 4766
Math.Tanh 4767
Math.Tanh function 4767
Math.TFPUException 4769
Math.TFPUException enumeration 4769
Math.TFPUExceptionMask 4773
Math.TFPUExceptionMask type 4773
Math.TFPUPrecisionMode 4770
Math.TFPUPrecisionMode enumeration 4770
Math.TFPURoundingMode 4771
Math.TFPURoundingMode enumeration 4771
Math.TotalVariance 4767
Math.TotalVariance function 4767
Math.TPaymentTime 4772
Math.TPaymentTime enumeration 4772
Math.TRoundToEXRangeExtended 4774
Math.TRoundToEXRangeExtended type 4774
Math.TRoundToRange 4774
Math.TRoundToRange type 4774
Math.TValueSign 4774
Math.TValueSign type 4774
Math.Variance 4768
Math.Variance function 4768
Math.ZeroValue 4782
Math.ZeroValue constant 4782
maTopToBottom enumeration member 4966
mbBarBreak enumeration member 4966
mbBreak enumeration member 4966
mbNone enumeration member 4966
mcDirective enumeration member 4698
mcField enumeration member 4698
mcFieldSeparator enumeration member 4698
mcIntlLiteral enumeration member 4698
mcLiteral enumeration member 4698
mcMask enumeration member 4698
mcMaskOpt enumeration member 4698
mcNone enumeration member 4698
MConnect 4783
MConnect namespace 4783
    Classes 4784
    Functions 4823
MConnect.GetMIDASAppServerList 4823
MConnect.GetMIDASAppServerList function 4823
MConnect.TCOMConnection 4784
MConnect.TCOMConnection.DoConnect 4786
MConnect.TCOMConnection.SetConnected 4786
MConnect.TCustomObjectBroker 4787
MConnect.TCustomObjectBroker.GetComputerForGUID 4788
MConnect.TCustomObjectBroker.GetComputerForProgID 4789
MConnect.TCustomObjectBroker.GetPortForComputer 4789
MConnect.TCustomObjectBroker.SetConnectStatus 4790
MConnect.TDCOMConnection 4790
MConnect.TDCOMConnection.ComputerName 4793
MConnect.TDCOMConnection.Create 4792
MConnect.TDCOMConnection.DoConnect 4793
MConnect.TDCOMConnection.ObjectBroker 4794
MConnect.TDispatchAppServer 4794
MConnect.TDispatchAppServer.Create 4795
MConnect.TDispatchAppServer.SafeCallException 4795
MConnect.TDispatchConnection 4796
MConnect.TDispatchConnection.AfterConnect 4808
MConnect.TDispatchConnection.AfterDisconnect 4808
MConnect.TDispatchConnection.BeforeConnect 4809
MConnect.TDispatchConnection.BeforeDisconnect 4809
MConnect.TDispatchConnection.Connected 4802
MConnect.TDispatchConnection.Create 4797
MConnect.TDispatchConnection.DoDisconnect 4798
MConnect.TDispatchConnection.GetConnected 4798
MConnect.TDispatchConnection.GetServer 4799

MConnect.TDispatchConnection.GetServerCLSID 4799
MConnect.TDispatchConnection.GetServerList 4800
MConnect.TDispatchConnection.LoginPrompt 4802
MConnect.TDispatchConnection.Notification 4800
MConnect.TDispatchConnection.ObjectBroker 4806
MConnect.TDispatchConnection.OnGetUsername 4810
MConnect.TDispatchConnection.OnLogin 4810
MConnect.TDispatchConnection.ServerGUID 4807
MConnect.TDispatchConnection.ServerName 4807
MConnect.TDispatchConnection.SetConnected 4800
MConnect.TOLEnterpriseConnection 4811
MConnect.TOLEnterpriseConnection.BrokerName 4813
MConnect.TOLEnterpriseConnection.ComputerName 4814
MConnect.TOLEnterpriseConnection.DoConnect 4813
MConnect.TSharedConnection 4814
MConnect.TSharedConnection.AfterConnect 4820
MConnect.TSharedConnection.AfterDisconnect 4821
MConnect.TSharedConnection.BeforeConnect 4821
MConnect.TSharedConnection.BeforeDisconnect 4822
MConnect.TSharedConnection.ChildName 4819
MConnect.TSharedConnection.ConnectEvent 4817
MConnect.TSharedConnection.Create 4816
MConnect.TSharedConnection.Destroy 4816
MConnect.TSharedConnection.DoConnect 4817
MConnect.TSharedConnection.DoDisconnect 4818
MConnect.TSharedConnection.GetConnected 4818
MConnect.TSharedConnection.GetServer 4818
MConnect.TSharedConnection.Notification 4819
MConnect.TSharedConnection.ParentConnection 4820
mdNearest enumeration member 2587
mdNull enumeration member 2587
mdPrimary enumeration member 2587
MendDB enumeration member 4026
Menus 4825
Menus namespace 4825
Classes 4827
Constants 4977
Functions 4951
Structs, Records, Enums 4965
Types 4970
Variables 4975
Menus.AnsiSameCaption 4951
Menus.AnsiSameCaption function 4951
Menus.cDialogSuffix 4977
Menus.cDialogSuffix constant 4977
Menus.cHotkeyPrefix 4977
Menus.cHotkeyPrefix constant 4977
Menus.cLineCaption 4977
Menus.cLineCaption constant 4977
Menus.DrawMenuItem 4952
Menus.DrawMenuItem function 4952
Menus.EMenuError 4827
Menus.GetHotkey 4952
Menus.GetHotkey function 4952
Menus.IsAltGRPressed 4952
Menus.IsAltGRPressed function 4952
Menus.NewItem 4953
Menus.NewItem function 4953
Menus.NewLine 4954
Menus.NewLine function 4954
Menus.NewMenu 4955
Menus.NewMenu function 4955
Menus.NewPopupMenu 4956
Menus.NewPopupMenu function 4956
Menus.NewSubMenu 4957
Menus.NewSubMenu function 4957
Menus.PMenuItem 4970
Menus.PMenuItem type 4970
Menus.PopupList 4975
Menus.PopupList variable 4975
Menus.SameCaption 4958
Menus.SameCaption function 4958
Menus.ShortCut 4958
Menus.ShortCut function 4958
Menus.ShortCutFromMessage 4959
Menus.ShortCutFromMessage function 4959
Menus.ShortCutItems 4975
Menus.ShortCutItems variable 4975

Menus.ShortCutToKey 4960
Menus.ShortCutToKey function 4960
Menus.ShortCutToText 4961
Menus.ShortCutToText function 4961
Menus.StripHotkey 4962
Menus.StripHotkey function 4962
Menus.TAdvancedMenuDrawItemEvent 4970
Menus.TAdvancedMenuDrawItemEvent type 4970
Menus.TextToShortCut 4963
Menus.TextToShortCut function 4963
Menus.TFindItemKind 4965
Menus.TFindItemKind enumeration 4965
Menus.TMainMenu 4828
Menus.TMainMenu.AutoHotkeys 4833
Menus.TMainMenu.AutoLineReduction 4834
Menus.TMainMenu.AutoMerge 4834
Menus.TMainMenu.BiDiMode 4835
Menus.TMainMenu.GetHandle 4830
Menus.TMainMenu.GetOle2AcceleratorTable 4830
Menus.TMainMenu.Images 4835
Menus.TMainMenu.MenuChanged 4831
Menus.TMainMenu.Merge 4831
Menus.TMainMenu.OnChange 4837
Menus.TMainMenu.OwnerDraw 4836
Menus.TMainMenu.ParentBiDiMode 4836
Menus.TMainMenu.PopulateOle2Menu 4832
Menus.TMainMenu.SetOle2MenuHandle 4832
Menus.TMainMenu.Unmerge 4832
Menus.TMenu 4837
Menus.TMenu.AdjustBiDiBehavior 4841
Menus.TMenu.AutoHotkeys 4849
Menus.TMenu.AutoLineReduction 4850
Menus.TMenu.BiDiMode 4850
Menus.TMenu.Create 4839
Menus.TMenu.CreateMenuItem 4841
Menus.TMenu.Destroy 4840
Menus.TMenu.DispatchCommand 4841
Menus.TMenu.DispatchPopup 4842
Menus.TMenu.DoBiDiModeChanged 4842
Menus.TMenu.DoChange 4842
Menus.TMenu.DoGetMenuString 4843
Menus.TMenu.FindItem 4843
Menus.TMenu.GetChildren 4844
Menus.TMenu.GetHandle 4844
Menus.TMenu.GetHelpContext 4844
Menus.TMenu.Handle 4851
Menus.TMenu.Images 4851
Menus.TMenu.IsOwnerDraw 4845
Menus.TMenu.IsRightToLeft 4845
Menus.TMenu.IsShortCut 4845
Menus.TMenu.Items 4852
Menus.TMenu.Loaded 4846
Menus.TMenu.MenuChanged 4846
Menus.TMenu.Notification 4847
Menus.TMenu.OnChange 4858
Menus.TMenu.OwnerDraw 4856
Menus.TMenu.ParentBiDiMode 4857
Menus.TMenu.ParentBiDiModeChanged 4847
Menus.TMenu.ProcessMenuChar 4848
Menus.TMenu.SetChildOrder 4848
Menus.TMenu.UpdateItems 4849
Menus.TMenu.WindowHandle 4857
Menus.TMenuActionLink 4858
Menus.TMenuActionLink.AssignClient 4860
Menus.TMenuActionLink.FClient 4860
Menus.TMenuActionLink.IsAutoCheckLinked 4861
Menus.TMenuActionLink.IsCaptionLinked 4861
Menus.TMenuActionLink.IsCheckedLinked 4862
Menus.TMenuActionLink.IsEnabledLinked 4862
Menus.TMenuActionLink.IsGroupIndexLinked 4862
Menus.TMenuActionLink.IsHelpContextLinked 4863
Menus.TMenuActionLink.IsHintLinked 4863
Menus.TMenuActionLink.IsImageIndexLinked 4864
Menus.TMenuActionLink.IsOnExecuteLinked 4864
Menus.TMenuActionLink.IsShortCutLinked 4864
Menus.TMenuActionLink.IsVisibleLinked 4865
Menus.TMenuActionLink.SetAutoCheck 4865
Menus.TMenuActionLink.SetCaption 4866

Menus.TMenuActionLink.SetChecked 4866
Menus.TMenuActionLink.SetEnabled 4866
Menus.TMenuActionLink.SetHelpContext 4867
Menus.TMenuActionLink.SetHint 4867
Menus.TMenuActionLink.SetImageIndex 4867
Menus.TMenuActionLink.SetOnExecute 4868
Menus.TMenuActionLink.SetShortCut 4868
Menus.TMenuActionLink.SetVisible 4869
Menus.TMenuActionLinkClass 4971
Menus.TMenuActionLinkClass type 4971
Menus.TMenuAnimation 4971
Menus.TMenuAnimation type 4971
Menus.TMenuAnimations 4966
Menus.TMenuAnimations enumeration 4966
Menus.TMenuAutoFlag 4972
Menus.TMenuAutoFlag type 4972
Menus.TMenuBreak 4966
Menus.TMenuBreak enumeration 4966
Menus.TMenuChangeEvent 4972
Menus.TMenuChangeEvent type 4972
Menus.TMenuDrawItemEvent 4973
Menus.TMenuDrawItemEvent type 4973
Menus.TMenuHandle 4973
Menus.TMenuHandle type 4973
Menus.TMenuItem 4869
Menus.TMenuItem.Action 4899
Menus.TMenuItem.ActionChange 4875
Menus.TMenuItem.ActionLink 4899
Menus.TMenuItem.Add 4875
Menus.TMenuItem.AdvancedDrawItem 4880
Menus.TMenuItem.AssignTo 4880
Menus.TMenuItem.AutoCheck 4899
Menus.TMenuItem.AutoHotkeys 4900
Menus.TMenuItem.AutoLineReduction 4900
Menus.TMenuItem.Bitmap 4901
Menus.TMenuItem.Break 4901
Menus.TMenuItem.Caption 4902
Menus.TMenuItem.Checked 4906
Menus.TMenuItem.Clear 4881
Menus.TMenuItem.Click 4881
Menus.TMenuItem.Command 4907
Menus.TMenuItem.Count 4907
Menus.TMenuItem.Create 4872
Menus.TMenuItem.Default 4912
Menus.TMenuItem.DefineProperties 4881
Menus.TMenuItem.Delete 4882
Menus.TMenuItem.Destroy 4873
Menus.TMenuItem.DoDrawText 4882
Menus.TMenuItem.DrawItem 4883
Menus.TMenuItem.Enabled 4912
Menus.TMenuItem.Find 4883
Menus.TMenuItem.GetActionLinkClass 4883
Menus.TMenuItem.GetAutoHotkeys 4884
Menus.TMenuItem.GetAutoLineReduction 4884
Menus.TMenuItem.GetChildren 4884
Menus.TMenuItem.GetCount 4884
Menus.TMenuItem.GetEnumerator 4884
Menus.TMenuItem.GetHandle 4885
Menus.TMenuItem.GetImageList 4885
Menus.TMenuItem.GetItem 4886
Menus.TMenuItem.GetMenuIndex 4886
Menus.TMenuItem.GetParentComponent 4886
Menus.TMenuItem.GetParentMenu 4886
Menus.TMenuItem.GroupIndex 4914
Menus.TMenuItem.Handle 4915
Menus.TMenuItem.HasParent 4887
Menus.TMenuItem.HelpContext 4915
Menus.TMenuItem.Hint 4916
Menus.TMenuItem.ImageIndex 4916
Menus.TMenuItem.IndexOf 4887
Menus.TMenuItem.InitiateAction 4887
Menus.TMenuItem.Insert 4888
Menus.TMenuItem.InsertNewLine 4889
Menus.TMenuItem.InsertNewLineAfter 4889
Menus.TMenuItem.InsertNewLineBefore 4890
Menus.TMenuItem.IsLine 4890
Menus.TMenuItem.Items 4917
Menus.TMenuItem.Loaded 4891

Menus.TMenuItem.MeasureItem 4891
Menus.TMenuItem.MenuChanged 4892
Menus.TMenuItem.MenuIndex 4919
Menus.TMenuItem.NewBottomLine 4892
Menus.TMenuItem.NewTopLine 4892
Menus.TMenuItem.Notification 4893
Menus.TMenuItem.OnAdvancedDrawItem 4922
Menus.TMenuItem.OnClick 4922
Menus.TMenuItem.OnDrawItem 4928
Menus.TMenuItem.OnMeasureItem 4929
Menus.TMenuItem.Parent 4919
Menus.TMenuItem.RadioItem 4920
Menus.TMenuItem.Remove 4893
Menus.TMenuItem.RethinkHotkeys 4894
Menus.TMenuItem.RethinkLines 4894
Menus.TMenuItem.SetBreak 4895
Menus.TMenuItem.SetCaption 4895
Menus.TMenuItem.SetChecked 4895
Menus.TMenuItem.SetChildOrder 4896
Menus.TMenuItem.SetDefault 4896
Menus.TMenuItem.SetEnabled 4896
Menus.TMenuItem.SetGroupIndex 4896
Menus.TMenuItem.SetImageIndex 4897
Menus.TMenuItem.SetMenuIndex 4897
Menus.TMenuItem.SetParentComponent 4897
Menus.TMenuItem.SetRadioItem 4897
Menus.TMenuItem.SetShortCut 4897
Menus.TMenuItem.SetVisible 4898
Menus.TMenuItem.ShortCut 4920
Menus.TMenuItem.SubMenuImages 4921
Menus.TMenuItem.UpdateItems 4898
Menus.TMenuItem.Visible 4921
Menus.TMenuItemAutoFlag 4967
Menus.TMenuItemAutoFlag enumeration 4967
Menus.TMenuItemEnumerator 4929
Menus.TMenuItemEnumerator.Create 4930
Menus.TMenuItemEnumerator.Current 4931
Menus.TMenuItemEnumerator.GetCurrent 4930
Menus.TMenuItemEnumerator.MoveNext 4930
Menus.TMenuItemStack 4931
Menus.TMenuItemStack.ClearItem 4932
Menus.TMenuMeasureItemEvent 4973
Menus.TMenuMeasureItemEvent type 4973
Menus.TPopupAlignment 4968
Menus.TPopupAlignment enumeration 4968
Menus.TPopupList 4932
Menus.TPopupList.Add 4934
Menus.TPopupList.FWindow 4933
Menus.TPopupList.MainWndProc 4934
Menus.TPopupList.Remove 4934
Menus.TPopupList.Window 4935
Menus.TPopupList.WndProc 4935
Menus.TPopupMenu 4935
Menus.TPopupMenu.Alignment 4941
Menus.TPopupMenu.AutoHotkeys 4941
Menus.TPopupMenu.AutoLineReduction 4942
Menus.TPopupMenu.AutoPopup 4942
Menus.TPopupMenu.BiDiMode 4943
Menus.TPopupMenu.Create 4938
Menus.TPopupMenu.Destroy 4938
Menus.TPopupMenu.DoPopup 4939
Menus.TPopupMenu.HelpContext 4944
Menus.TPopupMenu.Images 4944
Menus.TPopupMenu.MenuAnimation 4945
Menus.TPopupMenu.OnChange 4949
Menus.TPopupMenu.OnPopup 4949
Menus.TPopupMenu.OwnerDraw 4945
Menus.TPopupMenu.ParentBiDiMode 4946
Menus.TPopupMenu.Popup 4939
Menus.TPopupMenu.PopupComponent 4946
Menus.TPopupMenu.PopupPoint 4948
Menus.TPopupMenu.SetPopupPoint 4940
Menus.TPopupMenu.TrackButton 4948
Menus.TPopupMenu.UseRightToLeftAlignment 4940
Menus.TTrackButton 4968
Menus.TTrackButton enumeration 4968
Menus.ValidMenuHotkeys 4975
Menus.ValidMenuHotkeys variable 4975

Messages 4979
Messages namespace 4979
Constants 5077
Functions 4991
Structs, Records, Enums 4995
Types 5050
Messages.BM_CLICK 5084
Messages.BM_CLICK constant 5084
Messages.BM_GETCHECK 5085
Messages.BM_GETCHECK constant 5085
Messages.BM_GETIMAGE 5085
Messages.BM_GETIMAGE constant 5085
Messages.BM_GETSTATE 5085
Messages.BM_GETSTATE constant 5085
Messages.BM_SETCHECK 5086
Messages.BM_SETCHECK constant 5086
Messages.BM_SETIMAGE 5086
Messages.BM_SETIMAGE constant 5086
Messages.BM_SETSTATE 5086
Messages.BM_SETSTATE constant 5086
Messages.BM_SETSTYLE 5087
Messages.BM_SETSTYLE constant 5087
Messages.BN_CLICKED 5087
Messages.BN_CLICKED constant 5087
Messages.BN_DBLCLK 5087
Messages.BN_DBLCLK constant 5087
Messages.BN_DISABLE 5088
Messages.BN_DISABLE constant 5088
Messages.BN_DOUBLECLICKED 5088
Messages.BN_DOUBLECLICKED constant 5088
Messages.BN_HILITE 5088
Messages.BN_HILITE constant 5088
Messages.BN_KILLFOCUS 5089
Messages.BN_KILLFOCUS constant 5089
Messages.BN_PAINT 5089
Messages.BN_PAINT constant 5089
Messages.BN_PUSHED 5089
Messages.BN_PUSHED constant 5089
Messages.BN_SETFOCUS 5090
Messages.BN_SETFOCUS constant 5090
Messages.BN_UNHILITE 5090
Messages.BN_UNHILITE constant 5090
Messages.BN_UNPUSHED 5090
Messages.BN_UNPUSHED constant 5090
Messages.CB_ADDSTRING 5091
Messages.CB_ADDSTRING constant 5091
Messages.CB_DELETESTRING 5091
Messages.CB_DELETESTRING constant 5091
Messages.CB_DIR 5091
Messages.CB_DIR constant 5091
Messages.CB_FINDSTRING 5092
Messages.CB_FINDSTRING constant 5092
Messages.CB_FINDSTRINGEXACT 5092
Messages.CB_FINDSTRINGEXACT constant 5092
Messages.CB_GETCOUNT 5092
Messages.CB_GETCOUNT constant 5092
Messages.CB_GETCURSEL 5093
Messages.CB_GETCURSEL constant 5093
Messages.CB_GETDROPPEDCONTROLRECT 5093
Messages.CB_GETDROPPEDCONTROLRECT constant 5093
Messages.CB_GETDROPPEDSTATE 5093
Messages.CB_GETDROPPEDSTATE constant 5093
Messages.CB_GETDROPPEDWIDTH 5094
Messages.CB_GETDROPPEDWIDTH constant 5094
Messages.CB_GETEDITSEL 5094
Messages.CB_GETEDITSEL constant 5094
Messages.CB_GETEXTENDEDUI 5094
Messages.CB_GETEXTENDEDUI constant 5094
Messages.CB_GETHORIZONTALEXTENT 5095
Messages.CB_GETHORIZONTALEXTENT constant 5095
Messages.CB_GETITEMDATA 5095
Messages.CB_GETITEMDATA constant 5095
Messages.CB_GETITEMHEIGHT 5095
Messages.CB_GETITEMHEIGHT constant 5095
Messages.CB_GETLBTEXT 5096
Messages.CB_GETLBTEXT constant 5096
Messages.CB_GETLBTEXTLEN 5096
Messages.CB_GETLBTEXTLEN constant 5096

Messages.CB_GETLOCALE 5096
Messages.CB_GETLOCALE constant 5096
Messages.CB_GETTOPINDEX 5097
Messages.CB_GETTOPINDEX constant 5097
Messages.CB_INITSTORAGE 5097
Messages.CB_INITSTORAGE constant 5097
Messages.CB_INSERTSTRING 5097
Messages.CB_INSERTSTRING constant 5097
Messages.CB_LIMITTEXT 5098
Messages.CB_LIMITTEXT constant 5098
Messages.CB_MSGMAX 5098
Messages.CB_MSGMAX constant 5098
Messages.CB_RESETCONTENT 5098
Messages.CB_RESETCONTENT constant 5098
Messages.CB_SELECTSTRING 5099
Messages.CB_SELECTSTRING constant 5099
Messages.CB_SETCURSEL 5099
Messages.CB_SETCURSEL constant 5099
Messages.CB_SETDROPPEDWIDTH 5099
Messages.CB_SETDROPPEDWIDTH constant 5099
Messages.CB_SETEDITSEL 5100
Messages.CB_SETEDITSEL constant 5100
Messages.CB_SETEXTENDEDUI 5100
Messages.CB_SETEXTENDEDUI constant 5100
Messages.CB_SETHORIZONTALEXTENT 5100
Messages.CB_SETHORIZONTALEXTENT constant 5100
Messages.CB_SETITEMDATA 5101
Messages.CB_SETITEMDATA constant 5101
Messages.CB_SETITEMHEIGHT 5101
Messages.CB_SETITEMHEIGHT constant 5101
Messages.CB_SETLOCALE 5101
Messages.CB_SETLOCALE constant 5101
Messages.CB_SETTOPINDEX 5102
Messages.CB_SETTOPINDEX constant 5102
Messages.CB_SHOWDROPDOWN 5102
Messages.CB_SHOWDROPDOWN constant 5102
Messages.CBN_CLOSEUP 5102
Messages.CBN_CLOSEUP constant 5102
Messages.CBN_DBLCLK 5103
Messages.CBN_DBLCLK constant 5103
Messages.CBN_DROPDOWN 5103
Messages.CBN_DROPDOWN constant 5103
Messages.CBN_EDITCHANGE 5103
Messages.CBN_EDITCHANGE constant 5103
Messages.CBN_EDITUPDATE 5104
Messages.CBN_EDITUPDATE constant 5104
Messages.CBN_ERRSPACE 5104
Messages.CBN_ERRSPACE constant 5104
Messages.CBN_KILLFOCUS 5104
Messages.CBN_KILLFOCUS constant 5104
Messages.CBN_SELCHANGE 5105
Messages.CBN_SELCHANGE constant 5105
Messages.CBN_SELENDCANCEL 5105
Messages.CBN_SELENDCANCEL constant 5105
Messages.CBN_SELENDOK 5105
Messages.CBN_SELENDOK constant 5105
Messages.CBN_SETFOCUS 5106
Messages.CBN_SETFOCUS constant 5106
Messages.DM_GETDEFID 5106
Messages.DM_GETDEFID constant 5106
Messages.DM_REPOSITION 5106
Messages.DM_REPOSITION constant 5106
Messages.DM_SETDEFID 5107
Messages.DM_SETDEFID constant 5107
Messages.EM_CANUNDO 5107
Messages.EM_CANUNDO constant 5107
Messages.EM_CHARFROMPOS 5107
Messages.EM_CHARFROMPOS constant 5107
Messages.EM_EMPTYUNDOBUFFER 5108
Messages.EM_EMPTYUNDOBUFFER constant 5108
Messages.EM_FMTLINES 5108
Messages.EM_FMTLINES constant 5108
Messages.EM_GETFIRSTVISIBLELINE 5108
Messages.EM_GETFIRSTVISIBLELINE constant 5108
Messages.EM_GETHANDLE 5109
Messages.EM_GETHANDLE constant 5109
Messages.EM_GETIMESTATUS 5109
Messages.EM_GETIMESTATUS constant 5109

Messages.EM_GETLIMITTEXT 5109
Messages.EM_GETLIMITTEXT constant 5109
Messages.EM_GETLINE 5110
Messages.EM_GETLINE constant 5110
Messages.EM_GETLINECOUNT 5110
Messages.EM_GETLINECOUNT constant 5110
Messages.EM_GETMARGINS 5110
Messages.EM_GETMARGINS constant 5110
Messages.EM_GETMODIFY 5111
Messages.EM_GETMODIFY constant 5111
Messages.EM_GETPASSWORDCHAR 5111
Messages.EM_GETPASSWORDCHAR constant 5111
Messages.EM_GETRECT 5111
Messages.EM_GETRECT constant 5111
Messages.EM_GETSEL 5112
Messages.EM_GETSEL constant 5112
Messages.EM_GETTHUMB 5112
Messages.EM_GETTHUMB constant 5112
Messages.EM_GETWORDBREAKPROC 5112
Messages.EM_GETWORDBREAKPROC constant 5112
Messages.EM_LIMITTEXT 5113
Messages.EM_LIMITTEXT constant 5113
Messages.EM_LINEFROMCHAR 5113
Messages.EM_LINEFROMCHAR constant 5113
Messages.EM_LINEINDEX 5113
Messages.EM_LINEINDEX constant 5113
Messages.EM_LINELENGTH 5114
Messages.EM_LINELENGTH constant 5114
Messages.EM_LINESCROLL 5114
Messages.EM_LINESCROLL constant 5114
Messages.EM_POSFROMCHAR 5114
Messages.EM_POSFROMCHAR constant 5114
Messages.EM_REPLACESEL 5115
Messages.EM_REPLACESEL constant 5115
Messages.EM_SCROLL 5115
Messages.EM_SCROLL constant 5115
Messages.EM_SCROLLCARET 5115
Messages.EM_SCROLLCARET constant 5115
Messages.EM_SETHANDLE 5116
Messages.EM_SETHANDLE constant 5116
Messages.EM_SETIMESTATUS 5116
Messages.EM_SETIMESTATUS constant 5116
Messages.EM_SETLIMITTEXT 5116
Messages.EM_SETLIMITTEXT constant 5116
Messages.EM_SETMARGINS 5117
Messages.EM_SETMARGINS constant 5117
Messages.EM_SETMODIFY 5117
Messages.EM_SETMODIFY constant 5117
Messages.EM_SETPASSWORDCHAR 5117
Messages.EM_SETPASSWORDCHAR constant 5117
Messages.EM_SETREADONLY 5118
Messages.EM_SETREADONLY constant 5118
Messages.EM_SETRECT 5118
Messages.EM_SETRECT constant 5118
Messages.EM_SETRECTNP 5118
Messages.EM_SETRECTNP constant 5118
Messages.EM_SETSEL 5119
Messages.EM_SETSEL constant 5119
Messages.EM_SETTABSTOPS 5119
Messages.EM_SETTABSTOPS constant 5119
Messages.EM_SETWORDBREAKPROC 5119
Messages.EM_SETWORDBREAKPROC constant 5119
Messages.EM_UNDO 5120
Messages.EM_UNDO constant 5120
Messages.EN_CHANGE 5120
Messages.EN_CHANGE constant 5120
Messages.EN_ERRSPACE 5120
Messages.EN_ERRSPACE constant 5120
Messages.EN_HSCROLL 5121
Messages.EN_HSCROLL constant 5121
Messages.EN_KILLFOCUS 5121
Messages.EN_KILLFOCUS constant 5121
Messages.EN_MAXTEXT 5121
Messages.EN_MAXTEXT constant 5121
Messages.EN_SETFOCUS 5122
Messages.EN_SETFOCUS constant 5122
Messages.EN_UPDATE 5122
Messages.EN_UPDATE constant 5122

Messages.EN_VSCROLL 5122
Messages.EN_VSCROLL constant 5122
Messages.LB_ADDFILE 5123
Messages.LB_ADDFILE constant 5123
Messages.LB_ADDSTRING 5123
Messages.LB_ADDSTRING constant 5123
Messages.LB_DELETESTRING 5123
Messages.LB_DELETESTRING constant 5123
Messages.LB_DIR 5124
Messages.LB_DIR constant 5124
Messages.LB_FINDSTRING 5124
Messages.LB_FINDSTRING constant 5124
Messages.LB_FINDSTRINGEXACT 5124
Messages.LB_FINDSTRINGEXACT constant 5124
Messages.LB_GETANCHORINDEX 5125
Messages.LB_GETANCHORINDEX constant 5125
Messages.LB_GETCARETINDEX 5125
Messages.LB_GETCARETINDEX constant 5125
Messages.LB_GETCOUNT 5125
Messages.LB_GETCOUNT constant 5125
Messages.LB_GETCURSEL 5126
Messages.LB_GETCURSEL constant 5126
Messages.LB_GETHORIZONTALEXTENT 5126
Messages.LB_GETHORIZONTALEXTENT constant 5126
Messages.LB_GETITEMDATA 5126
Messages.LB_GETITEMDATA constant 5126
Messages.LB_GETITEMHEIGHT 5127
Messages.LB_GETITEMHEIGHT constant 5127
Messages.LB_GETITEMRECT 5127
Messages.LB_GETITEMRECT constant 5127
Messages.LB_GETLOCALE 5127
Messages.LB_GETLOCALE constant 5127
Messages.LB_GETSEL 5128
Messages.LB_GETSEL constant 5128
Messages.LB_GETSELCOUNT 5128
Messages.LB_GETSELCOUNT constant 5128
Messages.LB_GETSELITEMS 5128
Messages.LB_GETSELITEMS constant 5128
Messages.LB_GETTEXT 5129
Messages.LB_GETTEXT constant 5129
Messages.LB_GETTEXTLEN 5129
Messages.LB_GETTEXTLEN constant 5129
Messages.LB_GETTOPINDEX 5129
Messages.LB_GETTOPINDEX constant 5129
Messages.LB_INITSTORAGE 5130
Messages.LB_INITSTORAGE constant 5130
Messages.LB_INSERTSTRING 5130
Messages.LB_INSERTSTRING constant 5130
Messages.LB_ITEMFROMPOINT 5130
Messages.LB_ITEMFROMPOINT constant 5130
Messages.LB_MSGMAX 5131
Messages.LB_MSGMAX constant 5131
Messages.LB_RESETCONTENT 5131
Messages.LB_RESETCONTENT constant 5131
Messages.LB_SELECTSTRING 5131
Messages.LB_SELECTSTRING constant 5131
Messages.LB_SELITEMRANGE 5132
Messages.LB_SELITEMRANGE constant 5132
Messages.LB_SELITEMRANGEEX 5132
Messages.LB_SELITEMRANGEEX constant 5132
Messages.LB_SETANCHORINDEX 5132
Messages.LB_SETANCHORINDEX constant 5132
Messages.LB_SETCARETINDEX 5133
Messages.LB_SETCARETINDEX constant 5133
Messages.LB_SETCOLUMNWIDTH 5133
Messages.LB_SETCOLUMNWIDTH constant 5133
Messages.LB_SETCOUNT 5133
Messages.LB_SETCOUNT constant 5133
Messages.LB_SETCURSEL 5134
Messages.LB_SETCURSEL constant 5134
Messages.LB_SETHORIZONTALEXTENT 5134
Messages.LB_SETHORIZONTALEXTENT constant 5134
Messages.LB_SETITEMDATA 5134
Messages.LB_SETITEMDATA constant 5134
Messages.LB_SETITEMHEIGHT 5135
Messages.LB_SETITEMHEIGHT constant 5135
Messages.LB_SETLOCALE 5135
Messages.LB_SETLOCALE constant 5135

Messages.LB_SETSEL 5135
Messages.LB_SETSEL constant 5135
Messages.LB_SETTABSTOPS 5136
Messages.LB_SETTABSTOPS constant 5136
Messages.LB_SETTOPINDEX 5136
Messages.LB_SETTOPINDEX constant 5136
Messages.LBN_DBLCLK 5136
Messages.LBN_DBLCLK constant 5136
Messages.LBN_ERRSPACE 5137
Messages.LBN_ERRSPACE constant 5137
Messages.LBN_KILLFOCUS 5137
Messages.LBN_KILLFOCUS constant 5137
Messages.LBN_SELCANCEL 5137
Messages.LBN_SELCANCEL constant 5137
Messages.LBN_SELCHANGE 5138
Messages.LBN_SELCHANGE constant 5138
Messages.LBN_SETFOCUS 5138
Messages.LBN_SETFOCUS constant 5138
Messages.PMessage 5051
Messages.PMessage type 5051
Messages.PSM_PAGEINFO 5138
Messages.PSM_PAGEINFO constant 5138
Messages.PSM_SHEETINFO 5139
Messages.PSM_SHEETINFO constant 5139
Messages.SBM_ENABLE_ARROWS 5139
Messages.SBM_ENABLE_ARROWS constant 5139
Messages.SBM_GETPOS 5139
Messages.SBM_GETPOS constant 5139
Messages.SBM_GETRANGE 5140
Messages.SBM_GETRANGE constant 5140
Messages.SBM_GETSCROLLBARINFO 5140
Messages.SBM_GETSCROLLBARINFO constant 5140
Messages.SBM_GETSCROLLINFO 5140
Messages.SBM_GETSCROLLINFO constant 5140
Messages.SBM_SETPOS 5141
Messages.SBM_SETPOS constant 5141
Messages.SBM_SETRANGE 5141
Messages.SBM_SETRANGE constant 5141
Messages.SBM_SETRANGEREDRAW 5141
Messages.SBM_SETRANGEREDRAW constant 5141
Messages.SBM_SETSCROLLINFO 5142
Messages.SBM_SETSCROLLINFO constant 5142
Messages.SendGetIntMessage 4991
Messages.SendGetIntMessage function 4991
Messages.SendGetStructMessage 4992
Messages.SendGetStructMessage function 4992
Messages.SendStructMessage 4992
Messages.SendStructMessage function 4992
Messages.SendTextMessage 4993
Messages.SendTextMessage function 4993
Messages.TMessage 4997
Messages.TMessage record 4997
Messages.TMSHMouseWheel 4998
Messages.TMSHMouseWheel record 4998
Messages.TWMActivate 4998
Messages.TWMActivate record 4998
Messages.TWMActivateApp 4999
Messages.TWMActivateApp record 4999
Messages.TWMAskCBFormatName 4999
Messages.TWMAskCBFormatName record 4999
Messages.TWMCancelMode 5052
Messages.TWMCancelMode type 5052
Messages.TWMChangeCBChain 5000
Messages.TWMChangeCBChain record 5000
Messages.TWMChangeUIState 5052
Messages.TWMChangeUIState type 5052
Messages.TWMChar 5052
Messages.TWMChar type 5052
Messages.TWMCharToItem 5000
Messages.TWMCharToItem record 5000
Messages.TWMChildActivate 5053
Messages.TWMChildActivate type 5053
Messages.TWMChooseFont_GetLogFont 5001
Messages.TWMChooseFont_GetLogFont record 5001
Messages.TWMClear 5053
Messages.TWMClear type 5053
Messages.TWMClose 5053
Messages.TWMClose type 5053

Messages.TWMCommand 5001
Messages.TWMCommand record 5001
Messages.TWMCompacting 5002
Messages.TWMCompacting record 5002
Messages.TWMCompareItem 5002
Messages.TWMCompareItem record 5002
Messages.TWMContextMenu 5003
Messages.TWMContextMenu record 5003
Messages.TWMCopy 5054
Messages.TWMCopy type 5054
Messages.TWMCopyData 5003
Messages.TWMCopyData record 5003
Messages.TWMCreate 5004
Messages.TWMCreate record 5004
Messages.TWMCtlColor 5004
Messages.TWMCtlColor record 5004
Messages.TWMCtlColorBtn 5054
Messages.TWMCtlColorBtn type 5054
Messages.TWMCtlColorDlg 5054
Messages.TWMCtlColorDlg type 5054
Messages.TWMCtlColorEdit 5055
Messages.TWMCtlColorEdit type 5055
Messages.TWMCtlColorListbox 5055
Messages.TWMCtlColorListbox type 5055
Messages.TWMCtlColorMsgbox 5055
Messages.TWMCtlColorMsgbox type 5055
Messages.TWMCtlColorScrollbar 5056
Messages.TWMCtlColorScrollbar type 5056
Messages.TWMCtlColorStatic 5056
Messages.TWMCtlColorStatic type 5056
Messages.TWMCut 5056
Messages.TWMCut type 5056
Messages.TWMDDE_Ack 5005
Messages.TWMDDE_Ack record 5005
Messages.TWMDDE_Advise 5005
Messages.TWMDDE_Advise record 5005
Messages.TWMDDE_Data 5006
Messages.TWMDDE_Data record 5006
Messages.TWMDDE_Execute 5006
Messages.TWMDDE_Execute record 5006
Messages.TWMDDE_Initiate 5007
Messages.TWMDDE_Initiate record 5007
Messages.TWMDDE_Poke 5007
Messages.TWMDDE_Poke record 5007
Messages.TWMDDE_Request 5008
Messages.TWMDDE_Request record 5008
Messages.TWMDDE_Terminate 5008
Messages.TWMDDE_Terminate record 5008
Messages.TWMDDE_Unadvise 5009
Messages.TWMDDE_Unadvise record 5009
Messages.TWMDeadChar 5057
Messages.TWMDeadChar type 5057
Messages.TWMDeleteItem 5009
Messages.TWMDeleteItem record 5009
Messages.TWMDestroy 5057
Messages.TWMDestroy type 5057
Messages.TWMDestroyClipboard 5057
Messages.TWMDestroyClipboard type 5057
Messages.TWMDevModeChange 5010
Messages.TWMDevModeChange record 5010
Messages.TWMDisplayChange 5010
Messages.TWMDisplayChange record 5010
Messages.TWMDrawClipboard 5058
Messages.TWMDrawClipboard type 5058
Messages.TWMDrawItem 5011
Messages.TWMDrawItem record 5011
Messages.TWMDropFiles 5011
Messages.TWMDropFiles record 5011
Messages.TWMEnable 5012
Messages.TWMEnable record 5012
Messages.TWMEndSession 5012
Messages.TWMEndSession record 5012
Messages.TWMEnterIdle 5013
Messages.TWMEnterIdle record 5013
Messages.TWMEnterMenuLoop 5013
Messages.TWMEnterMenuLoop record 5013
Messages.TWMEraseBkgnd 5014
Messages.TWMEraseBkgnd record 5014

Messages.TWMExitMenuLoop 5058
Messages.TWMExitMenuLoop type 5058
Messages.TWMFontChange 5058
Messages.TWMFontChange type 5058
Messages.TWMGetDlgCode 5059
Messages.TWMGetDlgCode type 5059
Messages.TWMGetFont 5059
Messages.TWMGetFont type 5059
Messages.TWMGetHotKey 5059
Messages.TWMGetHotKey type 5059
Messages.TWMGetIcon 5014
Messages.TWMGetIcon record 5014
Messages.TWMGetMinMaxInfo 5015
Messages.TWMGetMinMaxInfo record 5015
Messages.TWMGetText 5015
Messages.TWMGetText record 5015
Messages.TWMGetTextLength 5060
Messages.TWMGetTextLength type 5060
Messages.TWMHelp 5016
Messages.TWMHelp record 5016
Messages.TWMHotKey 5016
Messages.TWMHotKey record 5016
Messages.TWMHScroll 5060
Messages.TWMHScroll type 5060
Messages.TWMHScrollClipboard 5017
Messages.TWMHScrollClipboard record 5017
Messages.TWMIconEraseBkgnd 5060
Messages.TWMIconEraseBkgnd type 5060
Messages.TWMInitDialog 5017
Messages.TWMInitDialog record 5017
Messages.TWMInitMenu 5018
Messages.TWMInitMenu record 5018
Messages.TWMInitMenuPopup 5018
Messages.TWMInitMenuPopup record 5018
Messages.TWMKey 5019
Messages.TWMKey record 5019
Messages.TWMKeyDown 5061
Messages.TWMKeyDown type 5061
Messages.TWMKeyUp 5061
Messages.TWMKeyUp type 5061
Messages.TWMKillFocus 5019
Messages.TWMKillFocus record 5019
Messages.TWMLButtonDblClk 5061
Messages.TWMLButtonDblClk type 5061
Messages.TWMLButtonDown 5062
Messages.TWMLButtonDown type 5062
Messages.TWMLButtonUp 5062
Messages.TWMLButtonUp type 5062
Messages.TWMMButtonDblClk 5062
Messages.TWMMButtonDblClk type 5062
Messages.TWMMButtonDown 5063
Messages.TWMMButtonDown type 5063
Messages.TWMMButtonUp 5063
Messages.TWMMButtonUp type 5063
Messages.TWMMDIActivate 5020
Messages.TWMMDIActivate record 5020
Messages.TWMMDICascade 5020
Messages.TWMMDICascade record 5020
Messages.TWMMDICreate 5021
Messages.TWMMDICreate record 5021
Messages.TWMMDIDestroy 5021
Messages.TWMMDIDestroy record 5021
Messages.TWMMDIGetActive 5063
Messages.TWMMDIGetActive type 5063
Messages.TWMMDIIconArrange 5064
Messages.TWMMDIIconArrange type 5064
Messages.TWMMDIMaximize 5022
Messages.TWMMDIMaximize record 5022
Messages.TWMMDINext 5022
Messages.TWMMDINext record 5022
Messages.TWMMDIRefreshMenu 5064
Messages.TWMMDIRefreshMenu type 5064
Messages.TWMMDIRestore 5023
Messages.TWMMDIRestore record 5023
Messages.TWMMDISetMenu 5023
Messages.TWMMDISetMenu record 5023
Messages.TWMMDITile 5024
Messages.TWMMDITile record 5024

Messages.TWMMeasureItem 5024
Messages.TWMMeasureItem record 5024
Messages.TWMMenuChar 5025
Messages.TWMMenuChar record 5025
Messages.TWMMenuSelect 5025
Messages.TWMMenuSelect record 5025
Messages.TWMMouse 5026
Messages.TWMMouse record 5026
Messages.TWMMouseActivate 5026
Messages.TWMMouseActivate record 5026
Messages.TWMMouseMove 5064
Messages.TWMMouseMove type 5064
Messages.TWMMouseWheel 5027
Messages.TWMMouseWheel record 5027
Messages.TWMMove 5027
Messages.TWMMove record 5027
Messages.TWMMoving 5028
Messages.TWMMoving record 5028
Messages.TWMNCActivate 5028
Messages.TWMNCActivate record 5028
Messages.TWMNCCalcSize 5029
Messages.TWMNCCalcSize record 5029
Messages.TWMNCCreate 5029
Messages.TWMNCCreate record 5029
Messages.TWMNCDestroy 5065
Messages.TWMNCDestroy type 5065
Messages.TWMNCHitMessage 5030
Messages.TWMNCHitMessage record 5030
Messages.TWMNCHitTest 5030
Messages.TWMNCHitTest record 5030
Messages.TWMNCLButtonDblClk 5065
Messages.TWMNCLButtonDblClk type 5065
Messages.TWMNCLButtonDown 5065
Messages.TWMNCLButtonDown type 5065
Messages.TWMNCLButtonUp 5066
Messages.TWMNCLButtonUp type 5066
Messages.TWMNCMButtonDblClk 5066
Messages.TWMNCMButtonDblClk type 5066
Messages.TWMNCMButtonDown 5066
Messages.TWMNCMButtonDown type 5066
Messages.TWMNCMButtonUp 5067
Messages.TWMNCMButtonUp type 5067
Messages.TWMNCMouseMove 5067
Messages.TWMNCMouseMove type 5067
Messages.TWMNCPaint 5031
Messages.TWMNCPaint record 5031
Messages.TWMNCRButtonDblClk 5067
Messages.TWMNCRButtonDblClk type 5067
Messages.TWMNCRButtonDown 5068
Messages.TWMNCRButtonDown type 5068
Messages.TWMNCRButtonUp 5068
Messages.TWMNCRButtonUp type 5068
Messages.TWMNextDlgCtl 5031
Messages.TWMNextDlgCtl record 5031
Messages.TWMNoParams 5032
Messages.TWMNoParams record 5032
Messages.TWMNotify 5032
Messages.TWMNotify record 5032
Messages.TWMNotifyFormat 5033
Messages.TWMNotifyFormat record 5033
Messages.TWMPaint 5033
Messages.TWMPaint record 5033
Messages.TWMPaintClipboard 5034
Messages.TWMPaintClipboard record 5034
Messages.TWMPaintIcon 5068
Messages.TWMPaintIcon type 5068
Messages.TWMPaletteChanged 5034
Messages.TWMPaletteChanged record 5034
Messages.TWMPaletteIsChanging 5035
Messages.TWMPaletteIsChanging record 5035
Messages.TWMParentNotify 5035
Messages.TWMParentNotify record 5035
Messages.TWMPaste 5069
Messages.TWMPaste type 5069
Messages.TWMPower 5036
Messages.TWMPower record 5036
Messages.TWMPrint 5036
Messages.TWMPrint record 5036

Messages.TWMPrintClient 5069
Messages.TWMPrintClient type 5069
Messages.TWMQueryDragIcon 5069
Messages.TWMQueryDragIcon type 5069
Messages.TWMQueryEndSession 5037
Messages.TWMQueryEndSession record 5037
Messages.TWMQueryNewPalette 5070
Messages.TWMQueryNewPalette type 5070
Messages.TWMQueryOpen 5070
Messages.TWMQueryOpen type 5070
Messages.TWMQueryUIState 5070
Messages.TWMQueryUIState type 5070
Messages.TWMQueueSync 5071
Messages.TWMQueueSync type 5071
Messages.TWMQuit 5037
Messages.TWMQuit record 5037
Messages.TWMRButtonDblClk 5071
Messages.TWMRButtonDblClk type 5071
Messages.TWMRButtonDown 5071
Messages.TWMRButtonDown type 5071
Messages.TWMRButtonUp 5072
Messages.TWMRButtonUp type 5072
Messages.TWMRenderAllFormats 5072
Messages.TWMRenderAllFormats type 5072
Messages.TWMRenderFormat 5038
Messages.TWMRenderFormat record 5038
Messages.TWMScroll 5038
Messages.TWMScroll record 5038
Messages.TWMSetCursor 5039
Messages.TWMSetCursor record 5039
Messages.TWMSetFocus 5039
Messages.TWMSetFocus record 5039
Messages.TWMSetFont 5040
Messages.TWMSetFont record 5040
Messages.TWMSetHotKey 5040
Messages.TWMSetHotKey record 5040
Messages.TWMSetIcon 5041
Messages.TWMSetIcon record 5041
Messages.TWMSetRedraw 5041
Messages.TWMSetRedraw record 5041
Messages.TWMSetText 5042
Messages.TWMSetText record 5042
Messages.TWMSettingChange 5042
Messages.TWMSettingChange record 5042
Messages.TWMShowWindow 5043
Messages.TWMShowWindow record 5043
Messages.TWMSize 5043
Messages.TWMSize record 5043
Messages.TWMSizeClipboard 5044
Messages.TWMSizeClipboard record 5044
Messages.TWMSpoolerStatus 5044
Messages.TWMSpoolerStatus record 5044
Messages.TWMStyleChange 5045
Messages.TWMStyleChange record 5045
Messages.TWMStyleChanged 5072
Messages.TWMStyleChanged type 5072
Messages.TWMStyleChanging 5073
Messages.TWMStyleChanging type 5073
Messages.TWMSysChar 5073
Messages.TWMSysChar type 5073
Messages.TWMSysColorChange 5073
Messages.TWMSysColorChange type 5073
Messages.TWMSysCommand 5045
Messages.TWMSysCommand record 5045
Messages.TWMSysDeadChar 5046
Messages.TWMSysDeadChar record 5046
Messages.TWMSysKeyDown 5074
Messages.TWMSysKeyDown type 5074
Messages.TWMSysKeyUp 5074
Messages.TWMSysKeyUp type 5074
Messages.TWMSystemError 5046
Messages.TWMSystemError record 5046
Messages.TWMTimeChange 5074
Messages.TWMTimeChange type 5074
Messages.TWMTimer 5047
Messages.TWMTimer record 5047
Messages.TWMUIState 5047
Messages.TWMUIState record 5047

Messages.TWMUndo 5075
Messages.TWMUndo type 5075
Messages.TWMUpdateUIState 5075
Messages.TWMUpdateUIState type 5075
Messages.TWMVKeyToItem 5075
Messages.TWMVKeyToItem type 5075
Messages.TWMVScroll 5076
Messages.TWMVScroll type 5076
Messages.TWMVScrollClipboard 5048
Messages.TWMVScrollClipboard record 5048
Messages.TWMWindowPosChanged 5076
Messages.TWMWindowPosChanged type 5076
Messages.TWMWindowPosChanging 5076
Messages.TWMWindowPosChanging type 5076
Messages.TWMWindowPosMsg 5048
Messages.TWMWindowPosMsg record 5048
Messages.TWMWinIniChange 5049
Messages.TWMWinIniChange record 5049
Messages.WM_ACTIVATE 5142
Messages.WM_ACTIVATE constant 5142
Messages.WM_ACTIVATEAPP 5142
Messages.WM_ACTIVATEAPP constant 5142
Messages.WM_APP 5143
Messages.WM_APP constant 5143
Messages.WM_APPCOMMAND 5143
Messages.WM_APPCOMMAND constant 5143
Messages.WM_ASKCBFORMATNAME 5143
Messages.WM_ASKCBFORMATNAME constant 5143
Messages.WM_CANCELJOURNAL 5144
Messages.WM_CANCELJOURNAL constant 5144
Messages.WM_CANCELMODE 5144
Messages.WM_CANCELMODE constant 5144
Messages.WM_CAPTURECHANGED 5144
Messages.WM_CAPTURECHANGED constant 5144
Messages.WM_CHANGECBCHAIN 5145
Messages.WM_CHANGECBCHAIN constant 5145
Messages.WM_CHANGEUISTATE 5145
Messages.WM_CHANGEUISTATE constant 5145
Messages.WM_CHAR 5145
Messages.WM_CHAR constant 5145
Messages.WM_CHARTOITEM 5146
Messages.WM_CHARTOITEM constant 5146
Messages.WM_CHILDACTIVATE 5146
Messages.WM_CHILDACTIVATE constant 5146
Messages.WM_CLEAR 5146
Messages.WM_CLEAR constant 5146
Messages.WM_CLOSE 5147
Messages.WM_CLOSE constant 5147
Messages.WM_COALESCE_FIRST 5147
Messages.WM_COALESCE_FIRST constant 5147
Messages.WM_COALESCE_LAST 5147
Messages.WM_COALESCE_LAST constant 5147
Messages.WM_COMMAND 5148
Messages.WM_COMMAND constant 5148
Messages.WM_COMMNOTIFY 5148
Messages.WM_COMMNOTIFY constant 5148
Messages.WM_COMPACTING 5148
Messages.WM_COMPACTING constant 5148
Messages.WM_COMPAREITEM 5149
Messages.WM_COMPAREITEM constant 5149
Messages.WM_CONTEXTMENU 5149
Messages.WM_CONTEXTMENU constant 5149
Messages.WM_COPY 5149
Messages.WM_COPY constant 5149
Messages.WM_COPYDATA 5150
Messages.WM_COPYDATA constant 5150
Messages.WM_CREATE 5150
Messages.WM_CREATE constant 5150
Messages.WM_CTLCOLOR 5150
Messages.WM_CTLCOLOR constant 5150
Messages.WM_CTLCOLORBTN 5151
Messages.WM_CTLCOLORBTN constant 5151
Messages.WM_CTLCOLORDLG 5151
Messages.WM_CTLCOLORDLG constant 5151
Messages.WM_CTLCOLOREDIT 5151
Messages.WM_CTLCOLOREDIT constant 5151
Messages.WM_CTLCOLORLISTBOX 5152
Messages.WM_CTLCOLORLISTBOX constant 5152

Messages.WM_CTLCOLORMSGBOX 5152
Messages.WM_CTLCOLORMSGBOX constant 5152
Messages.WM_CTLCOLORSCROLLBAR 5152
Messages.WM_CTLCOLORSCROLLBAR constant 5152
Messages.WM_CTLCOLORSTATIC 5153
Messages.WM_CTLCOLORSTATIC constant 5153
Messages.WM_CUT 5153
Messages.WM_CUT constant 5153
Messages.WM_DDE_ACK 5153
Messages.WM_DDE_ACK constant 5153
Messages.WM_DDE_ADVISE 5154
Messages.WM_DDE_ADVISE constant 5154
Messages.WM_DDE_DATA 5154
Messages.WM_DDE_DATA constant 5154
Messages.WM_DDE_EXECUTE 5154
Messages.WM_DDE_EXECUTE constant 5154
Messages.WM_DDE_FIRST 5155
Messages.WM_DDE_FIRST constant 5155
Messages.WM_DDE_INITIATE 5155
Messages.WM_DDE_INITIATE constant 5155
Messages.WM_DDE_LAST 5155
Messages.WM_DDE_LAST constant 5155
Messages.WM_DDE_POKE 5156
Messages.WM_DDE_POKE constant 5156
Messages.WM_DDE_REQUEST 5156
Messages.WM_DDE_REQUEST constant 5156
Messages.WM_DDE_TERMINATE 5156
Messages.WM_DDE_TERMINATE constant 5156
Messages.WM_DDE_UNADVISE 5157
Messages.WM_DDE_UNADVISE constant 5157
Messages.WM_DEADCHAR 5157
Messages.WM_DEADCHAR constant 5157
Messages.WM_DELETEITEM 5157
Messages.WM_DELETEITEM constant 5157
Messages.WM_DESTROY 5158
Messages.WM_DESTROY constant 5158
Messages.WM_DESTROYCLIPBOARD 5158
Messages.WM_DESTROYCLIPBOARD constant 5158
Messages.WM_DEVICECHANGE 5158
Messages.WM_DEVICECHANGE constant 5158
Messages.WM_DEVMODECHANGE 5159
Messages.WM_DEVMODECHANGE constant 5159
Messages.WM_DISPLAYCHANGE 5159
Messages.WM_DISPLAYCHANGE constant 5159
Messages.WM_DRAWCLIPBOARD 5159
Messages.WM_DRAWCLIPBOARD constant 5159
Messages.WM_DRAWITEM 5160
Messages.WM_DRAWITEM constant 5160
Messages.WM_DROPFILES 5160
Messages.WM_DROPFILES constant 5160
Messages.WM_DWMCOLORIZATIONCOLORCHANGED 5160
Messages.WM_DWMCOLORIZATIONCOLORCHANGED constant 5160
Messages.WM_DWMCOMPOSITIONCHANGED 5161
Messages.WM_DWMCOMPOSITIONCHANGED constant 5161
Messages.WM_DWMNCRENDERINGCHANGED 5161
Messages.WM_DWMNCRENDERINGCHANGED constant 5161
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE 5161
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE constant 5161
Messages.WM_ENABLE 5162
Messages.WM_ENABLE constant 5162
Messages.WM_ENDSESSION 5162
Messages.WM_ENDSESSION constant 5162
Messages.WM_ENTERIDLE 5162
Messages.WM_ENTERIDLE constant 5162
Messages.WM_ENTERMENULOOP 5163
Messages.WM_ENTERMENULOOP constant 5163
Messages.WM_ENTERSIZEMOVE 5163
Messages.WM_ENTERSIZEMOVE constant 5163
Messages.WM_ERASEBKGND 5163
Messages.WM_ERASEBKGND constant 5163
Messages.WM_EXITMENULOOP 5164
Messages.WM_EXITMENULOOP constant 5164
Messages.WM_EXITSIZEMOVE 5164
Messages.WM_EXITSIZEMOVE constant 5164
Messages.WM_FONTCHANGE 5164
Messages.WM_FONTCHANGE constant 5164

Messages.WM_GETDLGCODE 5165
Messages.WM_GETDLGCODE constant 5165
Messages.WM_GETFONT 5165
Messages.WM_GETFONT constant 5165
Messages.WM_GETHOTKEY 5165
Messages.WM_GETHOTKEY constant 5165
Messages.WM_GETICON 5166
Messages.WM_GETICON constant 5166
Messages.WM_GETMINMAXINFO 5166
Messages.WM_GETMINMAXINFO constant 5166
Messages.WM_GETOBJECT 5166
Messages.WM_GETOBJECT constant 5166
Messages.WM_GETTEXT 5167
Messages.WM_GETTEXT constant 5167
Messages.WM_GETTEXTLENGTH 5167
Messages.WM_GETTEXTLENGTH constant 5167
Messages.WM_HANDHELDFIRST 5167
Messages.WM_HANDHELDFIRST constant 5167
Messages.WM_HANDHELDLAST 5168
Messages.WM_HANDHELDLAST constant 5168
Messages.WM_HELP 5168
Messages.WM_HELP constant 5168
Messages.WM_HOTKEY 5168
Messages.WM_HOTKEY constant 5168
Messages.WM_HSCROLL 5169
Messages.WM_HSCROLL constant 5169
Messages.WM_HSCROLLCLIPBOARD 5169
Messages.WM_HSCROLLCLIPBOARD constant 5169
Messages.WM_ICONERASEBKGND 5169
Messages.WM_ICONERASEBKGND constant 5169
Messages.WM_IME_CHAR 5170
Messages.WM_IME_CHAR constant 5170
Messages.WM_IME_COMPOSITION 5170
Messages.WM_IME_COMPOSITION constant 5170
Messages.WM_IME_COMPOSITIONFULL 5170
Messages.WM_IME_COMPOSITIONFULL constant 5170
Messages.WM_IME_CONTROL 5171
Messages.WM_IME_CONTROL constant 5171
Messages.WM_IME_ENDCOMPOSITION 5171
Messages.WM_IME_ENDCOMPOSITION constant 5171
Messages.WM_IME_KEYDOWN 5171
Messages.WM_IME_KEYDOWN constant 5171
Messages.WM_IME_KEYLAST 5172
Messages.WM_IME_KEYLAST constant 5172
Messages.WM_IME_KEYUP 5172
Messages.WM_IME_KEYUP constant 5172
Messages.WM_IME_NOTIFY 5172
Messages.WM_IME_NOTIFY constant 5172
Messages.WM_IME_REQUEST 5173
Messages.WM_IME_REQUEST constant 5173
Messages.WM_IME_SELECT 5173
Messages.WM_IME_SELECT constant 5173
Messages.WM_IME_SETCONTEXT 5173
Messages.WM_IME_SETCONTEXT constant 5173
Messages.WM_IME_STARTCOMPOSITION 5174
Messages.WM_IME_STARTCOMPOSITION constant 5174
Messages.WM_INITDIALOG 5174
Messages.WM_INITDIALOG constant 5174
Messages.WM_INITMENU 5174
Messages.WM_INITMENU constant 5174
Messages.WM_INITMENUPOPUP 5175
Messages.WM_INITMENUPOPUP constant 5175
Messages.WM_INPUT 5175
Messages.WM_INPUT constant 5175
Messages.WM_INPUTLANGCHANGE 5175
Messages.WM_INPUTLANGCHANGE constant 5175
Messages.WM_INPUTLANGCHANGEREQUEST 5176
Messages.WM_INPUTLANGCHANGEREQUEST constant 5176
Messages.WM_KEYDOWN 5176
Messages.WM_KEYDOWN constant 5176
Messages.WM_KEYFIRST 5176
Messages.WM_KEYFIRST constant 5176
Messages.WM_KEYLAST 5177
Messages.WM_KEYLAST constant 5177
Messages.WM_KEYUP 5177
Messages.WM_KEYUP constant 5177
Messages.WM_KILLFOCUS 5177
Messages.WM_KILLFOCUS constant 5177

Messages.WM_LBUTTONDBLCLK 5178
Messages.WM_LBUTTONDBLCLK constant 5178
Messages.WM_LBUTTONDOWN 5178
Messages.WM_LBUTTONDOWN constant 5178
Messages.WM_LBUTTONUP 5178
Messages.WM_LBUTTONUP constant 5178
Messages.WM_MBUTTONDBLCLK 5179
Messages.WM_MBUTTONDBLCLK constant 5179
Messages.WM_MBUTTONDOWN 5179
Messages.WM_MBUTTONDOWN constant 5179
Messages.WM_MBUTTONUP 5179
Messages.WM_MBUTTONUP constant 5179
Messages.WM_MDIACTIVATE 5180
Messages.WM_MDIACTIVATE constant 5180
Messages.WM_MDICASCADE 5180
Messages.WM_MDICASCADE constant 5180
Messages.WM_MDICREATE 5180
Messages.WM_MDICREATE constant 5180
Messages.WM_MDIDESTROY 5181
Messages.WM_MDIDESTROY constant 5181
Messages.WM_MDIGETACTIVE 5181
Messages.WM_MDIGETACTIVE constant 5181
Messages.WM_MDIICONARRANGE 5181
Messages.WM_MDIICONARRANGE constant 5181
Messages.WM_MDIMAXIMIZE 5182
Messages.WM_MDIMAXIMIZE constant 5182
Messages.WM_MDINEXT 5182
Messages.WM_MDINEXT constant 5182
Messages.WM_MDIREFRESHMENU 5182
Messages.WM_MDIREFRESHMENU constant 5182
Messages.WM_MDIRESTORE 5183
Messages.WM_MDIRESTORE constant 5183
Messages.WM_MDISETMENU 5183
Messages.WM_MDISETMENU constant 5183
Messages.WM_MDITILE 5183
Messages.WM_MDITILE constant 5183
Messages.WM_MEASUREITEM 5184
Messages.WM_MEASUREITEM constant 5184
Messages.WM_MENUCHAR 5184
Messages.WM_MENUCHAR constant 5184
Messages.WM_MENUCOMMAND 5184
Messages.WM_MENUCOMMAND constant 5184
Messages.WM_MENUDRAG 5185
Messages.WM_MENUDRAG constant 5185
Messages.WM_MENUGETOBJECT 5185
Messages.WM_MENUGETOBJECT constant 5185
Messages.WM_MENURBUTTONUP 5185
Messages.WM_MENURBUTTONUP constant 5185
Messages.WM_MENUSELECT 5186
Messages.WM_MENUSELECT constant 5186
Messages.WM_MOUSEACTIVATE 5186
Messages.WM_MOUSEACTIVATE constant 5186
Messages.WM_MOUSEFIRST 5186
Messages.WM_MOUSEFIRST constant 5186
Messages.WM_MOUSEHOVER 5187
Messages.WM_MOUSEHOVER constant 5187
Messages.WM_MOUSELAST 5187
Messages.WM_MOUSELAST constant 5187
Messages.WM_MOUSELEAVE 5187
Messages.WM_MOUSELEAVE constant 5187
Messages.WM_MOUSEMOVE 5188
Messages.WM_MOUSEMOVE constant 5188
Messages.WM_MOUSEWHEEL 5188
Messages.WM_MOUSEWHEEL constant 5188
Messages.WM_MOVE 5188
Messages.WM_MOVE constant 5188
Messages.WM_MOVING 5189
Messages.WM_MOVING constant 5189
Messages.WM_NCACTIVATE 5189
Messages.WM_NCACTIVATE constant 5189
Messages.WM_NCCALCSIZE 5189
Messages.WM_NCCALCSIZE constant 5189
Messages.WM_NCCREATE 5190
Messages.WM_NCCREATE constant 5190
Messages.WM_NCDESTROY 5190
Messages.WM_NCDESTROY constant 5190
Messages.WM_NCHITTEST 5190
Messages.WM_NCHITTEST constant 5190

Messages.WM_NCLBUTTONDBLCLK 5191
Messages.WM_NCLBUTTONDBLCLK constant 5191
Messages.WM_NCLBUTTONDOWN 5191
Messages.WM_NCLBUTTONDOWN constant 5191
Messages.WM_NCLBUTTONUP 5191
Messages.WM_NCLBUTTONUP constant 5191
Messages.WM_NCMBUTTONDBLCLK 5192
Messages.WM_NCMBUTTONDBLCLK constant 5192
Messages.WM_NCMBUTTONDOWN 5192
Messages.WM_NCMBUTTONDOWN constant 5192
Messages.WM_NCMBUTTONUP 5192
Messages.WM_NCMBUTTONUP constant 5192
Messages.WM_NCMOUSEHOVER 5193
Messages.WM_NCMOUSEHOVER constant 5193
Messages.WM_NCMOUSELEAVE 5193
Messages.WM_NCMOUSELEAVE constant 5193
Messages.WM_NCMOUSEMOVE 5193
Messages.WM_NCMOUSEMOVE constant 5193
Messages.WM_NCPAINT 5194
Messages.WM_NCPAINT constant 5194
Messages.WM_NCRBUTTONDBLCLK 5194
Messages.WM_NCRBUTTONDBLCLK constant 5194
Messages.WM_NCRBUTTONDOWN 5194
Messages.WM_NCRBUTTONDOWN constant 5194
Messages.WM_NCRBUTTONUP 5195
Messages.WM_NCRBUTTONUP constant 5195
Messages.WM_NCXBUTTONDBLCLK 5195
Messages.WM_NCXBUTTONDBLCLK constant 5195
Messages.WM_NCXBUTTONDOWN 5195
Messages.WM_NCXBUTTONDOWN constant 5195
Messages.WM_NCXBUTTONUP 5196
Messages.WM_NCXBUTTONUP constant 5196
Messages.WM_NEXTDLGCTL 5196
Messages.WM_NEXTDLGCTL constant 5196
Messages.WM_NEXTMENU 5196
Messages.WM_NEXTMENU constant 5196
Messages.WM_NOTIFY 5197
Messages.WM_NOTIFY constant 5197
Messages.WM_NOTIFYFORMAT 5197
Messages.WM_NOTIFYFORMAT constant 5197
Messages.WM_NULL 5197
Messages.WM_NULL constant 5197
Messages.WM_PAINT 5198
Messages.WM_PAINT constant 5198
Messages.WM_PAINTCLIPBOARD 5198
Messages.WM_PAINTCLIPBOARD constant 5198
Messages.WM_PAINTICON 5198
Messages.WM_PAINTICON constant 5198
Messages.WM_PALETTECHANGED 5199
Messages.WM_PALETTECHANGED constant 5199
Messages.WM_PALETTEISCHANGING 5199
Messages.WM_PALETTEISCHANGING constant 5199
Messages.WM_PARENTNOTIFY 5199
Messages.WM_PARENTNOTIFY constant 5199
Messages.WM_PASTE 5200
Messages.WM_PASTE constant 5200
Messages.WM_PENWINFIRST 5200
Messages.WM_PENWINFIRST constant 5200
Messages.WM_PENWINLAST 5200
Messages.WM_PENWINLAST constant 5200
Messages.WM_POWER 5201
Messages.WM_POWER constant 5201
Messages.WM_POWERBROADCAST 5201
Messages.WM_POWERBROADCAST constant 5201
Messages.WM_PRINT 5201
Messages.WM_PRINT constant 5201
Messages.WM_PRINTCLIENT 5202
Messages.WM_PRINTCLIENT constant 5202
Messages.WM_QUERYDRAGICON 5202
Messages.WM_QUERYDRAGICON constant 5202
Messages.WM_QUERYENDSESSION 5202
Messages.WM_QUERYENDSESSION constant 5202
Messages.WM_QUERYNEWPALETTE 5203
Messages.WM_QUERYNEWPALETTE constant 5203
Messages.WM_QUERYOPEN 5203
Messages.WM_QUERYOPEN constant 5203
Messages.WM_QUERYUISTATE 5203
Messages.WM_QUERYUISTATE constant 5203

Messages.WM_QUEUESYNC 5204
Messages.WM_QUEUESYNC constant 5204
Messages.WM_QUIT 5204
Messages.WM_QUIT constant 5204
Messages.WM_RBUTTONDBLCLK 5204
Messages.WM_RBUTTONDBLCLK constant 5204
Messages.WM_RBUTTONDOWN 5205
Messages.WM_RBUTTONDOWN constant 5205
Messages.WM_RBUTTONUP 5205
Messages.WM_RBUTTONUP constant 5205
Messages.WM_RENDERALLFORMATS 5205
Messages.WM_RENDERALLFORMATS constant 5205
Messages.WM_RENDERFORMAT 5206
Messages.WM_RENDERFORMAT constant 5206
Messages.WM_SETCURSOR 5206
Messages.WM_SETCURSOR constant 5206
Messages.WM_SETFOCUS 5206
Messages.WM_SETFOCUS constant 5206
Messages.WM_SETFONT 5207
Messages.WM_SETFONT constant 5207
Messages.WM_SETHOTKEY 5207
Messages.WM_SETHOTKEY constant 5207
Messages.WM_SETICON 5207
Messages.WM_SETICON constant 5207
Messages.WM_SETREDRAW 5208
Messages.WM_SETREDRAW constant 5208
Messages.WM_SETTEXT 5208
Messages.WM_SETTEXT constant 5208
Messages.WM_SETTINGCHANGE 5208
Messages.WM_SETTINGCHANGE constant 5208
Messages.WM_SHOWWINDOW 5209
Messages.WM_SHOWWINDOW constant 5209
Messages.WM_SIZE 5209
Messages.WM_SIZE constant 5209
Messages.WM_SIZECLIPBOARD 5209
Messages.WM_SIZECLIPBOARD constant 5209
Messages.WM_SIZING 5210
Messages.WM_SIZING constant 5210
Messages.WM_SPOOLERSTATUS 5210
Messages.WM_SPOOLERSTATUS constant 5210
Messages.WM_STYLECHANGED 5210
Messages.WM_STYLECHANGED constant 5210
Messages.WM_STYLECHANGING 5211
Messages.WM_STYLECHANGING constant 5211
Messages.WM_SYSCHAR 5211
Messages.WM_SYSCHAR constant 5211
Messages.WM_SYSCOLORCHANGE 5211
Messages.WM_SYSCOLORCHANGE constant 5211
Messages.WM_SYSCOMMAND 5212
Messages.WM_SYSCOMMAND constant 5212
Messages.WM_SYSDEADCHAR 5212
Messages.WM_SYSDEADCHAR constant 5212
Messages.WM_SYSKEYDOWN 5212
Messages.WM_SYSKEYDOWN constant 5212
Messages.WM_SYSKEYUP 5213
Messages.WM_SYSKEYUP constant 5213
Messages.WM_SYSTEMERROR 5213
Messages.WM_SYSTEMERROR constant 5213
Messages.WM_TABLET_FIRST 5213
Messages.WM_TABLET_FIRST constant 5213
Messages.WM_TABLET_LAST 5214
Messages.WM_TABLET_LAST constant 5214
Messages.WM_TCARD 5214
Messages.WM_TCARD constant 5214
Messages.WM_THEMECHANGED 5214
Messages.WM_THEMECHANGED constant 5214
Messages.WM_TIMECHANGE 5215
Messages.WM_TIMECHANGE constant 5215
Messages.WM_TIMER 5215
Messages.WM_TIMER constant 5215
Messages.WM_UNDO 5215
Messages.WM_UNDO constant 5215
Messages.WM_UNICHAR 5216
Messages.WM_UNICHAR constant 5216
Messages.WM_UNINITMENUPOPUP 5216
Messages.WM_UNINITMENUPOPUP constant 5216
Messages.WM_UPDATEUISTATE 5216
Messages.WM_UPDATEUISTATE constant 5216

Messages.WM_USER 5217
Messages.WM_USER constant 5217
Messages.WM_USERCHANGED 5217
Messages.WM_USERCHANGED constant 5217
Messages.WM_VKEYTOITEM 5217
Messages.WM_VKEYTOITEM constant 5217
Messages.WM_VSCROLL 5218
Messages.WM_VSCROLL constant 5218
Messages.WM_VSCROLLCLIPBOARD 5218
Messages.WM_VSCROLLCLIPBOARD constant 5218
Messages.WM_WINDOWPOSCHANGED 5218
Messages.WM_WINDOWPOSCHANGED constant 5218
Messages.WM_WINDOWPOSCHANGING 5219
Messages.WM_WINDOWPOSCHANGING constant 5219
Messages.WM_WININICHANGE 5219
Messages.WM_WININICHANGE constant 5219
Messages.WM_WTSSESSION_CHANGE 5219
Messages.WM_WTSSESSION_CHANGE constant 5219
MetadataOnly enumeration member 4018
mgColored enumeration member 5823
mgDisabled enumeration member 5823
mgEnabled enumeration member 5823
Midas 5221
Midas namespace 5221
Constants 5232
Interfaces 5222
Midas.IAppServer 5222
Midas.IAppServer.AS_ApplyUpdates 5223
Midas.IAppServer.AS_DataRequest 5224
Midas.IAppServer.AS_Execute 5224
Midas.IAppServer.AS_GetParams 5225
Midas.IAppServer.AS_GetProviderNames 5226
Midas.IAppServer.AS_GetRecords 5227
Midas.IAppServer.AS_RowRequest 5228
Midas.IAppServerDisp 5229
Midas.IAppServerDisp.AS_ApplyUpdates 5230
Midas.IAppServerDisp.AS_DataRequest 5230
Midas.IAppServerDisp.AS_Execute 5230
Midas.IAppServerDisp.AS_GetParams 5230
Midas.IAppServerDisp.AS_GetProviderNames 5231
Midas.IAppServerDisp.AS_GetRecords 5231
Midas.IAppServerDisp.AS_RowRequest 5231
Midas.IID_IAppServer 5232
Midas.IID_IAppServer constant 5232
Midas.LIBID_Midas 5232
Midas.LIBID_Midas constant 5232
MidasCon 5233
MidasCon namespace 5233
Classes 5234
Structs, Records, Enums 5241
MidasCon.TConnectType 5241
MidasCon.TConnectType enumeration 5241
MidasCon.TMidasConnection 5234
MidasCon.TMidasConnection.ConnectType 5238
MidasCon.TMidasConnection.Create 5235
MidasCon.TMidasConnection.DoConnect 5236
MidasCon.TMidasConnection.DoDisconnect 5237
MidasCon.TMidasConnection.ServerPort 5238
MidasCon.TMidasConnection.SetConnectType 5237
MidasCon.TMidasConnection.SetUseBroker 5237
MidasCon.TMidasConnection.UseBroker 5239
MidasCon.TRemoteServer 5239
MidItems 5243
MidItems namespace 5243
Classes 5248
Constants 5686
Functions 5676
Interfaces 5644
Structs, Records, Enums 5681
Types 5684
MidItems.AddBoolAttrib 5676
MidItems.AddBoolAttrib function 5676
MidItems.AddCustomAttrib 5676
MidItems.AddCustomAttrib function 5676
MidItems.AddIntAttrib 5677
MidItems.AddIntAttrib function 5677
MidItems.AddQuotedAttrib 5677
MidItems.AddQuotedAttrib function 5677

MidItems.AddStringAttrib 5677
MidItems.AddStringAttrib function 5677
MidItems.CanAddClassHelper 5678
MidItems.CanAddClassHelper function 5678
MidItems.CreateDefaultButtonClasses 5678
MidItems.CreateDefaultButtonClasses function 5678
MidItems.DeclareSubmitForm 5678
MidItems.DeclareSubmitForm function 5678
MidItems.FindDispatcher 5679
MidItems.FindDispatcher function 5679
MidItems.FindProducer 5679
MidItems.FindProducer function 5679
MidItems.FormatColumnData 5679
MidItems.FormatColumnData function 5679
MidItems.FormatColumnHeading 5680
MidItems.FormatColumnHeading function 5680
MidItems.FormMethodNames 5686
MidItems.FormMethodNames constant 5686
MidItems.GetItemValuesFromDataSet 5680
MidItems.GetItemValuesFromDataSet function 5680
MidItems.HTMLAlignValues 5687
MidItems.HTMLAlignValues constant 5687
MidItems.HTMLVAlignValues 5687
MidItems.HTMLVAlignValues constant 5687
MidItems.IDataSetComponent 5644
MidItems.IDataSetComponent.DataSet 5646
MidItems.IDataSetComponent.GetDataSet 5645
MidItems.IDataSetComponent.GetXMLBroker 5645
MidItems.IDataSetComponent.GetXMLDataSet 5646
MidItems.IDataSetComponent.XMLBroker 5647
MidItems.IDataSetComponent.XMLDataSet 5647
MidItems.IDataSetField 5648
MidItems.IDataSetField.FieldName 5649
MidItems.IDataSetField.GetFieldName 5648
MidItems.IDataSetField.GetRowSetFieldAttributes 5649
MidItems.IDataSetField.SetFieldName 5649
MidItems.IDataSetFields 5650
MidItems.IDataSetFields.GetFieldsList 5651
MidItems.IDataSetFields.GetVisibleFields 5651
MidItems.IDataSetFields.HasStatusField 5651
MidItems.IDataSetFields.IsFieldInUse 5651
MidItems.IDataSetFields.ValidateDataSet 5651
MidItems.IDataSetFields.VisibleFields 5652
MidItems.IFormatColumn 5652
MidItems.IFormatColumn.FormatColumnData 5653
MidItems.IFormatColumn.FormatColumnHeading 5653
MidItems.IHTMLField 5653
MidItems.IHTMLField.GetHTMLControlName 5654
MidItems.IHTMLField.HTMLControlName 5655
MidItems.IHTMLForm 5655
MidItems.IHTMLForm.GetHTMLFormName 5656
MidItems.IHTMLForm.GetHTMLFormVarName 5656
MidItems.IHTMLForm.HTMLFormName 5657
MidItems.IHTMLForm.HTMLFormVarName 5657
MidItems.IInetXWebDataSet 5657
MidItems.IInetXWebDataSet.AddOnDataChange 5659
MidItems.IInetXWebDataSet.FetchParams 5659
MidItems.IInetXWebDataSet.FieldCount 5659
MidItems.IInetXWebDataSet.Fields 5660
MidItems.IInetXWebDataSet.ParamCount 5660
MidItems.IInetXWebDataSet.Params 5660
MidItems.IInetXWebDataSet.RemoveOnDataChange 5661
MidItems.IMidasWebDataSet 5684
MidItems.IMidasWebDataSet type 5684
MidItems.IQueryField 5661
MidItems.IQueryField.GetParamName 5662
MidItems.IQueryField.GetText 5662
MidItems.IQueryField.ParamName 5663
MidItems.IQueryField.SetParamName 5662
MidItems.IQueryField.SetText 5663
MidItems.IQueryField.Text 5663
MidItems.IQueryFields 5664
MidItems.IQueryFields.GetParamsList 5664
MidItems.IQueryFields.IsParamInUse 5665
MidItems.IQueryFields.ValidateDataSet 5665
MidItems.IQueryForm 5665
MidItems.IRestoreDefaults 5665
MidItems.IRestoreDefaults.RestoreDefaults 5666

MidItems.IStatusField 5666
MidItems.IValidateField 5667
MidItems.IValidateField.ValidateField 5667
MidItems.IValidateFields 5668
MidItems.IValidateFields.EnableValidateFields 5669
MidItems.IValidateFields.GetEnableValidateFields 5669
MidItems.IValidateFields.SetEnableValidateFields 5669
MidItems.IValidateFields.ValidateFields 5669
MidItems.IWebDisplay 5670
MidItems.IWebDisplay.GetIsMultipleRecordView 5670
MidItems.IWebDisplay.IsMultipleRecordView 5671
MidItems.IWebDisplayReference 5671
MidItems.IWebDisplayReference.GetWebDisplayComponent 5672
MidItems.IWebDisplayReference.WebDisplayComponent 5672
MidItems.IXMLDisplay 5673
MidItems.IXMLDisplay.GetIsMultipleRecordView 5673
MidItems.IXMLDisplay.GetXMLDisplayName 5674
MidItems.IXMLDisplay.GetXMLRowSetName 5674
MidItems.IXMLDisplay.IsMultipleRecordView 5674
MidItems.IXMLDisplay.XMLDisplayName 5675
MidItems.IXMLDisplay.XMLRowSetName 5675
MidItems.sActionAttr 5687
MidItems.sActionAttr constant 5687
MidItems.sAlignAttr 5688
MidItems.sAlignAttr constant 5688
MidItems.sBgColorAttr 5688
MidItems.sBgColorAttr constant 5688
MidItems.sBorderAttr 5688
MidItems.sBorderAttr constant 5688
MidItems.sCellPaddingAttr 5689
MidItems.sCellPaddingAttr constant 5689
MidItems.sCellSpacingAttr 5689
MidItems.sCellSpacingAttr constant 5689
MidItems.sCheckedAttr 5689
MidItems.sCheckedAttr constant 5689
MidItems.sClassAttr 5690
MidItems.sClassAttr constant 5690
MidItems.sColsAttr 5690
MidItems.sColsAttr constant 5690
MidItems.sColSpanAttr 5690
MidItems.sColSpanAttr constant 5690
MidItems.sEncTypeAttr 5691
MidItems.sEncTypeAttr constant 5691
MidItems.sImageHeightAttr 5691
MidItems.sImageHeightAttr constant 5691
MidItems.sImageWidthAttr 5691
MidItems.sImageWidthAttr constant 5691
MidItems.sMaxLengthAttr 5692
MidItems.sMaxLengthAttr constant 5692
MidItems.sMethodAttr 5692
MidItems.sMethodAttr constant 5692
MidItems.sMultipleAttr 5692
MidItems.sMultipleAttr constant 5692
MidItems.sNameAttr 5693
MidItems.sNameAttr constant 5693
MidItems.sOnClickAttr 5693
MidItems.sOnClickAttr constant 5693
MidItems.sReadOnlyAttr 5693
MidItems.sReadOnlyAttr constant 5693
MidItems.sRowsAttr 5694
MidItems.sRowsAttr constant 5694
MidItems.sSelectedAttr 5694
MidItems.sSelectedAttr constant 5694
MidItems.sSizeAttr 5694
MidItems.sSizeAttr constant 5694
MidItems.sStyleAttr 5695
MidItems.sStyleAttr constant 5695
MidItems.sTabIndexAttr 5695
MidItems.sTabIndexAttr constant 5695
MidItems.sTypeAttr 5695
MidItems.sTypeAttr constant 5695
MidItems.sVAlignAttr 5696
MidItems.sVAlignAttr constant 5696
MidItems.sValueAttr 5696
MidItems.sValueAttr constant 5696
MidItems.sWidthAttr 5696
MidItems.sWidthAttr constant 5696
MidItems.sWrapAttr 5697

MidItems.sWrapAttr constant 5697
MidItems.TApplyUpdatesButton 5250
MidItems.TApplyUpdatesButton.AddElements 5252
MidItems.TApplyUpdatesButton.Caption 5253
MidItems.TApplyUpdatesButton.Create 5251
MidItems.TApplyUpdatesButton.Custom 5254
MidItems.TApplyUpdatesButton.GetSubComponents 5252
MidItems.TApplyUpdatesButton.ImplAddElements 5252
MidItems.TApplyUpdatesButton.ImplContent 5252
MidItems.TApplyUpdatesButton.Style 5254
MidItems.TApplyUpdatesButton.StyleRule 5254
MidItems.TApplyUpdatesButton.XMLBroker 5255
MidItems.TApplyUpdatesButton.XMLUseParent 5255
MidItems.TCaptionAttributes 5256
MidItems.TCaptionAttributes.Align 5257
MidItems.TCaptionAttributes.Create 5257
MidItems.TCaptionAttributes.Custom 5258
MidItems.TCaptionAttributes.Parent 5258
MidItems.TCaptionAttributes.Style 5258
MidItems.TCaptionAttributes.StyleRule 5259
MidItems.TCaptionAttributes.VAlign 5259
MidItems.TCaptionPosition 5681
MidItems.TCaptionPosition enumeration 5681
MidItems.TCustomDataForm 5260
MidItems.TCustomDataForm.GetHTMLFormTag 5262
MidItems.TCustomDataForm.ImplCanAddClass 5262
MidItems.TCustomDataForm.ImplContent 5262
MidItems.TCustomDataGrid 5263
MidItems.TCustomDataGrid.Create 5266
MidItems.TCustomDataGrid.Destroy 5266
MidItems.TCustomDataGrid.DisplayRows 5269
MidItems.TCustomDataGrid.FormatTable 5266
MidItems.TCustomDataGrid.HeadingAttributes 5269
MidItems.TCustomDataGrid.ImplAddElements 5267
MidItems.TCustomDataGrid.ImplCanAddClass 5267
MidItems.TCustomDataGrid.ImplContent 5267
MidItems.TCustomDataGrid.ImplIsMultipleRecordView 5268
MidItems.TCustomDataGrid.RowAttributes 5270
MidItems.TCustomDataGrid.SetHeadingAttributes 5268
MidItems.TCustomDataGrid.SetRowAttributes 5268
MidItems.TCustomDataGrid.SetTableAttributes 5269
MidItems.TCustomDataGrid.TableAttributes 5270
MidItems.TCustomDataNavigator 5271
MidItems.TCustomDataNavigator.Custom 5275
MidItems.TCustomDataNavigator.GetDefaultButtons 5273
MidItems.TCustomDataNavigator.ImplCanAddClass 5273
MidItems.TCustomDataNavigator.ImplContent 5273
MidItems.TCustomDataNavigator.ImplGetSubComponents 5274
MidItems.TCustomDataNavigator.Style 5275
MidItems.TCustomDataNavigator.StyleRule 5276
MidItems.TCustomDataNavigator.VisibleButtons 5274
MidItems.TCustomFieldGroup 5276
MidItems.TCustomFieldGroup.Create 5279
MidItems.TCustomFieldGroup.Custom 5281
MidItems.TCustomFieldGroup.Destroy 5279
MidItems.TCustomFieldGroup.GetLayoutAttributes 5280
MidItems.TCustomFieldGroup.ImplContent 5280
MidItems.TCustomFieldGroup.Style 5281
MidItems.TCustomFieldGroup.StyleRule 5282
MidItems.TCustomLayoutGroup 5282
MidItems.TCustomLayoutGroup.Create 5284
MidItems.TCustomLayoutGroup.Custom 5286
MidItems.TCustomLayoutGroup.Destroy 5284
MidItems.TCustomLayoutGroup.DisplayColumns 5286
MidItems.TCustomLayoutGroup.GetLayoutAttributes 5284
MidItems.TCustomLayoutGroup.ImplCanAddClass 5285
MidItems.TCustomLayoutGroup.ImplContent 5285
MidItems.TCustomLayoutGroup.Style 5287
MidItems.TCustomLayoutGroup.StyleRule 5287
MidItems.TCustomQueryButtons 5288
MidItems.TCustomQueryButtons.Custom 5291
MidItems.TCustomQueryButtons.GetDefaultButtons 5289
MidItems.TCustomQueryButtons.ImplAddElements 5290
MidItems.TCustomQueryButtons.ImplCanAddClass 5290
MidItems.TCustomQueryButtons.ImplContent 5290
MidItems.TCustomQueryButtons.Style 5292
MidItems.TCustomQueryButtons.StyleRule 5292
MidItems.TCustomQueryButtons.VisibleButtons 5291

MidItems.TCustomQueryForm 5293
MidItems.TCustomQueryForm.Action 5298
MidItems.TCustomQueryForm.AssignStrings 5295
MidItems.TCustomQueryForm.AssignStringsCallback 5295
MidItems.TCustomQueryForm.Create 5295
MidItems.TCustomQueryForm.GetHTMLFormTag 5296
MidItems.TCustomQueryForm.ImplAddElements 5296
MidItems.TCustomQueryForm.ImplCanAddClass 5296
MidItems.TCustomQueryForm.ImplContent 5297
MidItems.TCustomQueryForm.Method 5298
MidItems.TCustomQueryForm.MethodString 5297
MidItems.TDataForm 5298
MidItems.TDataForm.Custom 5300
MidItems.TDataForm.Style 5301
MidItems.TDataForm.StyleRule 5301
MidItems.TDataGrid 5302
MidItems.TDataGrid.DisplayRows 5305
MidItems.TDataGrid.HeadingAttributes 5305
MidItems.TDataGrid.RowAttributes 5306
MidItems.TDataGrid.TableAttributes 5306
MidItems.TDataNavigator 5307
MidItems.TDataNavigator.Custom 5309
MidItems.TDataNavigator.Style 5309
MidItems.TDataNavigator.StyleRule 5310
MidItems.TDataNavigator.XMLComponent 5310
MidItems.TDataSetButton 5311
MidItems.TDataSetButton.AddElements 5314
MidItems.TDataSetButton.Caption 5316
MidItems.TDataSetButton.Custom 5316
MidItems.TDataSetButton.DefaultCaption 5313
MidItems.TDataSetButton.GetCaption 5314
MidItems.TDataSetButton.GetMethodName 5314
MidItems.TDataSetButton.GetSubComponents 5314
MidItems.TDataSetButton.ImplAddElements 5315
MidItems.TDataSetButton.ImplContent 5315
MidItems.TDataSetButton.Style 5316
MidItems.TDataSetButton.StyleRule 5317
MidItems.TDataSetButton.XMLComponent 5317
MidItems.TDataSetButton.XMLMethodName 5313
MidItems.TDataSetButton.XMLUseParent 5318
MidItems.TDeleteButton 5318
MidItems.TDeleteButton.Create 5320
MidItems.TextAreaWrap 5697
MidItems.TextAreaWrap constant 5697
MidItems.TFieldGroup 5320
MidItems.TFieldGroup.Custom 5324
MidItems.TFieldGroup.ImplAddElements 5323
MidItems.TFieldGroup.ImplCanAddClass 5324
MidItems.TFieldGroup.Style 5325
MidItems.TFieldGroup.StyleRule 5325
MidItems.TFieldRadioGroup 5326
MidItems.TFieldRadioGroup.Caption 5329
MidItems.TFieldRadioGroup.CaptionAttributes 5329
MidItems.TFieldRadioGroup.CaptionPosition 5330
MidItems.TFieldRadioGroup.Custom 5330
MidItems.TFieldRadioGroup.DataSet 5330
MidItems.TFieldRadioGroup.DisplayColumns 5331
MidItems.TFieldRadioGroup.DisplayWidth 5331
MidItems.TFieldRadioGroup.FieldName 5332
MidItems.TFieldRadioGroup.Items 5332
MidItems.TFieldRadioGroup.ItemsField 5333
MidItems.TFieldRadioGroup.ReadOnly 5333
MidItems.TFieldRadioGroup.Style 5333
MidItems.TFieldRadioGroup.StyleRule 5334
MidItems.TFieldRadioGroup.TabIndex 5334
MidItems.TFieldRadioGroup.Values 5334
MidItems.TFieldRadioGroup.ValuesField 5335
MidItems.TFieldSelectOptions 5335
MidItems.TFieldSelectOptions.Caption 5339
MidItems.TFieldSelectOptions.CaptionAttributes 5339
MidItems.TFieldSelectOptions.CaptionPosition 5339
MidItems.TFieldSelectOptions.Custom 5340
MidItems.TFieldSelectOptions.DataSet 5340
MidItems.TFieldSelectOptions.DisplayRows 5341
MidItems.TFieldSelectOptions.FieldName 5341
MidItems.TFieldSelectOptions.Items 5342
MidItems.TFieldSelectOptions.ItemsField 5342
MidItems.TFieldSelectOptions.Style 5342

MidItems.TFieldSelectOptions.StyleRule 5343
MidItems.TFieldSelectOptions.TabIndex 5343
MidItems.TFieldSelectOptions.Values 5343
MidItems.TFieldSelectOptions.ValuesField 5344
MidItems.TFieldStatus 5344
MidItems.TFieldStatus.Caption 5347
MidItems.TFieldStatus.CaptionAttributes 5347
MidItems.TFieldStatus.CaptionPosition 5347
MidItems.TFieldStatus.Custom 5348
MidItems.TFieldStatus.Style 5348
MidItems.TFieldStatus.StyleRule 5349
MidItems.TFieldText 5349
MidItems.TFieldText.Caption 5352
MidItems.TFieldText.CaptionAttributes 5352
MidItems.TFieldText.CaptionPosition 5352
MidItems.TFieldText.Custom 5353
MidItems.TFieldText.DisplayWidth 5353
MidItems.TFieldText.FieldName 5354
MidItems.TFieldText.ReadOnly 5354
MidItems.TFieldText.Style 5354
MidItems.TFieldText.StyleRule 5355
MidItems.TFieldText.TabIndex 5355
MidItems.TFieldTextArea 5355
MidItems.TFieldTextArea.Caption 5358
MidItems.TFieldTextArea.CaptionAttributes 5359
MidItems.TFieldTextArea.CaptionPosition 5359
MidItems.TFieldTextArea.Custom 5359
MidItems.TFieldTextArea.DisplayRows 5360
MidItems.TFieldTextArea.DisplayWidth 5360
MidItems.TFieldTextArea.FieldName 5360
MidItems.TFieldTextArea.ReadOnly 5361
MidItems.TFieldTextArea.Style 5361
MidItems.TFieldTextArea.StyleRule 5362
MidItems.TFieldTextArea.TabIndex 5362
MidItems.TFieldTextArea.Wrap 5362
MidItems.TFirstButton 5363
MidItems.TFirstButton.Create 5365
MidItems.TFormLayout 5365
MidItems.TFormLayout.BreakButtons 5372
MidItems.TFormLayout.ColumnCount 5372
MidItems.TFormLayout.Create 5366
MidItems.TFormLayout.EndButtons 5367
MidItems.TFormLayout.EndFields 5367
MidItems.TFormLayout.EndLayout 5367
MidItems.TFormLayout.EndRow 5368
MidItems.TFormLayout.EndTable 5368
MidItems.TFormLayout.ImplLayoutButton 5368
MidItems.TFormLayout.ImplLayoutField 5369
MidItems.TFormLayout.ImplLayoutLabelAndField 5369
MidItems.TFormLayout.ImplLayoutTable 5370
MidItems.TFormLayout.NextColumn 5370
MidItems.TFormLayout.StartButtons 5370
MidItems.TFormLayout.StartFields 5371
MidItems.TFormLayout.StartRow 5371
MidItems.TFormLayout.StartTable 5371
MidItems.TFormLayout.TableHeader 5372
MidItems.TFormMethod 5682
MidItems.TFormMethod enumeration 5682
MidItems.TGridAttributes 5373
MidItems.TGridAttributes.Align 5375
MidItems.TGridAttributes.AssignTo 5374
MidItems.TGridAttributes.BgColor 5376
MidItems.TGridAttributes.Border 5376
MidItems.TGridAttributes.CellPadding 5377
MidItems.TGridAttributes.CellSpacing 5377
MidItems.TGridAttributes.Create 5374
MidItems.TGridAttributes.Custom 5377
MidItems.TGridAttributes.Parent 5378
MidItems.TGridAttributes.Style 5378
MidItems.TGridAttributes.StyleRule 5379
MidItems.TGridRowAttributes 5379
MidItems.TGridRowAttributes.Align 5382
MidItems.TGridRowAttributes.AssignTo 5381
MidItems.TGridRowAttributes.BgColor 5382
MidItems.TGridRowAttributes.Create 5380
MidItems.TGridRowAttributes.Custom 5382
MidItems.TGridRowAttributes.Parent 5383
MidItems.TGridRowAttributes.Style 5383

MidItems.TGridRowAttributes.StyleRule 5384
MidItems.TGridRowAttributes.VAlign 5384
MidItems.TInsertButton 5385
MidItems.TInsertButton.Create 5386
MidItems.TLastButton 5387
MidItems.TLastButton.Create 5388
MidItems.TLayoutGroup 5389
MidItems.TLayoutGroup.Custom 5391
MidItems.TLayoutGroup.DisplayColumns 5391
MidItems.TLayoutGroup.Style 5391
MidItems.TLayoutGroup.StyleRule 5392
MidItems.TLayoutState 5682
MidItems.TLayoutState enumeration 5682
MidItems.TNextButton 5392
MidItems.TNextButton.Create 5394
MidItems.TNextPageButton 5394
MidItems.TNextPageButton.Create 5396
MidItems.TPostButton 5396
MidItems.TPostButton.Create 5398
MidItems.TPriorButton 5399
MidItems.TPriorButton.Create 5400
MidItems.TPriorPageButton 5401
MidItems.TPriorPageButton.Create 5403
MidItems.TQueryButton 5403
MidItems.TQueryButton.Caption 5409
MidItems.TQueryButton.Create 5404
MidItems.TQueryButton.Custom 5409
MidItems.TQueryButton.DefaultCaption 5405
MidItems.TQueryButton.EventContent 5406
MidItems.TQueryButton.GetCaption 5406
MidItems.TQueryButton.GetHTMLControlName 5406
MidItems.TQueryButton.GetHTMLForm 5407
MidItems.TQueryButton.GetInputType 5407
MidItems.TQueryButton.GetXMLComponent 5407
MidItems.TQueryButton.ImplContent 5407
MidItems.TQueryButton.InputType 5405
MidItems.TQueryButton.Notification 5408
MidItems.TQueryButton.ParentXMLComponent 5408
MidItems.TQueryButton.SetXMLComponent 5408
MidItems.TQueryButton.SetXMLUseParent 5409
MidItems.TQueryButton.Style 5410
MidItems.TQueryButton.StyleRule 5410
MidItems.TQueryButton.XMLComponent 5411
MidItems.TQueryButton.XMLUseParent 5411
MidItems.TQueryButtons 5412
MidItems.TQueryButtons.Custom 5413
MidItems.TQueryButtons.Style 5414
MidItems.TQueryButtons.StyleRule 5414
MidItems.TQueryFieldGroup 5415
MidItems.TQueryFieldGroup.Custom 5419
MidItems.TQueryFieldGroup.FindParam 5418
MidItems.TQueryFieldGroup.GetParamsList 5418
MidItems.TQueryFieldGroup.ImplCanAddClass 5419
MidItems.TQueryFieldGroup.IsParamInUse 5419
MidItems.TQueryFieldGroup.Style 5420
MidItems.TQueryFieldGroup.StyleRule 5420
MidItems.TQueryForm 5421
MidItems.TQueryForm.Action 5423
MidItems.TQueryForm.Custom 5423
MidItems.TQueryForm.Method 5424
MidItems.TQueryForm.Style 5424
MidItems.TQueryForm.StyleRule 5424
MidItems.TQueryRadioGroup 5425
MidItems.TQueryRadioGroup.AddAttributes 5428
MidItems.TQueryRadioGroup.Caption 5430
MidItems.TQueryRadioGroup.CaptionAttributes 5430
MidItems.TQueryRadioGroup.CaptionPosition 5431
MidItems.TQueryRadioGroup.Custom 5431
MidItems.TQueryRadioGroup.DataSet 5431
MidItems.TQueryRadioGroup.DisplayColumns 5432
MidItems.TQueryRadioGroup.DisplayWidth 5432
MidItems.TQueryRadioGroup.FieldName 5433
MidItems.TQueryRadioGroup.GetCheckIndex 5428
MidItems.TQueryRadioGroup.GetText 5429
MidItems.TQueryRadioGroup.IsQueryField 5429
MidItems.TQueryRadioGroup.Items 5433
MidItems.TQueryRadioGroup.ItemsField 5434
MidItems.TQueryRadioGroup.ParamName 5434

MidItems.TQueryRadioGroup.ReadOnly 5435
MidItems.TQueryRadioGroup.SetText 5429
MidItems.TQueryRadioGroup.Style 5435
MidItems.TQueryRadioGroup.StyleRule 5435
MidItems.TQueryRadioGroup.TabIndex 5436
MidItems.TQueryRadioGroup.Text 5436
MidItems.TQueryRadioGroup.Values 5436
MidItems.TQueryRadioGroup.ValuesField 5437
MidItems.TQuerySelectOptions 5437
MidItems.TQuerySelectOptions.Caption 5442
MidItems.TQuerySelectOptions.CaptionAttributes 5442
MidItems.TQuerySelectOptions.CaptionPosition 5443
MidItems.TQuerySelectOptions.Custom 5443
MidItems.TQuerySelectOptions.DataSet 5444
MidItems.TQuerySelectOptions.DisplayRows 5444
MidItems.TQuerySelectOptions.FieldName 5445
MidItems.TQuerySelectOptions.GetSelectIndex 5441
MidItems.TQuerySelectOptions.GetText 5441
MidItems.TQuerySelectOptions.IsQueryField 5441
MidItems.TQuerySelectOptions.Items 5445
MidItems.TQuerySelectOptions.ItemsField 5446
MidItems.TQuerySelectOptions.ParamName 5446
MidItems.TQuerySelectOptions.SetText 5441
MidItems.TQuerySelectOptions.Style 5447
MidItems.TQuerySelectOptions.StyleRule 5447
MidItems.TQuerySelectOptions.TabIndex 5447
MidItems.TQuerySelectOptions.Text 5448
MidItems.TQuerySelectOptions.Values 5448
MidItems.TQuerySelectOptions.ValuesField 5448
MidItems.TQueryText 5449
MidItems.TQueryText.AddAttributes 5452
MidItems.TQueryText.Caption 5453
MidItems.TQueryText.CaptionAttributes 5454
MidItems.TQueryText.CaptionPosition 5454
MidItems.TQueryText.Create 5451
MidItems.TQueryText.Custom 5455
MidItems.TQueryText.DisplayWidth 5455
MidItems.TQueryText.FieldName 5455
MidItems.TQueryText.GetText 5452
MidItems.TQueryText.IsQueryField 5452
MidItems.TQueryText.MaxWidth 5456
MidItems.TQueryText.ParamName 5456
MidItems.TQueryText.ReadOnly 5457
MidItems.TQueryText.SetText 5453
MidItems.TQueryText.Style 5457
MidItems.TQueryText.StyleRule 5458
MidItems.TQueryText.TabIndex 5458
MidItems.TQueryText.Text 5458
MidItems.TQueryTextArea 5458
MidItems.TQueryTextArea.AddAttributes 5462
MidItems.TQueryTextArea.Caption 5464
MidItems.TQueryTextArea.CaptionAttributes 5464
MidItems.TQueryTextArea.CaptionPosition 5465
MidItems.TQueryTextArea.ControlContent 5462
MidItems.TQueryTextArea.Create 5461
MidItems.TQueryTextArea.Custom 5465
MidItems.TQueryTextArea.Destroy 5461
MidItems.TQueryTextArea.DisplayRows 5466
MidItems.TQueryTextArea.DisplayWidth 5466
MidItems.TQueryTextArea.FieldName 5466
MidItems.TQueryTextArea.GetText 5462
MidItems.TQueryTextArea.IsQueryField 5463
MidItems.TQueryTextArea.Lines 5467
MidItems.TQueryTextArea.ParamName 5467
MidItems.TQueryTextArea.ReadOnly 5468
MidItems.TQueryTextArea.SetLines 5463
MidItems.TQueryTextArea.SetText 5463
MidItems.TQueryTextArea.Style 5468
MidItems.TQueryTextArea.StyleRule 5468
MidItems.TQueryTextArea.TabIndex 5469
MidItems.TQueryTextArea.Wrap 5469
MidItems.TResetQueryButton 5469
MidItems.TResetQueryButton.Create 5471
MidItems.TResetQueryButton.EventContent 5471
MidItems.TSelectOptionsColumn 5472
MidItems.TSelectOptionsColumn.Caption 5475
MidItems.TSelectOptionsColumn.CaptionAttributes 5476
MidItems.TSelectOptionsColumn.Custom 5476

MidItems.TSelectOptionsColumn.DataSet 5477
MidItems.TSelectOptionsColumn.DisplayRows 5477
MidItems.TSelectOptionsColumn.FieldName 5478
MidItems.TSelectOptionsColumn.IsColumn 5475
MidItems.TSelectOptionsColumn.Items 5478
MidItems.TSelectOptionsColumn.ItemsField 5478
MidItems.TSelectOptionsColumn.Style 5479
MidItems.TSelectOptionsColumn.StyleRule 5479
MidItems.TSelectOptionsColumn.TabIndex 5480
MidItems.TSelectOptionsColumn.Values 5480
MidItems.TSelectOptionsColumn.ValuesField 5480
MidItems.TStatusColumn 5481
MidItems.TStatusColumn.Caption 5484
MidItems.TStatusColumn.CaptionAttributes 5484
MidItems.TStatusColumn.Custom 5484
MidItems.TStatusColumn.IsColumn 5483
MidItems.TStatusColumn.Style 5485
MidItems.TStatusColumn.StyleRule 5485
MidItems.TSubmitQueryButton 5486
MidItems.TSubmitQueryButton.Create 5487
MidItems.TSubmitQueryButton.EventContent 5488
MidItems.TTextAreaColumn 5488
MidItems.TTextAreaColumn.Caption 5491
MidItems.TTextAreaColumn.CaptionAttributes 5492
MidItems.TTextAreaColumn.Custom 5492
MidItems.TTextAreaColumn.DisplayRows 5493
MidItems.TTextAreaColumn.DisplayWidth 5493
MidItems.TTextAreaColumn.FieldName 5493
MidItems.TTextAreaColumn.IsColumn 5491
MidItems.TTextAreaColumn.ReadOnly 5494
MidItems.TTextAreaColumn.Style 5494
MidItems.TTextAreaColumn.StyleRule 5494
MidItems.TTextAreaColumn.TabIndex 5495
MidItems.TTextAreaColumn.Wrap 5495
MidItems.TTextAreaWrap 5683
MidItems.TTextAreaWrap enumeration 5683
MidItems.TTextColumn 5495
MidItems.TTextColumn.Caption 5499
MidItems.TTextColumn.CaptionAttributes 5499
MidItems.TTextColumn.Custom 5500
MidItems.TTextColumn.DisplayWidth 5500
MidItems.TTextColumn.FieldName 5500
MidItems.TTextColumn.IsColumn 5498
MidItems.TTextColumn.ReadOnly 5501
MidItems.TTextColumn.Style 5501
MidItems.TTextColumn.StyleRule 5502
MidItems.TTextColumn.TabIndex 5502
MidItems.TUndoButton 5502
MidItems.TUndoButton.Create 5504
MidItems.TWebButton 5504
MidItems.TWebButton.Caption 5508
MidItems.TWebButton.Content 5506
MidItems.TWebButton.Create 5506
MidItems.TWebButton.Custom 5509
MidItems.TWebButton.GetCaption 5507
MidItems.TWebButton.GetLayoutAttributes 5507
MidItems.TWebButton.ImplContent 5507
MidItems.TWebButton.SetCaption 5508
MidItems.TWebButton.Style 5509
MidItems.TWebButton.StyleRule 5510
MidItems.TWebButtonClass 5684
MidItems.TWebButtonClass type 5684
MidItems.TWebControlGroup 5510
MidItems.TWebControlGroup.AddElements 5513
MidItems.TWebControlGroup.CanAddClass 5513
MidItems.TWebControlGroup.ChangeFieldControls 5514
MidItems.TWebControlGroup.Content 5514
MidItems.TWebControlGroup.Create 5512
MidItems.TWebControlGroup.DefaultWebComponents 5518
MidItems.TWebControlGroup.Destroy 5512
MidItems.TWebControlGroup.GetDefaultWebComponents 5515
MidItems.TWebControlGroup.GetLayoutAttributes 5515
MidItems.TWebControlGroup.GetSubComponents 5516
MidItems.TWebControlGroup.ImplAddElements 5516
MidItems.TWebControlGroup.ImplCanAddClass 5516
MidItems.TWebControlGroup.ImplContent 5517
MidItems.TWebControlGroup.ImplGetSubComponents 5517
MidItems.TWebControlGroup.SetDefaultWebComponents 5518

MidItems.TWebControlGroup.WebFieldControls 5519
MidItems.TWebDataDisplay 5519
MidItems.TWebDataDisplay.Caption 5529
MidItems.TWebDataDisplay.CaptionAttributes 5530
MidItems.TWebDataDisplay.CaptionPosition 5530
MidItems.TWebDataDisplay.Content 5523
MidItems.TWebDataDisplay.ControlContent 5523
MidItems.TWebDataDisplay.Create 5521
MidItems.TWebDataDisplay.Custom 5530
MidItems.TWebDataDisplay.Destroy 5522
MidItems.TWebDataDisplay.EventContent 5523
MidItems.TWebDataDisplay.FormatCaption 5524
MidItems.TWebDataDisplay.GetCaption 5524
MidItems.TWebDataDisplay.GetHTMLControlName 5524
MidItems.TWebDataDisplay.GetHTMLForm 5524
MidItems.TWebDataDisplay.GetLayoutAttributes 5525
MidItems.TWebDataDisplay.GetXmlDisplayName 5525
MidItems.TWebDataDisplay.GetXmlRowSetName 5526
MidItems.TWebDataDisplay.ImplContent 5526
MidItems.TWebDataDisplay.ImplGetHTMLControlName 5527
MidItems.TWebDataDisplay.IsColumn 5527
MidItems.TWebDataDisplay.IsQueryField 5528
MidItems.TWebDataDisplay.LabelContent 5528
MidItems.TWebDataDisplay.LayoutAttributes 5531
MidItems.TWebDataDisplay.SetCaption 5528
MidItems.TWebDataDisplay.SetCaptionAttributes 5529
MidItems.TWebDataDisplay.Style 5531
MidItems.TWebDataDisplay.StyleRule 5532
MidItems.TWebDataDisplay.TabIndex 5532
MidItems.TWebDataDisplayClass 5685
MidItems.TWebDataDisplayClass type 5685
MidItems.TWebDataInput 5532
MidItems.TWebDataInput.EventContent 5535
MidItems.TWebDataInput.FieldName 5539
MidItems.TWebDataInput.FindAssociatedField 5535
MidItems.TWebDataInput.GetCaption 5536
MidItems.TWebDataInput.GetFieldName 5536
MidItems.TWebDataInput.GetParamName 5536
MidItems.TWebDataInput.GetRowSetFieldAttributes 5536
MidItems.TWebDataInput.ImplGetFieldName 5537
MidItems.TWebDataInput.ImplGetHTMLControlName 5537
MidItems.TWebDataInput.ImplGetRowSetFieldAttributes 5537
MidItems.TWebDataInput.ImplRestoreDefaults 5537
MidItems.TWebDataInput.ParamName 5540
MidItems.TWebDataInput.RestoreDefaults 5538
MidItems.TWebDataInput.RestoreFieldDefaults 5538
MidItems.TWebDataInput.SetCaption 5538
MidItems.TWebDataInput.SetFieldName 5538
MidItems.TWebDataInput.SetParamName 5539
MidItems.TWebDataInput.ValidateField 5539
MidItems.TWebDataInputClass 5685
MidItems.TWebDataInputClass type 5685
MidItems.TWebDisplay 5540
MidItems.TWebDisplay.Changed 5542
MidItems.TWebDisplay.ChangedDisplayComponent 5542
MidItems.TWebDisplay.Create 5541
MidItems.TWebDisplay.DisplayComponent 5543
MidItems.TWebDisplay.GetDisplayComponent 5542
MidItems.TWebDisplay.Notification 5542
MidItems.TWebDisplay.OnChange 5543
MidItems.TWebDisplay.Parent 5543
MidItems.TWebDisplay.SetDisplayComponent 5542
MidItems.TWebDisplayReferenceGroup 5544
MidItems.TWebDisplayReferenceGroup.Create 5545
MidItems.TWebDisplayReferenceGroup.Destroy 5546
MidItems.TWebDisplayReferenceGroup.GetWebDisplayCompo nent
5546
MidItems.TWebDisplayReferenceGroup.Notification 5546
MidItems.TWebDisplayReferenceGroup.SetWebDisplayCompon ent
5547
MidItems.TWebDisplayReferenceGroup.WebDisplay 5547
MidItems.TWebDisplayReferenceGroup.WebDisplayComponent

5547
MidItems.TWebForm 5548
MidItems.TWebForm.AddElements 5550
MidItems.TWebForm.CanAddClass 5551
MidItems.TWebForm.Content 5551

MidItems.TWebForm.Create 5549
MidItems.TWebForm.Custom 5556
MidItems.TWebForm.Destroy 5550
MidItems.TWebForm.GetDefaultWebComponents 5552
MidItems.TWebForm.GetHTMLFormName 5552
MidItems.TWebForm.GetHTMLFormVarName 5553
MidItems.TWebForm.GetLayoutAttributes 5553
MidItems.TWebForm.GetSubComponents 5553
MidItems.TWebForm.GetVisibleFields 5554
MidItems.TWebForm.ImplAddElements 5554
MidItems.TWebForm.ImplCanAddClass 5554
MidItems.TWebForm.ImplContent 5555
MidItems.TWebForm.SetDefaultWebComponents 5555
MidItems.TWebForm.Style 5556
MidItems.TWebForm.StyleRule 5557
MidItems.TWebForm.VisibleFields 5557
MidItems.TWebForm.WebComponents 5558
MidItems.TWebListInput 5558
MidItems.TWebListInput.AddElements 5562
MidItems.TWebListInput.ControlContent 5562
MidItems.TWebListInput.Create 5561
MidItems.TWebListInput.DataSet 5564
MidItems.TWebListInput.Destroy 5561
MidItems.TWebListInput.FormatInputs 5562
MidItems.TWebListInput.GetSubComponents 5563
MidItems.TWebListInput.ImplAddElements 5563
MidItems.TWebListInput.Items 5565
MidItems.TWebListInput.ItemsField 5565
MidItems.TWebListInput.Notification 5563
MidItems.TWebListInput.SetDataSet 5564
MidItems.TWebListInput.SetItems 5564
MidItems.TWebListInput.SetValues 5564
MidItems.TWebListInput.Values 5566
MidItems.TWebListInput.ValuesField 5566
MidItems.TWebRadioGroupInput 5567
MidItems.TWebRadioGroupInput.AddAttributes 5570
MidItems.TWebRadioGroupInput.Create 5570
MidItems.TWebRadioGroupInput.DisplayColumns 5572
MidItems.TWebRadioGroupInput.DisplayWidth 5572
MidItems.TWebRadioGroupInput.EventContent 5571
MidItems.TWebRadioGroupInput.FormatInputs 5571
MidItems.TWebRadioGroupInput.GetCheckIndex 5571
MidItems.TWebRadioGroupInput.ImplGetRowSetFieldAttributes 5571
MidItems.TWebRadioGroupInput.ReadOnly 5573
MidItems.TWebRadioGroupInput.RestoreFieldDefaults 5571
MidItems.TWebSelectOptionsInput 5573
MidItems.TWebSelectOptionsInput.AddAttributes 5576
MidItems.TWebSelectOptionsInput.Create 5576
MidItems.TWebSelectOptionsInput.DisplayRows 5577
MidItems.TWebSelectOptionsInput.EventContent 5577
MidItems.TWebSelectOptionsInput.FormatInputs 5577
MidItems.TWebSelectOptionsInput.GetSelectIndex 5577
MidItems.TWebStatus 5578
MidItems.TWebStatus.Caption 5582
MidItems.TWebStatus.CaptionAttributes 5582
MidItems.TWebStatus.CaptionPosition 5583
MidItems.TWebStatus.ControlContent 5580
MidItems.TWebStatus.Create 5580
MidItems.TWebStatus.DisplayWidth 5583
MidItems.TWebStatus.GetCaption 5581
MidItems.TWebStatus.Identifier 5581
MidItems.TWebStatus.ImplGetHTMLControlName 5581
MidItems.TWebStatusClass 5685
MidItems.TWebStatusClass type 5685
MidItems.TWebTextAreaInput 5583
MidItems.TWebTextAreaInput.AddAttributes 5586
MidItems.TWebTextAreaInput.ControlContent 5586
MidItems.TWebTextAreaInput.Create 5586
MidItems.TWebTextAreaInput.DisplayRows 5588
MidItems.TWebTextAreaInput.DisplayWidth 5588
MidItems.TWebTextAreaInput.EventContent 5587
MidItems.TWebTextAreaInput.ImplGetRowSetFieldAttributes 5587
MidItems.TWebTextAreaInput.ReadOnly 5588
MidItems.TWebTextAreaInput.RestoreFieldDefaults 5587
MidItems.TWebTextAreaInput.Wrap 5588
MidItems.TWebTextInput 5589
MidItems.TWebTextInput.AddAttributes 5592

MidItems.TWebTextInput.ControlContent 5592
MidItems.TWebTextInput.Create 5592
MidItems.TWebTextInput.Custom 5593
MidItems.TWebTextInput.DisplayWidth 5594
MidItems.TWebTextInput.ImplGetRowSetFieldAttributes 5593
MidItems.TWebTextInput.ReadOnly 5594
MidItems.TWebTextInput.RestoreFieldDefaults 5593
MidItems.TWebTextInput.Style 5594
MidItems.TWebTextInput.StyleRule 5595
MidItems.TXMLButton 5595
MidItems.TXMLButton.Create 5596
MidItems.TXMLButton.DefaultCaption 5599
MidItems.TXMLButton.Destroy 5597
MidItems.TXMLButton.GetCaption 5597
MidItems.TXMLButton.GetXMLBroker 5598
MidItems.TXMLButton.GetXMLUseParent 5598
MidItems.TXMLButton.Notification 5598
MidItems.TXMLButton.SetXMLBroker 5599
MidItems.TXMLButton.SetXMLUseParent 5599
MidItems.TXMLButton.XMLBroker 5599
MidItems.TXMLButton.XMLData 5600
MidItems.TXMLButton.XMLUseParent 5600
MidItems.TXMLData 5601
MidItems.TXMLData.Changed 5602
MidItems.TXMLData.ChangedXMLBroker 5603
MidItems.TXMLData.Create 5602
MidItems.TXMLData.GetXMLBroker 5603
MidItems.TXMLData.Notification 5603
MidItems.TXMLData.OnChange 5605
MidItems.TXMLData.Parent 5604
MidItems.TXMLData.SetXMLBroker 5603
MidItems.TXMLData.XMLBroker 5604
MidItems.TXMLDataParent 5605
MidItems.TXMLDataParent.ChangedUseParent 5606
MidItems.TXMLDataParent.ChangedXMLBroker 5606
MidItems.TXMLDataParent.GetXMLBroker 5607
MidItems.TXMLDataParent.ParentXMLBroker 5607
MidItems.TXMLDataParent.SetUseParent 5607
MidItems.TXMLDataParent.UseParent 5608
MidItems.TXMLDataSet 5608
MidItems.TXMLDataSet._AddRef 5610
MidItems.TXMLDataSet._Release 5611
MidItems.TXMLDataSet.ChangedDataSetField 5611
MidItems.TXMLDataSet.ChangedXMLBroker 5611
MidItems.TXMLDataSet.CompareXMLData 5611
MidItems.TXMLDataSet.ConnectionChange 5611
MidItems.TXMLDataSet.Create 5609
MidItems.TXMLDataSet.CreateDataSet 5612
MidItems.TXMLDataSet.DataSet 5614
MidItems.TXMLDataSet.DataSetField 5614
MidItems.TXMLDataSet.DataSetPath 5615
MidItems.TXMLDataSet.Destroy 5610
MidItems.TXMLDataSet.FindDataSet 5612
MidItems.TXMLDataSet.GetDataSet 5612
MidItems.TXMLDataSet.GetDataSetField 5612
MidItems.TXMLDataSet.GetDataSetPath 5613
MidItems.TXMLDataSet.Parent 5615
MidItems.TXMLDataSet.QueryInterface 5613
MidItems.TXMLDataSet.SetDataSetField 5613
MidItems.TXMLDataSet.SetXMLBroker 5613
MidItems.TXMLDataSet.WrapperDataChange 5613
MidItems.TXMLDataSetParent 5616
MidItems.TXMLDataSetParent.ChangedDataSetField 5617
MidItems.TXMLDataSetParent.ChangedUseParent 5618
MidItems.TXMLDataSetParent.ChangedXMLBroker 5618
MidItems.TXMLDataSetParent.GetDataSetField 5618
MidItems.TXMLDataSetParent.GetXMLBroker 5618
MidItems.TXMLDataSetParent.ParentDataSetField 5618
MidItems.TXMLDataSetParent.ParentXMLBroker 5619
MidItems.TXMLDataSetParent.ParentXMLDataSet 5619
MidItems.TXMLDataSetParent.SetUseParent 5619
MidItems.TXMLDataSetParent.UseParent 5619
MidItems.TXMLDisplayGroup 5620
MidItems.TXMLDisplayGroup.Create 5622
MidItems.TXMLDisplayGroup.CreateDefaultFields 5623
MidItems.TXMLDisplayGroup.Destroy 5622
MidItems.TXMLDisplayGroup.EnableValidateFields 5630
MidItems.TXMLDisplayGroup.FindField 5623

MidItems.TXMLDisplayGroup.GetDataSet 5624
MidItems.TXMLDisplayGroup.GetDataSetField 5624
MidItems.TXMLDisplayGroup.GetEnableValidateFields 5624
MidItems.TXMLDisplayGroup.GetFieldsList 5624
MidItems.TXMLDisplayGroup.GetIsMultipleRecordView 5625
MidItems.TXMLDisplayGroup.GetVisibleFields 5625
MidItems.TXMLDisplayGroup.GetXMLBroker 5625
MidItems.TXMLDisplayGroup.GetXMLDataSet 5625
MidItems.TXMLDisplayGroup.GetXMLDisplayName 5626
MidItems.TXMLDisplayGroup.GetXMLRowSetName 5626
MidItems.TXMLDisplayGroup.HasStatusField 5626
MidItems.TXMLDisplayGroup.ImplAddElements 5626
MidItems.TXMLDisplayGroup.ImplGetSubComponents 5627
MidItems.TXMLDisplayGroup.ImplIsMultipleRecordView 5627
MidItems.TXMLDisplayGroup.IsFieldInUse 5627
MidItems.TXMLDisplayGroup.Notification 5628
MidItems.TXMLDisplayGroup.SetDataSetField 5628
MidItems.TXMLDisplayGroup.SetEnableValidateFields 5628
MidItems.TXMLDisplayGroup.SetXMLBroker 5629
MidItems.TXMLDisplayGroup.ValidateDataSet 5629
MidItems.TXMLDisplayGroup.ValidateFields 5629
MidItems.TXMLDisplayGroup.VisibleFields 5630
MidItems.TXMLDisplayGroup.XMLBroker 5631
MidItems.TXMLDisplayGroup.XMLDataChange 5629
MidItems.TXMLDisplayGroup.XMLDataSet 5631
MidItems.TXMLDisplayGroup.XMLDataSetField 5631
MidItems.TXMLDisplayParent 5632
MidItems.TXMLDisplayParent.ChangedDisplayComponent 5633
MidItems.TXMLDisplayParent.ChangedUseParent 5634
MidItems.TXMLDisplayParent.Create 5633
MidItems.TXMLDisplayParent.GetDisplayComponent 5634
MidItems.TXMLDisplayParent.SetUseParent 5634
MidItems.TXMLDisplayParent.UseParent 5634
MidItems.TXMLDisplayReferenceButton 5635
MidItems.TXMLDisplayReferenceButton.Create 5637
MidItems.TXMLDisplayReferenceButton.Destroy 5637
MidItems.TXMLDisplayReferenceButton.GetDisplayComponent 5637
MidItems.TXMLDisplayReferenceButton.GetDisplayComponent Parent 5638
MidItems.TXMLDisplayReferenceButton.GetXmlDisplayName 5638
MidItems.TXMLDisplayReferenceButton.Notification 5638
MidItems.TXMLDisplayReferenceButton.SetDisplayComponent 5639
MidItems.TXMLDisplayReferenceButton.SetDisplayComponent Parent 5639
MidItems.TXMLDisplayReferenceButton.XMLComponent 5639
MidItems.TXMLDisplayReferenceButton.XMLDisplay 5640
MidItems.TXMLDisplayReferenceButton.XMLUseParent 5640
MidItems.TXMLDisplayReferenceGroup 5640
MidItems.TXMLDisplayReferenceGroup.ImplAddElements 5642
MidItems.TXMLDisplayReferenceGroup.XMLComponent 5643
MidProd 5699
MidProd namespace 5699
Classes 5701
Constants 5745
Functions 5741
Structs, Records, Enums 5742
Types 5743
MidProd.AddStdIncludes 5741
MidProd.AddStdIncludes function 5741
MidProd.PathInfoToRelativePath 5741
MidProd.PathInfoToRelativePath function 5741
MidProd.sXMLBroker 5745
MidProd.sXMLBroker constant 5745
MidProd.sXMLReadyVar 5745
MidProd.sXMLReadyVar constant 5745
MidProd.TBaseInetXPageProducer 5701
MidProd.TBaseInetXPageProducer.AddIncludes 5705
MidProd.TBaseInetXPageProducer.AddScriptComponents 5705
MidProd.TBaseInetXPageProducer.CanAddClass 5705
MidProd.TBaseInetXPageProducer.ContentFromStream 5706
MidProd.TBaseInetXPageProducer.Create 5703
MidProd.TBaseInetXPageProducer.CreatePageElements 5706
MidProd.TBaseInetXPageProducer.CreateScriptManager 5707
MidProd.TBaseInetXPageProducer.DeclareFunctions 5707
MidProd.TBaseInetXPageProducer.DeclareIncludes 5707
MidProd.TBaseInetXPageProducer.DeclareRowSets 5707

MidProd.TBaseInetXPageProducer.DeclareScript 5708
MidProd.TBaseInetXPageProducer.DeclareStyles 5708
MidProd.TBaseInetXPageProducer.DeclareWarnings 5708
MidProd.TBaseInetXPageProducer.DeclareXMLDocuments 5708
MidProd.TBaseInetXPageProducer.DeclareXMLIslands 5708
MidProd.TBaseInetXPageProducer.Destroy 5704
MidProd.TBaseInetXPageProducer.DoAfterGetContent 5709
MidProd.TBaseInetXPageProducer.DoAfterGetXMLData 5709
MidProd.TBaseInetXPageProducer.DoBeforeGetContent 5709
MidProd.TBaseInetXPageProducer.DoBeforeGetXMLData 5709
MidProd.TBaseInetXPageProducer.DoTagEvent 5710
MidProd.TBaseInetXPageProducer.EnableXMLIslands 5715
MidProd.TBaseInetXPageProducer.FindXMLBroker 5710
MidProd.TBaseInetXPageProducer.FindXMLBrokerName 5710
MidProd.TBaseInetXPageProducer.GeneratePageElements 5711
MidProd.TBaseInetXPageProducer.GetContentOptions 5711
MidProd.TBaseInetXPageProducer.GetDefaultTemplate 5711
MidProd.TBaseInetXPageProducer.GetRequiredTags 5711
MidProd.TBaseInetXPageProducer.GetScriptManager 5712
MidProd.TBaseInetXPageProducer.GetUseXMLIslands 5712
MidProd.TBaseInetXPageProducer.GetXMLData 5712
MidProd.TBaseInetXPageProducer.GetXMLDelta 5713
MidProd.TBaseInetXPageProducer.GetXMLErrors 5713
MidProd.TBaseInetXPageProducer.ImplCanAddClass 5713
MidProd.TBaseInetXPageProducer.IncludePathURL 5715
MidProd.TBaseInetXPageProducer.OnAfterGetContent 5718
MidProd.TBaseInetXPageProducer.OnAfterGetXMLData 5718
MidProd.TBaseInetXPageProducer.OnBeforeGetContent 5719
MidProd.TBaseInetXPageProducer.OnBeforeGetXMLData 5719
MidProd.TBaseInetXPageProducer.PageElements 5716
MidProd.TBaseInetXPageProducer.ScriptManager 5716
MidProd.TBaseInetXPageProducer.SetIncludePathURL 5713
MidProd.TBaseInetXPageProducer.SetStyles 5714
MidProd.TBaseInetXPageProducer.SetStylesFile 5714
MidProd.TBaseInetXPageProducer.SetWebContentOptions 5714
MidProd.TBaseInetXPageProducer.Styles 5716
MidProd.TBaseInetXPageProducer.StylesFile 5717
MidProd.TBaseMidasPageProducer 5743
MidProd.TBaseMidasPageProducer type 5743
MidProd.TCustomInetXPageProducer 5720
MidProd.TCustomInetXPageProducer.Content 5723
MidProd.TCustomInetXPageProducer.Create 5722
MidProd.TCustomInetXPageProducer.Destroy 5723
MidProd.TCustomInetXPageProducer.GetTemplateFileName 5725
MidProd.TCustomInetXPageProducer.GetTemplateStream 5725
MidProd.TCustomInetXPageProducer.HTMLDoc 5726
MidProd.TCustomInetXPageProducer.HTMLFile 5726
MidProd.TCustomInetXPageProducer.WebContent 5725
MidProd.TCustomMidasPageProducer 5743
MidProd.TCustomMidasPageProducer type 5743
MidProd.TInetXPageElements 5727
MidProd.TInetXPageElements.BodyContent 5729
MidProd.TInetXPageElements.FormsContent 5728
MidProd.TInetXPageElements.IncludesContent 5728
MidProd.TInetXPageElements.ScriptContent 5729
MidProd.TInetXPageElements.StylesContent 5729
MidProd.TInetXPageElements.WarningsContent 5729
MidProd.TInetXPageProducer 5730
MidProd.TInetXPageProducer.EnableXMLIslands 5733
MidProd.TInetXPageProducer.HTMLDoc 5733
MidProd.TInetXPageProducer.HTMLFile 5734
MidProd.TInetXPageProducer.IncludePathURL 5735
MidProd.TInetXPageProducer.OnAfterGetContent 5738
MidProd.TInetXPageProducer.OnAfterGetXMLData 5738
MidProd.TInetXPageProducer.OnBeforeGetContent 5739
MidProd.TInetXPageProducer.OnBeforeGetXMLData 5739
MidProd.TInetXPageProducer.ScriptEngine 5735
MidProd.TInetXPageProducer.Styles 5736
MidProd.TInetXPageProducer.StylesFile 5736
MidProd.TInetXPageProducer.WebPageItems 5737
MidProd.TMidasPageElements 5744
MidProd.TMidasPageElements type 5744
MidProd.TMidasPageProducer 5740
MidProd.TStdInclude 5742
MidProd.TStdInclude enumeration 5742
MidProd.TStdIncludes 5744

MidProd.TStdIncludes type 5744
MidProd.TXMLDataEvent 5744
MidProd.TXMLDataEvent type 5744
moClient enumeration member 3871
moConServer enumeration member 3871
moDevelopment enumeration member 3871
ModifyFirstName enumeration member 4022
ModifyGroupId enumeration member 4022
ModifyLastName enumeration member 4022
ModifyMiddleName enumeration member 4022
ModifyPassword enumeration member 4022
ModifyUserId enumeration member 4022
moDocumentation enumeration member 3871
moGuiTools enumeration member 3871
moServer enumeration member 3871
mpCanEject enumeration member 5821
mpCanPlay enumeration member 5821
mpCanRecord enumeration member 5821
mpCanStep enumeration member 5821
MPlayer 5747
MPlayer namespace 5747
Classes 5749
Structs, Records, Enums 5819
Types 5827
MPlayer.EMCIDeviceError 5749
MPlayer.EMPNotify 5827
MPlayer.EMPNotify type 5827
MPlayer.EMPPostNotify 5827
MPlayer.EMPPostNotify type 5827
MPlayer.TButtonSet 5828
MPlayer.TButtonSet type 5828
MPlayer.TMediaDevice 5828
MPlayer.TMediaDevice type 5828
MPlayer.TMediaPlayer 5749
MPlayer.TMediaPlayer.Anchors 5776
MPlayer.TMediaPlayer.AutoButtonSet 5756
MPlayer.TMediaPlayer.AutoEnable 5777
MPlayer.TMediaPlayer.AutoOpen 5780
MPlayer.TMediaPlayer.AutoRewind 5781
MPlayer.TMediaPlayer.Back 5757
MPlayer.TMediaPlayer.Capabilities 5781
MPlayer.TMediaPlayer.Click 5757
MPlayer.TMediaPlayer.Close 5758
MPlayer.TMediaPlayer.ColoredButtons 5781
MPlayer.TMediaPlayer.Constraints 5782
MPlayer.TMediaPlayer.Create 5755
MPlayer.TMediaPlayer.Destroy 5755
MPlayer.TMediaPlayer.DeviceID 5782
MPlayer.TMediaPlayer.DeviceType 5783
MPlayer.TMediaPlayer.Display 5783
MPlayer.TMediaPlayer.DisplayRect 5784
MPlayer.TMediaPlayer.DoNotify 5759
MPlayer.TMediaPlayer.DoubleBuffered 5784
MPlayer.TMediaPlayer.Eject 5759
MPlayer.TMediaPlayer.Enabled 5786
MPlayer.TMediaPlayer.EnabledButtons 5786
MPlayer.TMediaPlayer.EndPos 5789
MPlayer.TMediaPlayer.Error 5789
MPlayer.TMediaPlayer.ErrorMessage 5790
MPlayer.TMediaPlayer.FileName 5791
MPlayer.TMediaPlayer.Frames 5793
MPlayer.TMediaPlayer.KeyDown 5761
MPlayer.TMediaPlayer.Length 5794
MPlayer.TMediaPlayer.Loaded 5761
MPlayer.TMediaPlayer.MMNotify 5762
MPlayer.TMediaPlayer.Mode 5794
MPlayer.TMediaPlayer.Next 5762
MPlayer.TMediaPlayer.Notification 5763
MPlayer.TMediaPlayer.Notify 5794
MPlayer.TMediaPlayer.NotifyValue 5795
MPlayer.TMediaPlayer.OnClick 5810
MPlayer.TMediaPlayer.OnContextPopup 5811
MPlayer.TMediaPlayer.OnEnter 5811
MPlayer.TMediaPlayer.OnExit 5814
MPlayer.TMediaPlayer.OnMouseEnter 5817
MPlayer.TMediaPlayer.OnMouseLeave 5817
MPlayer.TMediaPlayer.OnNotify 5817
MPlayer.TMediaPlayer.OnPostClick 5818

MPlayer.TMediaPlayer.Open 5763
MPlayer.TMediaPlayer.Paint 5766
MPlayer.TMediaPlayer.ParentDoubleBuffered 5795
MPlayer.TMediaPlayer.ParentShowHint 5796
MPlayer.TMediaPlayer.Pause 5766
MPlayer.TMediaPlayer.PauseOnly 5767
MPlayer.TMediaPlayer.Play 5767
MPlayer.TMediaPlayer.PopupMenu 5796
MPlayer.TMediaPlayer.Position 5801
MPlayer.TMediaPlayer.PostClick 5770
MPlayer.TMediaPlayer.Previous 5771
MPlayer.TMediaPlayer.Resume 5771
MPlayer.TMediaPlayer.Rewind 5772
MPlayer.TMediaPlayer.Save 5772
MPlayer.TMediaPlayer.Shareable 5802
MPlayer.TMediaPlayer.ShowHint 5802
MPlayer.TMediaPlayer.Start 5804
MPlayer.TMediaPlayer.StartPos 5804
MPlayer.TMediaPlayer.StartRecording 5773
MPlayer.TMediaPlayer.Step 5774
MPlayer.TMediaPlayer.Stop 5774
MPlayer.TMediaPlayer.TabOrder 5804
MPlayer.TMediaPlayer.TabStop 5805
MPlayer.TMediaPlayer.TimeFormat 5805
MPlayer.TMediaPlayer.TrackLength 5806
MPlayer.TMediaPlayer.TrackPosition 5806
MPlayer.TMediaPlayer.Tracks 5807
MPlayer.TMediaPlayer.Updated 5775
MPlayer.TMediaPlayer.Visible 5807
MPlayer.TMediaPlayer.VisibleButtons 5809
MPlayer.TMediaPlayer.Wait 5809
MPlayer.TMPBtnType 5819
MPlayer.TMPBtnType enumeration 5819
MPlayer.TMPButton 5820
MPlayer.TMPButton record 5820
MPlayer.TMPDevCaps 5821
MPlayer.TMPDevCaps enumeration 5821
MPlayer.TMPDevCapsSet 5828
MPlayer.TMPDevCapsSet type 5828
MPlayer.TMPDeviceTypes 5821
MPlayer.TMPDeviceTypes enumeration 5821
MPlayer.TMPGlyph 5823
MPlayer.TMPGlyph enumeration 5823
MPlayer.TMPModes 5823
MPlayer.TMPModes enumeration 5823
MPlayer.TMPNotifyValues 5824
MPlayer.TMPNotifyValues enumeration 5824
MPlayer.TMPTimeFormats 5825
MPlayer.TMPTimeFormats enumeration 5825
mpNotReady enumeration member 5823
mpOpen enumeration member 5823
mpPaused enumeration member 5823
mpPlaying enumeration member 5823
mpRecording enumeration member 5823
mpSeeking enumeration member 5823
mpStopped enumeration member 5823
mpUsesWindow enumeration member 5821
mtAny enumeration member 3444
mtApplicationMenu enumeration member 7287
mtCollapsed enumeration member 7287
mtControl enumeration member 7287
mtDialogButton enumeration member 7287
mtGet enumeration member 3444
mtGroup enumeration member 7287
mtHead enumeration member 3444
mtPost enumeration member 3444
mtPut enumeration member 3444
Mtsobj 5831
Mtsobj namespace 5831
Classes 5832
Mtsobj.TMtsAutoObject 5832
Mtsobj.TMtsAutoObject.Activate 5833
Mtsobj.TMtsAutoObject.CanBePooled 5834
Mtsobj.TMtsAutoObject.Deactivate 5834
Mtsobj.TMtsAutoObject.DisableCommit 5834
Mtsobj.TMtsAutoObject.EnableCommit 5835
Mtsobj.TMtsAutoObject.IsCallerInRole 5835
Mtsobj.TMtsAutoObject.IsInTransaction 5836

Mtsobj.TMtsAutoObject.IsSecurityEnabled 5836
Mtsobj.TMtsAutoObject.ObjectContext 5838
Mtsobj.TMtsAutoObject.OnActivate 5836
Mtsobj.TMtsAutoObject.OnDeactivate 5837
Mtsobj.TMtsAutoObject.Pooled 5838
Mtsobj.TMtsAutoObject.SetAbort 5837
Mtsobj.TMtsAutoObject.SetComplete 5837
MtsRdm 5839
MtsRdm namespace 5839
Classes 5840
MtsRdm.TMtsDataModule 5840
MtsRdm.TMtsDataModule.Activate 5842
MtsRdm.TMtsDataModule.AS_ApplyUpdates 5843
MtsRdm.TMtsDataModule.AS_DataRequest 5843
MtsRdm.TMtsDataModule.AS_Execute 5844
MtsRdm.TMtsDataModule.AS_GetParams 5845
MtsRdm.TMtsDataModule.AS_GetProviderNames 5846
MtsRdm.TMtsDataModule.AS_GetRecords 5846
MtsRdm.TMtsDataModule.AS_RowRequest 5847
MtsRdm.TMtsDataModule.AutoComplete 5853
MtsRdm.TMtsDataModule.CallComplete 5848
MtsRdm.TMtsDataModule.CanBePooled 5848
MtsRdm.TMtsDataModule.Create 5841
MtsRdm.TMtsDataModule.Deactivate 5848
MtsRdm.TMtsDataModule.DisableCommit 5849
MtsRdm.TMtsDataModule.EnableCommit 5849
MtsRdm.TMtsDataModule.IsCallerInRole 5850
MtsRdm.TMtsDataModule.IsInTransaction 5850
MtsRdm.TMtsDataModule.IsSecurityEnabled 5851
MtsRdm.TMtsDataModule.ObjectContext 5853
MtsRdm.TMtsDataModule.OnActivate 5854
MtsRdm.TMtsDataModule.OnDeactivate 5854
MtsRdm.TMtsDataModule.Pooled 5853
MtsRdm.TMtsDataModule.SetAbort 5851
MtsRdm.TMtsDataModule.SetComplete 5852
mtTabs enumeration member 7287
Mtx 5857
Mtx namespace 5857
Constants 5884
Functions 5879
Interfaces 5859
Mtx.CLASS_SharedPropertyGroupManager 5884
Mtx.CLASS_SharedPropertyGroupManager constant 5884
Mtx.CLASS_TransactionContext 5884
Mtx.CLASS_TransactionContext constant 5884
Mtx.CLASS_TransactionContextEx 5885
Mtx.CLASS_TransactionContextEx constant 5885
Mtx.CreateSharedPropertyGroup 5879
Mtx.CreateSharedPropertyGroup function 5879
Mtx.CreateSharedPropertyGroupManager 5880
Mtx.CreateSharedPropertyGroupManager function 5880
Mtx.CreateTransactionContext 5880
Mtx.CreateTransactionContext function 5880
Mtx.CreateTransactionContextEx 5881
Mtx.CreateTransactionContextEx function 5881
Mtx.GetObjectContext 5882
Mtx.GetObjectContext function 5882
Mtx.IEnumNames 5859
Mtx.IEnumNames.Clone 5860
Mtx.IEnumNames.Next 5860
Mtx.IEnumNames.Reset 5860
Mtx.IEnumNames.Skip 5860
Mtx.IGetContextProperties 5861
Mtx.IGetContextProperties.Count 5861
Mtx.IGetContextProperties.EnumNames 5862
Mtx.IGetContextProperties.GetProperty 5862
Mtx.IObjectContext 5862
Mtx.IObjectContext.CreateInstance 5863
Mtx.IObjectContext.DisableCommit 5863
Mtx.IObjectContext.EnableCommit 5864
Mtx.IObjectContext.IsCallerInRole 5864
Mtx.IObjectContext.IsInTransaction 5864
Mtx.IObjectContext.IsSecurityEnabled 5864
Mtx.IObjectContext.SetAbort 5864
Mtx.IObjectContext.SetComplete 5865
Mtx.IObjectContextActivity 5865
Mtx.IObjectContextActivity.GetActivityId 5866
Mtx.IObjectControl 5866

Mtx.IObjectControl.Activate 5867
Mtx.IObjectControl.CanBePooled 5867
Mtx.IObjectControl.Deactivate 5867
Mtx.ISecurityProperty 5867
Mtx.ISecurityProperty.GetDirectCallerSID 5868
Mtx.ISecurityProperty.GetDirectCreatorSID 5868
Mtx.ISecurityProperty.GetOriginalCallerSID 5868
Mtx.ISecurityProperty.GetOriginalCreatorSID 5869
Mtx.ISecurityProperty.ReleaseSID 5869
Mtx.ISharedProperty 5869
Mtx.ISharedProperty.Get_Value 5870
Mtx.ISharedProperty.Set_Value 5870
Mtx.ISharedProperty.Value 5871
Mtx.ISharedPropertyGroup 5871
Mtx.ISharedPropertyGroup.CreateProperty 5872
Mtx.ISharedPropertyGroup.CreatePropertyByPosition 5872
Mtx.ISharedPropertyGroup.Get_PropertyByName 5872
Mtx.ISharedPropertyGroup.Get_PropertyByPosition 5872
Mtx.ISharedPropertyGroup.PropertyByName 5873
Mtx.ISharedPropertyGroup.PropertyByPosition 5873
Mtx.ISharedPropertyGroupManager 5873
Mtx.ISharedPropertyGroupManager._NewEnum 5875
Mtx.ISharedPropertyGroupManager.CreatePropertyGroup 5874
Mtx.ISharedPropertyGroupManager.Get__NewEnum 5874
Mtx.ISharedPropertyGroupManager.Get_Group 5875
Mtx.ISharedPropertyGroupManager.Group 5875
Mtx.ITransactionContext 5875
Mtx.ITransactionContext.Abort 5876
Mtx.ITransactionContext.Commit 5876
Mtx.ITransactionContext.CreateInstance 5877
Mtx.ITransactionContextEx 5877
Mtx.ITransactionContextEx.Abort 5878
Mtx.ITransactionContextEx.Commit 5878
Mtx.ITransactionContextEx.CreateInstance 5878
Mtx.LockMethod 5885
Mtx.LockMethod constant 5885
Mtx.LockSetGet 5885
Mtx.LockSetGet constant 5885
Mtx.mtsErrCtxAborted 5886
Mtx.mtsErrCtxAborted constant 5886
Mtx.mtsErrCtxAborting 5886
Mtx.mtsErrCtxAborting constant 5886
Mtx.mtsErrCtxActivityTimeout 5886
Mtx.mtsErrCtxActivityTimeout constant 5886
Mtx.mtsErrCtxNoContext 5887
Mtx.mtsErrCtxNoContext constant 5887
Mtx.mtsErrCtxNoSecurity 5887
Mtx.mtsErrCtxNoSecurity constant 5887
Mtx.mtsErrCtxNotRegistered 5887
Mtx.mtsErrCtxNotRegistered constant 5887
Mtx.mtsErrCtxOldReference 5888
Mtx.mtsErrCtxOldReference constant 5888
Mtx.mtsErrCtxRoleNotFound 5888
Mtx.mtsErrCtxRoleNotFound constant 5888
Mtx.mtsErrCtxTMNotAvailable 5888
Mtx.mtsErrCtxTMNotAvailable constant 5888
Mtx.mtsErrCtxWrongThread 5889
Mtx.mtsErrCtxWrongThread constant 5889
Mtx.Process 5889
Mtx.Process constant 5889
Mtx.SafeRef 5882
Mtx.SafeRef function 5882
Mtx.Standard 5889
Mtx.Standard constant 5889

N

NamedPipe enumeration member 4020
NoGarbageCollection enumeration member 4018
NoGlobalAction enumeration member 4024
NonTransportable enumeration member 4018
NoShadow enumeration member 4021
NoValidityCheck enumeration member 4021
nsClientReceive enumeration member 6204
nsClientSend enumeration member 6204
nsServerReceive enumeration member 6204
nsServerSend enumeration member 6204
nvAborted enumeration member 5824
nvFailure enumeration member 5824

nvSuccessful enumeration member 5824
nvSuperseded enumeration member 5824

O

oaAdd enumeration member 6380
oaAddChild enumeration member 6380
oaInsert enumeration member 6380
obClose enumeration member 6381
ObjAuto 5891
ObjAuto namespace 5891
Constants 5905
Functions 5895
Interfaces 5893
Structs, Records, Enums 5898
Types 5901
ObjAuto.CreateMethodPointer 5895
ObjAuto.CreateMethodPointer function 5895
ObjAuto.GetInvokeInstance 5896
ObjAuto.GetInvokeInstance function 5896
ObjAuto.GetMethodInfo 5896
ObjAuto.GetMethodInfo function 5896
ObjAuto.GetMethods 5896
ObjAuto.GetMethods function 5896
ObjAuto.IMethodHandler 5893
ObjAuto.IMethodHandler.Execute 5894
ObjAuto.IMethodHandler.InstanceToVariant 5894
ObjAuto.ObjectInvoke 5897
ObjAuto.ObjectInvoke function 5897
ObjAuto.paEAX 5905
ObjAuto.paEAX constant 5905
ObjAuto.paECX 5905
ObjAuto.paECX constant 5905
ObjAuto.paEDX 5906
ObjAuto.paEDX constant 5906
ObjAuto.paStack 5906
ObjAuto.paStack constant 5906
ObjAuto.PMethodInfoHeader 5901
ObjAuto.PMethodInfoHeader type 5901
ObjAuto.PParameters 5901
ObjAuto.PParameters type 5901
ObjAuto.PParamInfo 5902
ObjAuto.PParamInfo type 5902
ObjAuto.PPointer 5902
ObjAuto.PPointer type 5902
ObjAuto.PReturnInfo 5902
ObjAuto.PReturnInfo type 5902
ObjAuto.PWord 5903
ObjAuto.PWord type 5903
ObjAuto.ReleaseMethodPointer 5897
ObjAuto.ReleaseMethodPointer function 5897
ObjAuto.TCallingConvention 5898
ObjAuto.TCallingConvention enumeration 5898
ObjAuto.TDynamicInvokeEvent 5903
ObjAuto.TDynamicInvokeEvent type 5903
ObjAuto.TMethodInfoArray 5903
ObjAuto.TMethodInfoArray type 5903
ObjAuto.TMethodInfoHeader 5898
ObjAuto.TMethodInfoHeader record 5898
ObjAuto.TParameters 5899
ObjAuto.TParameters record 5899
ObjAuto.TParamFlags 5904
ObjAuto.TParamFlags type 5904
ObjAuto.TParamInfo 5899
ObjAuto.TParamInfo record 5899
ObjAuto.TReturnInfo 5900
ObjAuto.TReturnInfo record 5900
ObjBrkr 5907
ObjBrkr namespace 5907
Classes 5908
ObjBrkr.EBrokerException 5908
ObjBrkr.TServerCollection 5908
ObjBrkr.TServerCollection.Create 5909
ObjBrkr.TServerCollection.FindServer 5910
ObjBrkr.TServerCollection.GetBalancedName 5910
ObjBrkr.TServerCollection.GetNextName 5911
ObjBrkr.TServerCollection.Items 5911
ObjBrkr.TServerItem 5912
ObjBrkr.TServerItem.ComputerName 5914

ObjBrkr.TServerItem.Create 5913
ObjBrkr.TServerItem.Enabled 5915
ObjBrkr.TServerItem.GetDisplayName 5914
ObjBrkr.TServerItem.HasFailed 5915
ObjBrkr.TServerItem.Port 5916
ObjBrkr.TSimpleObjectBroker 5916
ObjBrkr.TSimpleObjectBroker.Create 5918
ObjBrkr.TSimpleObjectBroker.Destroy 5918
ObjBrkr.TSimpleObjectBroker.GetComputerForGUID 5919
ObjBrkr.TSimpleObjectBroker.GetComputerForProgID 5919
ObjBrkr.TSimpleObjectBroker.GetPortForComputer 5920
ObjBrkr.TSimpleObjectBroker.LoadBalanced 5922
ObjBrkr.TSimpleObjectBroker.LoadFromStream 5920
ObjBrkr.TSimpleObjectBroker.SaveToStream 5921
ObjBrkr.TSimpleObjectBroker.ServerData 5922
ObjBrkr.TSimpleObjectBroker.Servers 5923
ObjBrkr.TSimpleObjectBroker.SetConnectStatus 5921
ObjComAuto 5925
ObjComAuto namespace 5925
Classes 5926
Functions 5931
Structs, Records, Enums 5933
Types 5935
ObjComAuto.CreateMethodPointer 5931
ObjComAuto.CreateMethodPointer function 5931
ObjComAuto.ReleaseMethodPointer 5932
ObjComAuto.ReleaseMethodPointer function 5932
ObjComAuto.TDispatchInfo 5933
ObjComAuto.TDispatchInfo record 5933
ObjComAuto.TDispatchInfos 5935
ObjComAuto.TDispatchInfos type 5935
ObjComAuto.TDispatchKind 5933
ObjComAuto.TDispatchKind enumeration 5933
ObjComAuto.TObjectDispatch 5926
ObjComAuto.TObjectDispatch.Create 5927
ObjComAuto.TObjectDispatch.Destroy 5927
ObjComAuto.TObjectDispatch.GetIDsOfNames 5928
ObjComAuto.TObjectDispatch.GetMethodInfo 5928
ObjComAuto.TObjectDispatch.GetObjectDispatch 5928
ObjComAuto.TObjectDispatch.GetPropInfo 5929
ObjComAuto.TObjectDispatch.GetTypeInfo 5929
ObjComAuto.TObjectDispatch.GetTypeInfoCount 5929
ObjComAuto.TObjectDispatch.Instance 5930
ObjComAuto.TObjectDispatch.Invoke 5929
obLeaf enumeration member 6381
obMinus enumeration member 6381
obOpen enumeration member 6381
obPlus enumeration member 6381
ocDefault enumeration member 4484
ocGreater enumeration member 6381
ocInvalid enumeration member 6381
ocLess enumeration member 6381
ocMultiRef enumeration member 4484
ocNoMultiRef enumeration member 4484
ocoDontPrefixNode enumeration member 4480
ocoDontPutTypeAttr enumeration member 4480
ocoDontSerializeProps enumeration member 4480
ocSame enumeration member 6381
OldMetadataDesc enumeration member 4018
OleAuto 5937
OleAuto namespace 5937
Classes 5939
Constants 5965
Functions 5955
Structs, Records, Enums 5961
Types 5963
Variables 5964
OleAuto.Automation 5964
OleAuto.Automation variable 5964
OleAuto.ClassIDToProgID 5955
OleAuto.ClassIDToProgID function 5955
OleAuto.ClassIDToString 5955
OleAuto.ClassIDToString function 5955
OleAuto.CreateOleObject 5956
OleAuto.CreateOleObject function 5956
OleAuto.DispInvoke 5956
OleAuto.DispInvoke function 5956
OleAuto.DispInvokeError 5956

OleAuto.DispInvokeError function 5956
OleAuto.DllCanUnloadNow 5957
OleAuto.DllCanUnloadNow function 5957
OleAuto.DllGetClassObject 5957
OleAuto.DllGetClassObject function 5957
OleAuto.DllRegisterServer 5957
OleAuto.DllRegisterServer function 5957
OleAuto.DllUnregisterServer 5958
OleAuto.DllUnregisterServer function 5958
OleAuto.EOleError 5939
OleAuto.EOleException 5939
OleAuto.EOleException.Create 5940
OleAuto.EOleException.ErrorCode 5940
OleAuto.EOleException.HelpFile 5941
OleAuto.EOleException.Source 5941
OleAuto.EOleSysError 5941
OleAuto.EOleSysError.Create 5942
OleAuto.EOleSysError.ErrorCode 5942
OleAuto.GetActiveOleObject 5958
OleAuto.GetActiveOleObject function 5958
OleAuto.MaxDispArgs 5965
OleAuto.MaxDispArgs constant 5965
OleAuto.OleCheck 5958
OleAuto.OleCheck function 5958
OleAuto.OleError 5959
OleAuto.OleError function 5959
OleAuto.ProgIDToClassID 5959
OleAuto.ProgIDToClassID function 5959
OleAuto.StringToClassID 5959
OleAuto.StringToClassID function 5959
OleAuto.TAutoClass 5963
OleAuto.TAutoClass type 5963
OleAuto.TAutoClassInfo 5961
OleAuto.TAutoClassInfo record 5961
OleAuto.TAutoClassInstancing 5961
OleAuto.TAutoClassInstancing enumeration 5961
OleAuto.TAutoDispatch 5942
OleAuto.TAutoDispatch.AddRef 5944
OleAuto.TAutoDispatch.AutoObject 5946
OleAuto.TAutoDispatch.Create 5943
OleAuto.TAutoDispatch.GetAutoObject 5944
OleAuto.TAutoDispatch.GetIDsOfNames 5944
OleAuto.TAutoDispatch.GetTypeInfo 5944
OleAuto.TAutoDispatch.GetTypeInfoCount 5944
OleAuto.TAutoDispatch.Invoke 5945
OleAuto.TAutoDispatch.QueryInterface 5945
OleAuto.TAutoDispatch.Release 5945
OleAuto.TAutomation 5946
OleAuto.TAutomation.AutoObjectCount 5948
OleAuto.TAutomation.Create 5947
OleAuto.TAutomation.Destroy 5947
OleAuto.TAutomation.IsInprocServer 5948
OleAuto.TAutomation.OnLastRelease 5949
OleAuto.TAutomation.RegisterClass 5947
OleAuto.TAutomation.StartMode 5948
OleAuto.TAutomation.UpdateRegistry 5947
OleAuto.TAutoObject 5949
OleAuto.TAutoObject.AddRef 5951
OleAuto.TAutoObject.AutoDispatch 5952
OleAuto.TAutoObject.Create 5950
OleAuto.TAutoObject.CreateAutoDispatch 5951
OleAuto.TAutoObject.Destroy 5950
OleAuto.TAutoObject.GetExceptionInfo 5951
OleAuto.TAutoObject.OleObject 5952
OleAuto.TAutoObject.RefCount 5952
OleAuto.TAutoObject.Release 5952
OleAuto.TLastReleaseEvent 5963
OleAuto.TLastReleaseEvent type 5963
OleAuto.TRegistryClass 5953
OleAuto.TRegistryClass.Create 5953
OleAuto.TRegistryClass.Destroy 5953
OleAuto.TRegistryClass.UpdateRegistry 5954
OleAuto.TStartMode 5962
OleAuto.TStartMode enumeration 5962
OleAuto.VarFromInterface 5960
OleAuto.VarFromInterface function 5960
OleAuto.VarToAutoObject 5960
OleAuto.VarToAutoObject function 5960

OleAuto.VarToInterface 5960
OleAuto.VarToInterface function 5960
OleCtnrs 5967
OleCtnrs namespace 5967
Classes 5969
Constants 6061
Functions 6055
Interfaces 6053
Structs, Records, Enums 6056
Types 6060
OleCtnrs.DestroyMetaPict 6055
OleCtnrs.DestroyMetaPict function 6055
OleCtnrs.IVCLFrameForm 6053
OleCtnrs.IVCLFrameForm.AddContainer 6054
OleCtnrs.IVCLFrameForm.ClearBorderSpace 6054
OleCtnrs.IVCLFrameForm.Form 6054
OleCtnrs.IVCLFrameForm.RemoveContainer 6054
OleCtnrs.ovDiscardUndoState 6061
OleCtnrs.ovDiscardUndoState constant 6061
OleCtnrs.ovHide 6061
OleCtnrs.ovHide constant 6061
OleCtnrs.ovInPlaceActivate 6062
OleCtnrs.ovInPlaceActivate constant 6062
OleCtnrs.ovOpen 6062
OleCtnrs.ovOpen constant 6062
OleCtnrs.ovPrimary 6062
OleCtnrs.ovPrimary constant 6062
OleCtnrs.ovShow 6063
OleCtnrs.ovShow constant 6063
OleCtnrs.ovUIActivate 6063
OleCtnrs.ovUIActivate constant 6063
OleCtnrs.TAutoActivate 6056
OleCtnrs.TAutoActivate enumeration 6056
OleCtnrs.TCreateInfo 6056
OleCtnrs.TCreateInfo record 6056
OleCtnrs.TCreateType 6057
OleCtnrs.TCreateType enumeration 6057
OleCtnrs.TObjectMoveEvent 6060
OleCtnrs.TObjectMoveEvent type 6060
OleCtnrs.TObjectState 6057
OleCtnrs.TObjectState enumeration 6057
OleCtnrs.TOleContainer 5969
OleCtnrs.TOleContainer._AddRef 5974
OleCtnrs.TOleContainer._Release 5975
OleCtnrs.TOleContainer.Align 5991
OleCtnrs.TOleContainer.AllowActiveDoc 5995
OleCtnrs.TOleContainer.AllowInPlace 5995
OleCtnrs.TOleContainer.Anchors 5996
OleCtnrs.TOleContainer.AutoActivate 5996
OleCtnrs.TOleContainer.AutoVerbMenu 5997
OleCtnrs.TOleContainer.BorderStyle 5997
OleCtnrs.TOleContainer.CanPaste 5997
OleCtnrs.TOleContainer.Caption 5998
OleCtnrs.TOleContainer.Changed 5975
OleCtnrs.TOleContainer.ChangeIconDialog 5976
OleCtnrs.TOleContainer.Close 5976
OleCtnrs.TOleContainer.Color 6000
OleCtnrs.TOleContainer.Constraints 6001
OleCtnrs.TOleContainer.Copy 5976
OleCtnrs.TOleContainer.CopyOnSave 6002
OleCtnrs.TOleContainer.Create 5972
OleCtnrs.TOleContainer.CreateLinkToFile 5977
OleCtnrs.TOleContainer.CreateObject 5977
OleCtnrs.TOleContainer.CreateObjectFromFile 5978
OleCtnrs.TOleContainer.CreateObjectFromInfo 5979
OleCtnrs.TOleContainer.CreateParams 5979
OleCtnrs.TOleContainer.Ctl3D 6002
OleCtnrs.TOleContainer.DblClick 5980
OleCtnrs.TOleContainer.DefineProperties 5980
OleCtnrs.TOleContainer.Destroy 5973
OleCtnrs.TOleContainer.DestroyObject 5981
OleCtnrs.TOleContainer.DoEnter 5981
OleCtnrs.TOleContainer.DoVerb 5981
OleCtnrs.TOleContainer.DragCursor 6003
OleCtnrs.TOleContainer.DragMode 6003
OleCtnrs.TOleContainer.Enabled 6005
OleCtnrs.TOleContainer.GetIconMetaPict 5982
OleCtnrs.TOleContainer.GetPopupMenu 5982

OleCtnrs.TOleContainer.Iconic 6005
OleCtnrs.TOleContainer.InsertObjectDialog 5982
OleCtnrs.TOleContainer.KeyDown 5983
OleCtnrs.TOleContainer.Linked 6005
OleCtnrs.TOleContainer.LoadFromFile 5984
OleCtnrs.TOleContainer.LoadFromStream 5984
OleCtnrs.TOleContainer.Modified 6006
OleCtnrs.TOleContainer.MouseDown 5984
OleCtnrs.TOleContainer.NewInserted 6006
OleCtnrs.TOleContainer.ObjectPropertiesDialog 5985
OleCtnrs.TOleContainer.ObjectVerbs 6006
OleCtnrs.TOleContainer.OldStreamFormat 6007
OleCtnrs.TOleContainer.OleClassName 6007
OleCtnrs.TOleContainer.OleObject 6008
OleCtnrs.TOleContainer.OleObjectInterface 6008
OleCtnrs.TOleContainer.OnActivate 6021
OleCtnrs.TOleContainer.OnDeactivate 6022
OleCtnrs.TOleContainer.OnDragDrop 6022
OleCtnrs.TOleContainer.OnDragOver 6025
OleCtnrs.TOleContainer.OnEndDrag 6028
OleCtnrs.TOleContainer.OnEnter 6029
OleCtnrs.TOleContainer.OnExit 6032
OleCtnrs.TOleContainer.OnKeyDown 6035
OleCtnrs.TOleContainer.OnKeyPress 6038
OleCtnrs.TOleContainer.OnKeyUp 6039
OleCtnrs.TOleContainer.OnMouseActivate 6041
OleCtnrs.TOleContainer.OnMouseDown 6042
OleCtnrs.TOleContainer.OnMouseMove 6044
OleCtnrs.TOleContainer.OnMouseUp 6047
OleCtnrs.TOleContainer.OnObjectMove 6049
OleCtnrs.TOleContainer.OnResize 6049
OleCtnrs.TOleContainer.OnStartDrag 6049
OleCtnrs.TOleContainer.Paint 5985
OleCtnrs.TOleContainer.ParentColor 6008
OleCtnrs.TOleContainer.ParentCtl3D 6009
OleCtnrs.TOleContainer.ParentShowHint 6009
OleCtnrs.TOleContainer.Paste 5986
OleCtnrs.TOleContainer.PasteSpecialDialog 5986
OleCtnrs.TOleContainer.PopupMenu 6010
OleCtnrs.TOleContainer.PrimaryVerb 6015
OleCtnrs.TOleContainer.QueryInterface 5987
OleCtnrs.TOleContainer.Run 5987
OleCtnrs.TOleContainer.SaveAsDocument 5987
OleCtnrs.TOleContainer.SaveToFile 5989
OleCtnrs.TOleContainer.SaveToStream 5989
OleCtnrs.TOleContainer.ShowHint 6015
OleCtnrs.TOleContainer.SizeMode 6017
OleCtnrs.TOleContainer.SourceDoc 6017
OleCtnrs.TOleContainer.State 6017
OleCtnrs.TOleContainer.StorageInterface 6018
OleCtnrs.TOleContainer.TabOrder 6018
OleCtnrs.TOleContainer.TabStop 6019
OleCtnrs.TOleContainer.UpdateObject 5990
OleCtnrs.TOleContainer.UpdateVerbs 5990
OleCtnrs.TOleContainer.Visible 6019
OleCtnrs.TOleForm 6050
OleCtnrs.TOleForm.Create 6051
OleCtnrs.TOleForm.Destroy 6051
OleCtnrs.TSizeMode 6058
OleCtnrs.TSizeMode enumeration 6058
OleCtnrs.TVerbInfo 6058
OleCtnrs.TVerbInfo record 6058
OleCtrls 6065
OleCtrls namespace 6065
    Classes 6067
    Functions 6122
    Structs, Records, Enums 6124
    Types 6127
OleCtrls.EmptyParam 6122
OleCtrls.EmptyParam function 6122
OleCtrls.EOleCtrlError 6067
OleCtrls.FontToOleFont 6122
OleCtrls.FontToOleFont function 6122
OleCtrls.OleFontToFont 6123
OleCtrls.OleFontToFont function 6123
OleCtrls.PControlData 6127
OleCtrls.PControlData type 6127
OleCtrls.PControlData2 6127

OleCtrls.PControlData2 type 6127
OleCtrls.PEnumValueList 6128
OleCtrls.PEnumValueList type 6128
OleCtrls.TControlData 6124
OleCtrls.TControlData record 6124
OleCtrls.TControlData2 6125
OleCtrls.TControlData2 record 6125
OleCtrls.TEnumPropDesc 6067
OleCtrls.TEnumPropDesc.Create 6068
OleCtrls.TEnumPropDesc.Destroy 6069
OleCtrls.TEnumPropDesc.GetStrings 6069
OleCtrls.TEnumPropDesc.StringToValue 6070
OleCtrls.TEnumPropDesc.ValueToString 6070
OleCtrls.TEnumValue 6125
OleCtrls.TEnumValue record 6125
OleCtrls.TEnumValueList 6128
OleCtrls.TEnumValueList type 6128
OleCtrls.TEventDispatch 6070
OleCtrls.TEventDispatch._AddRef 6072
OleCtrls.TEventDispatch._Release 6072
OleCtrls.TEventDispatch.Control 6073
OleCtrls.TEventDispatch.Create 6071
OleCtrls.TEventDispatch.GetIDsOfNames 6072
OleCtrls.TEventDispatch.GetTypeInfo 6072
OleCtrls.TEventDispatch.GetTypeInfoCount 6073
OleCtrls.TEventDispatch.Invoke 6073
OleCtrls.TEventDispatch.QueryInterface 6073
OleCtrls.TOleControl 6074
OleCtrls.TOleControl._AddRef 6081
OleCtrls.TOleControl._Release 6081
OleCtrls.TOleControl._SetColorProp 6082
OleCtrls.TOleControl._SetTColorProp 6082
OleCtrls.TOleControl._SetTFontProp 6082
OleCtrls.TOleControl._SetTOleEnumProp 6083
OleCtrls.TOleControl._SetTPictureProp 6083
OleCtrls.TOleControl.Anchors 6119
OleCtrls.TOleControl.BrowseProperties 6083
OleCtrls.TOleControl.CanInPlaceActivate 6083
OleCtrls.TOleControl.ContextSensitiveHelp 6083
OleCtrls.TOleControl.ControlData 6119
OleCtrls.TOleControl.Create 6077
OleCtrls.TOleControl.CreateWnd 6084
OleCtrls.TOleControl.DeactivateAndUndo 6084
OleCtrls.TOleControl.DefaultDispatch 6120
OleCtrls.TOleControl.DefaultHandler 6084
OleCtrls.TOleControl.DefineProperties 6085
OleCtrls.TOleControl.Destroy 6078
OleCtrls.TOleControl.DestroyWindowHandle 6085
OleCtrls.TOleControl.DiscardUndoState 6086
OleCtrls.TOleControl.DoObjectVerb 6086
OleCtrls.TOleControl.EnableModeless 6086
OleCtrls.TOleControl.GetBorder 6087
OleCtrls.TOleControl.GetByteProp 6087
OleCtrls.TOleControl.GetColorProp 6087
OleCtrls.TOleControl.GetCompProp 6087
OleCtrls.TOleControl.GetContainer 6087
OleCtrls.TOleControl.GetCurrencyProp 6088
OleCtrls.TOleControl.GetDoubleProp 6088
OleCtrls.TOleControl.GetEnumPropDesc 6088
OleCtrls.TOleControl.GetExtendedControl 6088
OleCtrls.TOleControl.GetHelpContext 6089
OleCtrls.TOleControl.GetIDispatchProp 6089
OleCtrls.TOleControl.GetIDsOfNames 6089
OleCtrls.TOleControl.GetIntegerProp 6089
OleCtrls.TOleControl.GetIUnknownProp 6090
OleCtrls.TOleControl.GetMoniker 6090
OleCtrls.TOleControl.GetObjectVerbs 6090
OleCtrls.TOleControl.GetOleBoolProp 6090
OleCtrls.TOleControl.GetOleDateProp 6091
OleCtrls.TOleControl.GetOleEnumProp 6091
OleCtrls.TOleControl.GetOleVariantProp 6091
OleCtrls.TOleControl.GetPropDisplayString 6091
OleCtrls.TOleControl.GetPropDisplayStrings 6092
OleCtrls.TOleControl.GetProperty 6092
OleCtrls.TOleControl.GetShortIntProp 6092
OleCtrls.TOleControl.GetSingleProp 6092
OleCtrls.TOleControl.GetSmallintProp 6093
OleCtrls.TOleControl.GetStringProp 6093

OleCtrls.TOleControl.GetTColorProp 6093
OleCtrls.TOleControl.GetTDateTimeProp 6093
OleCtrls.TOleControl.GetTFontProp 6094
OleCtrls.TOleControl.GetTOleEnumProp 6094
OleCtrls.TOleControl.GetTPictureProp 6094
OleCtrls.TOleControl.GetTypeInfo 6094
OleCtrls.TOleControl.GetTypeInfoCount 6094
OleCtrls.TOleControl.GetVariantProp 6095
OleCtrls.TOleControl.GetWideStringProp 6095
OleCtrls.TOleControl.GetWindowContext 6095
OleCtrls.TOleControl.GetWordBoolProp 6095
OleCtrls.TOleControl.GetWordProp 6096
OleCtrls.TOleControl.InitControlData 6096
OleCtrls.TOleControl.InitControlInterface 6096
OleCtrls.TOleControl.InsertMenus 6096
OleCtrls.TOleControl.Invoke 6096
OleCtrls.TOleControl.InvokeEvent 6097
OleCtrls.TOleControl.InvokeMethod 6097
OleCtrls.TOleControl.IsCustomProperty 6097
OleCtrls.TOleControl.IsPropPageProperty 6097
OleCtrls.TOleControl.LockInPlaceActive 6098
OleCtrls.TOleControl.OleControlSite_TranslateAccelerator 6098
OleCtrls.TOleControl.OleInPlaceFrame_GetWindow 6098
OleCtrls.TOleControl.OleInPlaceFrame_TranslateAccelerator 6098
OleCtrls.TOleControl.OleInPlaceSite_GetWindow 6099
OleCtrls.TOleControl.OleObject 6120
OleCtrls.TOleControl.OnChanged 6099
OleCtrls.TOleControl.OnControlInfoChanged 6099
OleCtrls.TOleControl.OnFocus 6099
OleCtrls.TOleControl.OnInPlaceActivate 6100
OleCtrls.TOleControl.OnInPlaceDeactivate 6100
OleCtrls.TOleControl.OnPosRectChange 6100
OleCtrls.TOleControl.OnRequestEdit 6100
OleCtrls.TOleControl.OnShowWindow 6100
OleCtrls.TOleControl.OnUIActivate 6101
OleCtrls.TOleControl.OnUIDeactivate 6101
OleCtrls.TOleControl.PaletteChanged 6101
OleCtrls.TOleControl.PerPropBrowsing 6120
OleCtrls.TOleControl.PictureChanged 6102
OleCtrls.TOleControl.PostMessageFilter 6102
OleCtrls.TOleControl.PreMessageFilter 6102
OleCtrls.TOleControl.QueryInterface 6103
OleCtrls.TOleControl.QueryService 6103
OleCtrls.TOleControl.RemoveMenus 6103
OleCtrls.TOleControl.RequestBorderSpace 6104
OleCtrls.TOleControl.RequestNewObjectLayout 6104
OleCtrls.TOleControl.SaveObject 6104
OleCtrls.TOleControl.Scroll 6104
OleCtrls.TOleControl.ServiceQuery 6121
OleCtrls.TOleControl.SetActiveObject 6104
OleCtrls.TOleControl.SetBorderSpace 6105
OleCtrls.TOleControl.SetBounds 6105
OleCtrls.TOleControl.SetByteProp 6105
OleCtrls.TOleControl.SetColorProp 6105
OleCtrls.TOleControl.SetCompProp 6106
OleCtrls.TOleControl.SetCurrencyProp 6106
OleCtrls.TOleControl.SetDoubleProp 6106
OleCtrls.TOleControl.SetIDispatchProp 6106
OleCtrls.TOleControl.SetIntegerProp 6107
OleCtrls.TOleControl.SetIUnknownProp 6107
OleCtrls.TOleControl.SetMenu 6107
OleCtrls.TOleControl.SetName 6107
OleCtrls.TOleControl.SetOleBoolProp 6108
OleCtrls.TOleControl.SetOleDateProp 6108
OleCtrls.TOleControl.SetOleEnumProp 6108
OleCtrls.TOleControl.SetOleVariantProp 6108
OleCtrls.TOleControl.SetParent 6108
OleCtrls.TOleControl.SetPropDisplayString 6109
OleCtrls.TOleControl.SetProperty 6109
OleCtrls.TOleControl.SetShortIntProp 6110
OleCtrls.TOleControl.SetSingleProp 6110
OleCtrls.TOleControl.SetSmallintProp 6110
OleCtrls.TOleControl.SetStatusText 6110
OleCtrls.TOleControl.SetStringProp 6111
OleCtrls.TOleControl.SetTColorProp 6111
OleCtrls.TOleControl.SetTDateTimeProp 6111
OleCtrls.TOleControl.SetTFontProp 6111
OleCtrls.TOleControl.SetTOleEnumProp 6111

OleCtrls.TOleControl.SetTPictureProp 6112
OleCtrls.TOleControl.SetVariantProp 6112
OleCtrls.TOleControl.SetWideStringProp 6112
OleCtrls.TOleControl.SetWordBoolProp 6112
OleCtrls.TOleControl.SetWordProp 6113
OleCtrls.TOleControl.ShowAboutBox 6113
OleCtrls.TOleControl.ShowObject 6113
OleCtrls.TOleControl.ShowPropertyFrame 6113
OleCtrls.TOleControl.StandardEvent 6113
OleCtrls.TOleControl.SuppressException 6114
OleCtrls.TOleControl.TabStop 6121
OleCtrls.TOleControl.TransformCoords 6114
OleCtrls.TOleControl.WndProc 6114
OleCtrls.TOleEnum 6128
OleCtrls.TOleEnum type 6128
OleCtrls.TServiceQuery 6129
OleCtrls.TServiceQuery type 6129
OleServer 6131
OleServer namespace 6131
Classes 6132
Structs, Records, Enums 6148
Types 6150
OleServer.PServerData 6150
OleServer.PServerData type 6150
OleServer.TConnectKind 6148
OleServer.TConnectKind enumeration 6148
OleServer.TOleServer 6132
OleServer.TOleServer._AddRef 6135
OleServer.TOleServer._Release 6136
OleServer.TOleServer.AutoConnect 6141
OleServer.TOleServer.Connect 6136
OleServer.TOleServer.ConnectEvents 6137
OleServer.TOleServer.ConnectKind 6142
OleServer.TOleServer.Create 6133
OleServer.TOleServer.Destroy 6134
OleServer.TOleServer.Disconnect 6137
OleServer.TOleServer.DisconnectEvents 6137
OleServer.TOleServer.EventDispatch 6143
OleServer.TOleServer.GetAutoConnect 6138
OleServer.TOleServer.GetConnectKind 6138
OleServer.TOleServer.GetServer 6138
OleServer.TOleServer.InitServerData 6139
OleServer.TOleServer.InvokeEvent 6139
OleServer.TOleServer.Loaded 6140
OleServer.TOleServer.QueryInterface 6140
OleServer.TOleServer.RemoteMachineName 6143
OleServer.TOleServer.ServerData 6143
OleServer.TOleServer.SetAutoConnect 6140
OleServer.TOleServer.SetConnectKind 6141
OleServer.TServerData 6148
OleServer.TServerData record 6148
OleServer.TServerEventDispatch 6144
OleServer.TServerEventDispatch._AddRef 6145
OleServer.TServerEventDispatch._Release 6145
OleServer.TServerEventDispatch.Create 6144
OleServer.TServerEventDispatch.GetIDsOfNames 6145
OleServer.TServerEventDispatch.GetTypeInfo 6146
OleServer.TServerEventDispatch.GetTypeInfoCount 6146
OleServer.TServerEventDispatch.Invoke 6146
OleServer.TServerEventDispatch.QueryInterface 6146
OleServer.TServerEventDispatch.Server 6147
OleServer.TServerEventDispatch.ServerDisconnect 6147
OleServer.TVariantArray 6150
OleServer.TVariantArray type 6150
OneRelationAtATime enumeration member 4021
ooDrawFocusRect enumeration member 6382
ooDrawTreeRoot enumeration member 6382
ooStretchBitmaps enumeration member 6382
OPConvert 6151
OPConvert namespace 6151
Interfaces 6152
Structs, Records, Enums 6162
Types 6163
OPConvert.IOPConvert 6152
OPConvert.IOPConvert.Attachments 6160
OPConvert.IOPConvert.Encoding 6160
OPConvert.IOPConvert.GetAttachments 6154
OPConvert.IOPConvert.GetEncoding 6154

OPConvert.IOPConvert.GetOptions 6154
OPConvert.IOPConvert.GetTempDir 6154
OPConvert.IOPConvert.InvContextToMsg 6155
OPConvert.IOPConvert.MakeFault 6155
OPConvert.IOPConvert.MakeResponse 6156
OPConvert.IOPConvert.MsgToInvContext 6157
OPConvert.IOPConvert.Options 6161
OPConvert.IOPConvert.ProcessResponse 6158
OPConvert.IOPConvert.SetAttachments 6158
OPConvert.IOPConvert.SetEncoding 6159
OPConvert.IOPConvert.SetOptions 6159
OPConvert.IOPConvert.SetTempDir 6159
OPConvert.IOPConvert.TempDir 6161
OPConvert.TSOAPConvertOption 6162
OPConvert.TSOAPConvertOption enumeration 6162
OPConvert.TSOAPConvertOptions 6163
OPConvert.TSOAPConvertOptions type 6163
OPToSOAPDomConv 6165
OPToSOAPDomConv namespace 6165
Classes 6167
Constants 6210
Functions 6203
Structs, Records, Enums 6204
Types 6206
Variables 6209
OPToSOAPDomConv.ConvNodeState 6204
OPToSOAPDomConv.ConvNodeState enumeration 6204
OPToSOAPDomConv.DefArrayElemName 6209
OPToSOAPDomConv.DefArrayElemName variable 6209
OPToSOAPDomConv.ESOAPDomConvertError 6167
OPToSOAPDomConv.GetOrdPropEx 6203
OPToSOAPDomConv.GetOrdPropEx function 6203
OPToSOAPDomConv.SVarArrayType 6210
OPToSOAPDomConv.SVarArrayType constant 6210
OPToSOAPDomConv.TMemberDataNotReceivedEvent 6206
OPToSOAPDomConv.TMemberDataNotReceivedEvent type 6206
OPToSOAPDomConv.TMultiRefNodeElem 6204
OPToSOAPDomConv.TMultiRefNodeElem record 6204
OPToSOAPDomConv.TMultiRefNodeMap 6206
OPToSOAPDomConv.TMultiRefNodeMap type 6206
OPToSOAPDomConv.TMultiRefNodeMapElem 6205
OPToSOAPDomConv.TMultiRefNodeMapElem record 6205
OPToSOAPDomConv.TMultiRefNodes 6207
OPToSOAPDomConv.TMultiRefNodes type 6207
OPToSOAPDomConv.TOPToSoapDomConvert 6167
OPToSOAPDomConv.TOPToSoapDomConvert.Attachments 6176
OPToSOAPDomConv.TOPToSoapDomConvert.Create 6171
OPToSOAPDomConv.TOPToSoapDomConvert.Destroy 6171
OPToSOAPDomConv.TOPToSoapDomConvert.Encoding 6177
OPToSOAPDomConv.TOPToSoapDomConvert.InvContextToMsg 6172
OPToSOAPDomConv.TOPToSoapDomConvert.MakeFault 6172
OPToSOAPDomConv.TOPToSoapDomConvert.MakeResponse 6173
OPToSOAPDomConv.TOPToSoapDomConvert.MsgToInvContext 6174
OPToSOAPDomConv.TOPToSoapDomConvert.NewXMLDocument 6174
OPToSOAPDomConv.TOPToSoapDomConvert.ProcessResponse 6175
OPToSOAPDomConv.TOPToSoapDomConvert.TempDir 6177
OPToSOAPDomConv.TOPToSoapDomConvert.WSDLView 6178
OPToSOAPDomConv.TSOAPArrayDesc 6207
OPToSOAPDomConv.TSOAPArrayDesc type 6207
OPToSOAPDomConv.TSOAPArrayElemDesc 6205
OPToSOAPDomConv.TSOAPArrayElemDesc record 6205
OPToSOAPDomConv.TSOAPDomConv 6178
OPToSOAPDomConv.TSOAPDomConv.AddAttachment 6183
OPToSOAPDomConv.TSOAPDomConv.AddMultiRefNode 6183
OPToSOAPDomConv.TSOAPDomConv.AddNamespaceDecl 6183
OPToSOAPDomConv.TSOAPDomConv.AddObjectAsWriting 6184
OPToSOAPDomConv.TSOAPDomConv.CheckEncodingStyle 6184
OPToSOAPDomConv.TSOAPDomConv.ChildNodesAreNull

6184
OPToSOAPDomConv.TSOAPDomConv.ConvertByteArrayToSoap 6184
OPToSOAPDomConv.TSOAPDomConv.ConvertEnumToSoap 6184
OPToSOAPDomConv.TSOAPDomConv.ConvertNativeArrayToSoap 6185
OPToSOAPDomConv.TSOAPDomConv.ConvertNativeDataToSoap 6185
OPToSOAPDomConv.TSOAPDomConv.ConvertObjectToSOAP 6186
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToEnum 6186
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeArray 6186
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeArrayElem 6186
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeData 6187
OPToSOAPDomConv.TSOAPDomConv.ConvertSOAPToObject 6187
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToVariant 6187
OPToSOAPDomConv.TSOAPDomConv.ConvertVariantToSoap 6188
OPToSOAPDomConv.TSOAPDomConv.Create 6180
OPToSOAPDomConv.TSOAPDomConv.CreateMultiRefNode 6188
OPToSOAPDomConv.TSOAPDomConv.CreateNULLNode 6188
OPToSOAPDomConv.TSOAPDomConv.CreateObjectNode 6188
OPToSOAPDomConv.TSOAPDomConv.CreateScalarNodeXS 6189
OPToSOAPDomConv.TSOAPDomConv.CreateTypedNode 6189
OPToSOAPDomConv.TSOAPDomConv.Destroy 6181
OPToSOAPDomConv.TSOAPDomConv.FinalizeMultiRefNodes 6189
OPToSOAPDomConv.TSOAPDomConv.FindAttachment 6189
OPToSOAPDomConv.TSOAPDomConv.FindMultiRefNodeByID 6190
OPToSOAPDomConv.TSOAPDomConv.FindMultiRefNodeByInstance 6190
OPToSOAPDomConv.TSOAPDomConv.FindNodeByHREF 6190
OPToSOAPDomConv.TSOAPDomConv.FindPrefixForURI 6190
OPToSOAPDomConv.TSOAPDomConv.GetDataNode 6191
OPToSOAPDomConv.TSOAPDomConv.GetElementType 6191
OPToSOAPDomConv.TSOAPDomConv.GetNewID 6191
OPToSOAPDomConv.TSOAPDomConv.GetNodeAsText 6191
OPToSOAPDomConv.TSOAPDomConv.GetObjectPropAsText 6192
OPToSOAPDomConv.TSOAPDomConv.GetOptions 6192
OPToSOAPDomConv.TSOAPDomConv.GetTypeBySchemaNS 6192
OPToSOAPDomConv.TSOAPDomConv.InitObjectFromSOAP 6192
OPToSOAPDomConv.TSOAPDomConv.IsNodeAVarArray 6193
OPToSOAPDomConv.TSOAPDomConv.IsObjectWriting 6193
OPToSOAPDomConv.TSOAPDomConv.LoadObject 6193
OPToSOAPDomConv.TSOAPDomConv.MakeArrayNode 6194
OPToSOAPDomConv.TSOAPDomConv.MultiRefObject 6194
OPToSOAPDomConv.TSOAPDomConv.NodeIsNULL 6194
OPToSOAPDomConv.TSOAPDomConv.ObjectMemberNoShow 6195
OPToSOAPDomConv.TSOAPDomConv.ObjInstanceToSOAP 6195
OPToSOAPDomConv.TSOAPDomConv.OnMemberDataNotReceived 6202
OPToSOAPDomConv.TSOAPDomConv.OnUnhandledNode 6202
OPToSOAPDomConv.TSOAPDomConv.Options 6201
OPToSOAPDomConv.TSOAPDomConv.ReadHeader 6195
OPToSOAPDomConv.TSOAPDomConv.ReadRectDynArray 6196
OPToSOAPDomConv.TSOAPDomConv.ReadRectDynArrayElem 6196
OPToSOAPDomConv.TSOAPDomConv.ReadRow 6196

OPToSOAPDomConv.TSOAPDomConv.ReadVarArrayDim 6196
OPToSOAPDomConv.TSOAPDomConv.ReadVariant 6197
OPToSOAPDomConv.TSOAPDomConv.RemoveObjectAsWriting 6197
OPToSOAPDomConv.TSOAPDomConv.ResetMultiRef 6197
OPToSOAPDomConv.TSOAPDomConv.SerializationOptions 6198
OPToSOAPDomConv.TSOAPDomConv.SetNodeType 6198
OPToSOAPDomConv.TSOAPDomConv.SetObjectPropFromText 6198
OPToSOAPDomConv.TSOAPDomConv.SetOptions 6199
OPToSOAPDomConv.TSOAPDomConv.UnhandledNode 6199
OPToSOAPDomConv.TSOAPDomConv.WriteHeader 6199
OPToSOAPDomConv.TSOAPDomConv.WriteNonRectDynArray 6199
OPToSOAPDomConv.TSOAPDomConv.WriteNonRectDynArrayElem 6200
OPToSOAPDomConv.TSOAPDomConv.WriteRectDynArray 6200
OPToSOAPDomConv.TSOAPDomConv.WriteRectDynArrayElem 6200
OPToSOAPDomConv.TSOAPDomConv.WriteVarArray 6200
OPToSOAPDomConv.TSOAPDomConv.WriteVarArrayAsB64 6201
OPToSOAPDomConv.TSOAPDomConv.WriteVariant 6201
OPToSOAPDomConv.TUnhandledNodeEvent 6207
OPToSOAPDomConv.TUnhandledNodeEvent type 6207
OPToSOAPDomConv.TXMLNodeArray 6208
OPToSOAPDomConv.TXMLNodeArray type 6208
OrbPas 6211
OrbPas namespace 6211
Functions 6241
Interfaces 6213
Structs, Records, Enums 6246
Types 6248
OrbPas.AnyToVariant 6241
OrbPas.AnyToVariant function 6241
OrbPas.BindStub 6241
OrbPas.BindStub function 6241
OrbPas.CorbaAnyType 6242
OrbPas.CorbaAnyType function 6242
OrbPas.CorbaBoolean 6248
OrbPas.CorbaBoolean type 6248
OrbPas.CorbaDuplicateAny 6242
OrbPas.CorbaDuplicateAny function 6242
OrbPas.CorbaReleaseAny 6242
OrbPas.CorbaReleaseAny function 6242
OrbPas.CorbaStringFree 6243
OrbPas.CorbaStringFree function 6243
OrbPas.CorbaULong 6248
OrbPas.CorbaULong type 6248
OrbPas.CorbaWStringFree 6243
OrbPas.CorbaWStringFree function 6243
OrbPas.CreateSkeleton 6243
OrbPas.CreateSkeleton function 6243
OrbPas.CreateUserException 6244
OrbPas.CreateUserException function 6244
OrbPas.IBOA 6213
OrbPas.IBOA.Deactivate 6214
OrbPas.IBOA.GetPrincipal 6214
OrbPas.IBOA.GetPrincipalLength 6214
OrbPas.IBOA.ImplIsReady 6214
OrbPas.IBOA.ObjIsReady 6215
OrbPas.ICorbaObj 6215
OrbPas.ICorbaObj.CorbaObject 6216
OrbPas.ICorbaObj.IsLocal 6216
OrbPas.IMarshalInBuffer 6216
OrbPas.IMarshalInBuffer.Buffer 6217
OrbPas.IMarshalInBuffer.GetAny 6218
OrbPas.IMarshalInBuffer.GetChar 6218
OrbPas.IMarshalInBuffer.GetDouble 6218
OrbPas.IMarshalInBuffer.GetFloat 6218
OrbPas.IMarshalInBuffer.GetLong 6218
OrbPas.IMarshalInBuffer.GetObject 6219
OrbPas.IMarshalInBuffer.GetShort 6219
OrbPas.IMarshalInBuffer.GetText 6219
OrbPas.IMarshalInBuffer.GetUnsignedChar 6219

OrbPas.IMarshalInBuffer.GetUnsignedLong 6220
OrbPas.IMarshalInBuffer.GetUnsignedShort 6220
OrbPas.IMarshalInBuffer.GetWidechar 6220
OrbPas.IMarshalInBuffer.GetWideText 6220
OrbPas.IMarshalOutBuffer 6220
OrbPas.IMarshalOutBuffer.Buffer 6222
OrbPas.IMarshalOutBuffer.PutAny 6222
OrbPas.IMarshalOutBuffer.PutChar 6222
OrbPas.IMarshalOutBuffer.PutDouble 6222
OrbPas.IMarshalOutBuffer.PutFloat 6222
OrbPas.IMarshalOutBuffer.PutLong 6223
OrbPas.IMarshalOutBuffer.PutObject 6223
OrbPas.IMarshalOutBuffer.PutShort 6223
OrbPas.IMarshalOutBuffer.PutText 6223
OrbPas.IMarshalOutBuffer.PutUnsignedChar 6224
OrbPas.IMarshalOutBuffer.PutUnsignedLong 6224
OrbPas.IMarshalOutBuffer.PutUnsignedShort 6224
OrbPas.IMarshalOutBuffer.PutWidechar 6224
OrbPas.IMarshalOutBuffer.PutWideText 6224
OrbPas.InitORB 6244
OrbPas.InitORB function 6244
OrbPas.IORB 6225
OrbPas.IORB.BOAInit 6226
OrbPas.IORB.CreateAliasTC 6226
OrbPas.IORB.CreateArrayTC 6226
OrbPas.IORB.CreateObjRefTC 6227
OrbPas.IORB.CreateSequenceTC 6227
OrbPas.IORB.CreateStructTC 6227
OrbPas.IORB.CreateTC 6227
OrbPas.IORB.DispatchStruct 6227
OrbPas.IORB.FindRepositoryTC 6228
OrbPas.IORB.MakeAny 6228
OrbPas.IORB.ObjectToString 6228
OrbPas.IORB.Orb 6228
OrbPas.IORB.Shutdown 6229
OrbPas.IORB.StringToObject 6229
OrbPas.ISkeleton 6229
OrbPas.ISkeleton.GetImplementation 6230
OrbPas.ISkeleton.GetReplyBuffer 6230
OrbPas.ISkeletonObject 6230
OrbPas.ISkeletonObject.Execute 6231
OrbPas.ISkeletonObject.GetImplementation 6231
OrbPas.ISkeletonObject.GetSkeleton 6231
OrbPas.IStub 6232
OrbPas.IStub.CreateRequest 6233
OrbPas.IStub.Dispatch 6233
OrbPas.IStub.GetInterface 6233
OrbPas.IStub.Hash 6233
OrbPas.IStub.Invoke 6234
OrbPas.IStub.IsA 6234
OrbPas.IStub.NonExistent 6234
OrbPas.IStub.RepositoryID 6234
OrbPas.IStub.SetPrincipal 6235
OrbPas.IStubObject 6235
OrbPas.IStubObject.GetStub 6236
OrbPas.ITypeCode 6236
OrbPas.ITypeCode.ContentType 6237
OrbPas.ITypeCode.DefaultIndex 6237
OrbPas.ITypeCode.DiscriminatorType 6237
OrbPas.ITypeCode.Equal 6238
OrbPas.ITypeCode.Id 6238
OrbPas.ITypeCode.Kind 6238
OrbPas.ITypeCode.Length 6238
OrbPas.ITypeCode.MemberCount 6239
OrbPas.ITypeCode.MemberLabel 6239
OrbPas.ITypeCode.MemberName 6239
OrbPas.ITypeCode.MemberType 6239
OrbPas.ITypeCode.Name 6239
OrbPas.ITypeCode.Parameter 6240
OrbPas.ITypeCode.ParameterCount 6240
OrbPas.PCorbaAny 6249
OrbPas.PCorbaAny type 6249
OrbPas.PCorbaInterfaceDef 6249
OrbPas.PCorbaInterfaceDef type 6249
OrbPas.PCorbaObject 6249
OrbPas.PCorbaObject type 6249
OrbPas.PCorbaOrb 6250
OrbPas.PCorbaOrb type 6250

OrbPas.PExceptionDescription 6250
OrbPas.PExceptionDescription type 6250
OrbPas.PMarshalInbuffer 6250
OrbPas.PMarshalInbuffer type 6250
OrbPas.PMarshalOutbuffer 6251
OrbPas.PMarshalOutbuffer type 6251
OrbPas.PUserExceptionProxy 6251
OrbPas.PUserExceptionProxy type 6251
OrbPas.RegisterUserException 6244
OrbPas.RegisterUserException function 6244
OrbPas.SequenceToVariant 6245
OrbPas.SequenceToVariant function 6245
OrbPas.TArgv 6251
OrbPas.TArgv type 6251
OrbPas.TCKind 6246
OrbPas.TCKind enumeration 6246
OrbPas.TCopyUserExceptionProc 6252
OrbPas.TCopyUserExceptionProc type 6252
OrbPas.TStructMember 6247
OrbPas.TStructMember record 6247
OrbPas.TStructMembers 6252
OrbPas.TStructMembers type 6252
OrbPas.TThrowUserExceptionProc 6252
OrbPas.TThrowUserExceptionProc type 6252
OrbPas.TUserExceptionFactoryProc 6253
OrbPas.TUserExceptionFactoryProc type 6253
OrbPas.UnRegisterUserException 6245
OrbPas.UnRegisterUserException function 6245
OrbPas.VariantToAny 6245
OrbPas.VariantToAny function 6245
osEmpty enumeration member 6057
osInPlaceActive enumeration member 6057
osLoaded enumeration member 6057
osOpen enumeration member 6057
osPictureText enumeration member 6382
osPlusMinusPictureText enumeration member 6382
osPlusMinusText enumeration member 6382
osRunning enumeration member 6057
osText enumeration member 6382
osTreePictureText enumeration member 6382
osTreeText enumeration member 6382
osUIActive enumeration member 6057
otOwnerDraw enumeration member 6383
otStandard enumeration member 6383
Outline 6255
Outline namespace 6255
    Classes 6257
    Structs, Records, Enums 6380
    Types 6385
Outline.EOutlineChange 6385
Outline.EOutlineChange type 6385
Outline.EOutlineError 6257
Outline.OutlineError 6258
Outline.TAttachMode 6380
Outline.TAttachMode enumeration 6380
Outline.TBitmapArray 6385
Outline.TBitmapArray type 6385
Outline.TBitmapArrayRange 6386
Outline.TBitmapArrayRange type 6386
Outline.TChangeRange 6386
Outline.TChangeRange type 6386
Outline.TCustomOutline 6258
Outline.TCustomOutline.Add 6264
Outline.TCustomOutline.AddChild 6265
Outline.TCustomOutline.AddChildObject 6266
Outline.TCustomOutline.AddObject 6266
Outline.TCustomOutline.BadIndex 6267
Outline.TCustomOutline.BeginUpdate 6268
Outline.TCustomOutline.Canvas 6284
Outline.TCustomOutline.Clear 6268
Outline.TCustomOutline.Click 6268
Outline.TCustomOutline.Collapse 6269
Outline.TCustomOutline.Create 6262
Outline.TCustomOutline.DblClick 6269
Outline.TCustomOutline.DefineProperties 6269
Outline.TCustomOutline.Delete 6270
Outline.TCustomOutline.DeleteNode 6270
Outline.TCustomOutline.Destroy 6263

Outline.TCustomOutline.DrawCell 6271
Outline.TCustomOutline.EndUpdate 6271
Outline.TCustomOutline.Expand 6272
Outline.TCustomOutline.FullCollapse 6272
Outline.TCustomOutline.FullExpand 6272
Outline.TCustomOutline.GetDataItem 6273
Outline.TCustomOutline.GetItem 6273
Outline.TCustomOutline.GetNodeDisplayWidth 6274
Outline.TCustomOutline.GetTextItem 6274
Outline.TCustomOutline.GetVisibleNode 6274
Outline.TCustomOutline.Insert 6275
Outline.TCustomOutline.InsertObject 6275
Outline.TCustomOutline.InvalidateCell 6276
Outline.TCustomOutline.ItemCount 6286
Outline.TCustomOutline.ItemHeight 6286
Outline.TCustomOutline.Items 6287
Outline.TCustomOutline.ItemSeparator 6288
Outline.TCustomOutline.KeyDown 6277
Outline.TCustomOutline.KeyPress 6277
Outline.TCustomOutline.Lines 6288
Outline.TCustomOutline.Loaded 6278
Outline.TCustomOutline.LoadFromFile 6278
Outline.TCustomOutline.LoadFromStream 6279
Outline.TCustomOutline.MouseDown 6280
Outline.TCustomOutline.Move 6280
Outline.TCustomOutline.OnCollapse 6295
Outline.TCustomOutline.OnDrawItem 6295
Outline.TCustomOutline.OnExpand 6296
Outline.TCustomOutline.Options 6288
Outline.TCustomOutline.OutlineStyle 6289
Outline.TCustomOutline.PictureClosed 6290
Outline.TCustomOutline.PictureLeaf 6291
Outline.TCustomOutline.PictureMinus 6291
Outline.TCustomOutline.PictureOpen 6291
Outline.TCustomOutline.PicturePlus 6292
Outline.TCustomOutline.Row 6292
Outline.TCustomOutline.SaveToFile 6281
Outline.TCustomOutline.SaveToStream 6281
Outline.TCustomOutline.SelectedItem 6294
Outline.TCustomOutline.SetDisplayWidth 6282
Outline.TCustomOutline.SetGoodIndex 6282
Outline.TCustomOutline.SetLevel 6283
Outline.TCustomOutline.SetUpdateState 6283
Outline.TCustomOutline.Style 6294
Outline.TOutline 6296
Outline.TOutline.Align 6302
Outline.TOutline.BorderStyle 6305
Outline.TOutline.Color 6306
Outline.TOutline.Ctl3D 6306
Outline.TOutline.DoubleBuffered 6307
Outline.TOutline.DragCursor 6309
Outline.TOutline.DragKind 6309
Outline.TOutline.DragMode 6309
Outline.TOutline.Enabled 6311
Outline.TOutline.Font 6311
Outline.TOutline.ItemHeight 6313
Outline.TOutline.ItemSeparator 6313
Outline.TOutline.Lines 6313
Outline.TOutline.OnClick 6330
Outline.TOutline.OnCollapse 6332
Outline.TOutline.OnContextPopup 6332
Outline.TOutline.OnDblClick 6333
Outline.TOutline.OnDragDrop 6333
Outline.TOutline.OnDragOver 6336
Outline.TOutline.OnDrawItem 6340
Outline.TOutline.OnEndDock 6340
Outline.TOutline.OnEndDrag 6341
Outline.TOutline.OnEnter 6342
Outline.TOutline.OnExit 6344
Outline.TOutline.OnExpand 6347
Outline.TOutline.OnKeyDown 6347
Outline.TOutline.OnKeyPress 6351
Outline.TOutline.OnKeyUp 6352
Outline.TOutline.OnMouseActivate 6354
Outline.TOutline.OnMouseDown 6355
Outline.TOutline.OnMouseEnter 6357
Outline.TOutline.OnMouseLeave 6357
Outline.TOutline.OnMouseMove 6357

Outline.TOutline.OnMouseUp 6360
Outline.TOutline.OnStartDock 6362
Outline.TOutline.OnStartDrag 6362
Outline.TOutline.Options 6314
Outline.TOutline.OutlineStyle 6315
Outline.TOutline.ParentColor 6315
Outline.TOutline.ParentCtl3D 6316
Outline.TOutline.ParentDoubleBuffered 6316
Outline.TOutline.ParentFont 6316
Outline.TOutline.ParentShowHint 6317
Outline.TOutline.PictureClosed 6318
Outline.TOutline.PictureLeaf 6318
Outline.TOutline.PictureMinus 6318
Outline.TOutline.PictureOpen 6319
Outline.TOutline.PicturePlus 6319
Outline.TOutline.PopupMenu 6320
Outline.TOutline.ScrollBars 6325
Outline.TOutline.ShowHint 6325
Outline.TOutline.Style 6327
Outline.TOutline.TabOrder 6327
Outline.TOutline.TabStop 6328
Outline.TOutline.Visible 6328
Outline.TOutlineBitmap 6381
Outline.TOutlineBitmap enumeration 6381
Outline.TOutlineBitmaps 6386
Outline.TOutlineBitmaps type 6386
Outline.TOutlineNode 6363
Outline.TOutlineNode.AddNode 6366
Outline.TOutlineNode.ChangeLevelBy 6367
Outline.TOutlineNode.Collapse 6367
Outline.TOutlineNode.Create 6365
Outline.TOutlineNode.Data 6374
Outline.TOutlineNode.Destroy 6365
Outline.TOutlineNode.Expand 6368
Outline.TOutlineNode.ExpandCount 6374
Outline.TOutlineNode.Expanded 6375
Outline.TOutlineNode.FullExpand 6368
Outline.TOutlineNode.FullPath 6375
Outline.TOutlineNode.GetDataItem 6368
Outline.TOutlineNode.GetDisplayWidth 6369
Outline.TOutlineNode.getFirstChild 6369
Outline.TOutlineNode.GetLastChild 6369
Outline.TOutlineNode.GetNextChild 6370
Outline.TOutlineNode.GetNodeAtIndex 6370
Outline.TOutlineNode.GetPrevChild 6370
Outline.TOutlineNode.GetRowOfNode 6371
Outline.TOutlineNode.GetTextItem 6371
Outline.TOutlineNode.GetVisibleNode 6371
Outline.TOutlineNode.HasAsParent 6372
Outline.TOutlineNode.HasItems 6376
Outline.TOutlineNode.Index 6376
Outline.TOutlineNode.InsertNode 6372
Outline.TOutlineNode.InternalRemove 6372
Outline.TOutlineNode.IsVisible 6376
Outline.TOutlineNode.Items 6377
Outline.TOutlineNode.Level 6377
Outline.TOutlineNode.List 6378
Outline.TOutlineNode.MoveTo 6372
Outline.TOutlineNode.Outline 6378
Outline.TOutlineNode.Parent 6378
Outline.TOutlineNode.Remove 6373
Outline.TOutlineNode.Text 6378
Outline.TOutlineNode.TopItem 6379
Outline.TOutlineNode.WriteNode 6373
Outline.TOutlineNodeCompare 6381
Outline.TOutlineNodeCompare enumeration 6381
Outline.TOutlineOption 6382
Outline.TOutlineOption enumeration 6382
Outline.TOutlineOptions 6387
Outline.TOutlineOptions type 6387
Outline.TOutlineStyle 6382
Outline.TOutlineStyle enumeration 6382
Outline.TOutlineType 6383
Outline.TOutlineType enumeration 6383
OutlineError class 6258
about OutlineError class 6258

P

paCenter enumeration member 4968
PagItems 6389
PagItems namespace 6389
Classes 6390
PagItems.TBasePageItemsProducer 6390
PagItems.TBasePageItemsProducer.Create 6391
PagItems.TBasePageItemsProducer.Destroy 6391
PagItems.TBasePageItemsProducer.GetChildren 6392
PagItems.TBasePageItemsProducer.GetComponentList 6392
PagItems.TBasePageItemsProducer.GetDefaultComponentList 6392
PagItems.TBasePageItemsProducer.SetChildOrder 6393
PagItems.TBasePageItemsProducer.WebPageItems 6393
PagItems.TPageItemsProducer 6394
PagItems.TPageItemsProducer.Create 6395
PagItems.TPageItemsProducer.Destroy 6395
PagItems.TPageItemsProducer.GetTemplateStream 6396
PagItems.TPageItemsProducer.HTMLDoc 6396
PagItems.TPageItemsProducer.HTMLFile 6397
paLeft enumeration member 4968
paRight enumeration member 4968
pcCollation enumeration member 6438
pcCopies enumeration member 6438
pcOrientation enumeration member 6438
pdDown enumeration member 4248
pdLeft enumeration member 4248
pdRight enumeration member 4248
pdUp enumeration member 4248
pf15bit enumeration member 2871
pf16bit enumeration member 2871
pf1bit enumeration member 2871
pf24bit enumeration member 2871
pf32bit enumeration member 2871
pf4bit enumeration member 2871
pf8bit enumeration member 2871
pfCustom enumeration member 2871
pfDevice enumeration member 2871
PlatformDefaultStyleActnCtrls 6399
PlatformDefaultStyleActnCtrls namespace 6399
Classes 6400
Variables 6403
PlatformDefaultStyleActnCtrls.PlatformDefaultStyle 6403
PlatformDefaultStyleActnCtrls.PlatformDefaultStyle variable 6403
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars 6400
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetAddRemoveItemClass 6401
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetColorMapClass 6401
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetControlClass 6401
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetPopupClass 6402
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetScrollBtnClass 6402
PlatformDefaultStyleActnCtrls.TPlatformDefaultStyleActionBars.GetStyleName 6402
pmAuto enumeration member 2588
pmBlack enumeration member 2868
pmCopy enumeration member 2868
pmDouble enumeration member 4770
pmExplicit enumeration member 2588
pmExtended enumeration member 4770
pmMask enumeration member 2868
pmMaskNotPen enumeration member 2868
pmMaskPenNot enumeration member 2868
pmMerge enumeration member 2868
pmMergeNotPen enumeration member 2868
pmMergePenNot enumeration member 2868
pmNone enumeration member 2588
pmNop enumeration member 2868
pmNot enumeration member 2868
pmNotCopy enumeration member 2868
pmNotMask enumeration member 2868

pmNotMerge enumeration member 2868
pmNotXor enumeration member 2868
pmReserved enumeration member 4770
pmSingle enumeration member 4770
pmWhite enumeration member 2868
pmXor enumeration member 2868
poAllowCommandText enumeration member 6585
poAllowMultiRecordUpdates enumeration member 6585
poAutoRefresh enumeration member 6585
poCascadeDeletes enumeration member 6585
poCascadeUpdates enumeration member 6585
poDefault enumeration member 2590
poDefaultPosOnly enumeration member 2590
poDefaultSizeOnly enumeration member 2590
poDesigned enumeration member 2590
poDesktopCenter enumeration member 2590
poDisableDeletes enumeration member 6585
poDisableEdits enumeration member 6585
poDisableInserts enumeration member 6585
poFetchBlobsOnDemand enumeration member 6585
poFetchDetailsOnDemand enumeration member 6585
poHorizontal enumeration member 4248
poIncFieldProps enumeration member 6585
poLandscape enumeration member 6438
poMainFormCenter enumeration member 2590
poNone enumeration member 2591
poNoReset enumeration member 6585
poOwnerFormCenter enumeration member 2590
poPortrait enumeration member 6438
poPrintToFit enumeration member 2591
poPropogateChanges enumeration member 6585
poProportional enumeration member 2591
poReadOnly enumeration member 6585
poRetainServerOrder enumeration member 6585
poScreenCenter enumeration member 2590
poUseQuoteChar enumeration member 6585
poVertical enumeration member 4248
Printers 6405
Printers namespace 6405
Classes 6406
Functions 6434
Structs, Records, Enums 6438
Types 6440
Printers.AssignPrn 6434
Printers.AssignPrn function 6434
Printers.EPrinter 6406
Printers.Printer 6434
Printers.Printer function 6434
Printers.SetPrinter 6436
Printers.SetPrinter function 6436
Printers.TPrinter 6406
Printers.TPrinter.Abort 6409
Printers.TPrinter.Aborted 6419
Printers.TPrinter.BeginDoc 6411
Printers.TPrinter.Canvas 6421
Printers.TPrinter.Capabilities 6421
Printers.TPrinter.Copies 6422
Printers.TPrinter.Create 6408
Printers.TPrinter.Destroy 6408
Printers.TPrinter.EndDoc 6413
Printers.TPrinter.Fonts 6422
Printers.TPrinter.GetPrinter 6415
Printers.TPrinter.Handle 6422
Printers.TPrinter.NewPage 6415
Printers.TPrinter.Orientation 6424
Printers.TPrinter.PageHeight 6425
Printers.TPrinter.PageNumber 6427
Printers.TPrinter.PageWidth 6428
Printers.TPrinter.PrinterIndex 6430
Printers.TPrinter.Printers 6430
Printers.TPrinter.Printing 6430
Printers.TPrinter.Refresh 6417
Printers.TPrinter.SetPrinter 6418
Printers.TPrinter.Title 6433
Printers.TPrinterCapabilities 6440
Printers.TPrinterCapabilities type 6440
Printers.TPrinterCapability 6438
Printers.TPrinterCapability enumeration 6438

Printers.TPrinterOrientation 6438
Printers.TPrinterOrientation enumeration 6438
Printers.TPrinterState 6439
Printers.TPrinterState enumeration 6439
Provider 6441
Provider namespace 6441
Classes 6443
Constants 6596
Functions 6582
Interfaces 6580
Structs, Records, Enums 6584
Types 6589
Variables 6595
Provider.EDSWriter 6443
Provider.EDSWriter.Create 6444
Provider.EDSWriter.ErrorCode 6444
Provider.GetObjectProperty 6582
Provider.GetObjectProperty function 6582
Provider.GetStringProperty 6582
Provider.GetStringProperty function 6582
Provider.InformixLob 6595
Provider.InformixLob variable 6595
Provider.IProviderContainer 6580
Provider.IProviderContainer.RegisterProvider 6581
Provider.IProviderContainer.UnRegisterProvider 6581
Provider.MetaDataOption 6596
Provider.MetaDataOption constant 6596
Provider.PPutFieldInfo 6589
Provider.PPutFieldInfo type 6589
Provider.ResetOption 6597
Provider.ResetOption constant 6597
Provider.TAfterUpdateRecordEvent 6589
Provider.TAfterUpdateRecordEvent type 6589
Provider.TBaseProvider 6445
Provider.TBaseProvider.AfterUpdateRecord 6458
Provider.TBaseProvider.BeforeUpdateRecord 6458
Provider.TBaseProvider.CheckResolver 6449
Provider.TBaseProvider.Create 6447
Provider.TBaseProvider.CreateDataPacket 6449
Provider.TBaseProvider.CreateResolver 6450
Provider.TBaseProvider.Destroy 6448
Provider.TBaseProvider.DoAfterUpdateRecord 6450
Provider.TBaseProvider.DoBeforeCommit 6450
Provider.TBaseProvider.DoBeforeUpdateRecord 6451
Provider.TBaseProvider.DoOnGetData 6451
Provider.TBaseProvider.DoOnUpdateData 6451
Provider.TBaseProvider.FetchDetails 6451
Provider.TBaseProvider.FreeResolver 6452
Provider.TBaseProvider.InternalApplyUpdates 6452
Provider.TBaseProvider.InternalGetRecords 6453
Provider.TBaseProvider.InternalRowRequest 6454
Provider.TBaseProvider.LocateRecord 6454
Provider.TBaseProvider.OnDataRequest 6458
Provider.TBaseProvider.OnGetData 6459
Provider.TBaseProvider.OnUpdateData 6459
Provider.TBaseProvider.OnUpdateError 6460
Provider.TBaseProvider.Options 6455
Provider.TBaseProvider.PacketDataSet 6456
Provider.TBaseProvider.Resolver 6456
Provider.TBaseProvider.UpdateMode 6457
Provider.TBaseProvider.UpdateRecord 6455
Provider.TBeforeCommitEvent 6590
Provider.TBeforeCommitEvent type 6590
Provider.TBeforeUpdateRecordEvent 6590
Provider.TBeforeUpdateRecordEvent type 6590
Provider.TCustomPacketWriter 6460
Provider.TCustomPacketWriter.AddAttribute 6463
Provider.TCustomPacketWriter.Check 6463
Provider.TCustomPacketWriter.Create 6461
Provider.TCustomPacketWriter.Destroy 6461
Provider.TCustomPacketWriter.DSWriter 6464
Provider.TCustomPacketWriter.FBuffer 6462
Provider.TCustomPacketWriter.SetDSWriter 6463
Provider.TCustomProvider 6464
Provider.TCustomProvider.ActiveUpdateException 6482
Provider.TCustomProvider.AfterApplyUpdates 6483
Provider.TCustomProvider.AfterExecute 6484
Provider.TCustomProvider.AfterGetParams 6485

Provider.TCustomProvider.AfterGetRecords 6485
Provider.TCustomProvider.AfterRowRequest 6486
Provider.TCustomProvider.ApplyUpdates 6468
Provider.TCustomProvider.BeforeApplyUpdates 6487
Provider.TCustomProvider.BeforeExecute 6487
Provider.TCustomProvider.BeforeGetParams 6488
Provider.TCustomProvider.BeforeGetRecords 6488
Provider.TCustomProvider.BeforeRowRequest 6490
Provider.TCustomProvider.Create 6466
Provider.TCustomProvider.Data 6482
Provider.TCustomProvider.DataRequest 6468
Provider.TCustomProvider.Destroy 6467
Provider.TCustomProvider.DoAfterApplyUpdates 6469
Provider.TCustomProvider.DoAfterExecute 6469
Provider.TCustomProvider.DoAfterGetParams 6470
Provider.TCustomProvider.DoAfterGetRecords 6470
Provider.TCustomProvider.DoAfterRowRequest 6471
Provider.TCustomProvider.DoApplyUpdates 6471
Provider.TCustomProvider.DoBeforeApplyUpdates 6471
Provider.TCustomProvider.DoBeforeExecute 6472
Provider.TCustomProvider.DoBeforeGetParams 6472
Provider.TCustomProvider.DoBeforeGetRecords 6473
Provider.TCustomProvider.DoBeforeRowRequest 6474
Provider.TCustomProvider.DoGetRecords 6474
Provider.TCustomProvider.Execute 6474
Provider.TCustomProvider.Exported 6482
Provider.TCustomProvider.GetParams 6475
Provider.TCustomProvider.GetRecords 6476
Provider.TCustomProvider.InternalApplyUpdates 6477
Provider.TCustomProvider.InternalExecute 6478
Provider.TCustomProvider.InternalGetParams 6478
Provider.TCustomProvider.InternalGetRecords 6479
Provider.TCustomProvider.InternalRowRequest 6480
Provider.TCustomProvider.OnDataRequest 6490
Provider.TCustomProvider.OnValidate 6491
Provider.TCustomProvider.OwnerData 6483
Provider.TCustomProvider.RowRequest 6480
Provider.TCustomProvider.SetActiveUpdateException 6481
Provider.TCustomResolver 6491
Provider.TCustomResolver.ApplyUpdates 6494
Provider.TCustomResolver.BeginUpdate 6494
Provider.TCustomResolver.Create 6493
Provider.TCustomResolver.Destroy 6493
Provider.TCustomResolver.DoDelete 6495
Provider.TCustomResolver.DoInsert 6495
Provider.TCustomResolver.DoUpdate 6496
Provider.TCustomResolver.EndUpdate 6496
Provider.TCustomResolver.FreeTreeData 6497
Provider.TCustomResolver.HandleUpdateError 6497
Provider.TCustomResolver.InitializeConflictBuffer 6497
Provider.TCustomResolver.InitKeyFields 6498
Provider.TCustomResolver.InitTreeData 6498
Provider.TCustomResolver.InternalBeforeResolve 6498
Provider.TCustomResolver.InternalUpdateRecord 6499
Provider.TCustomResolver.LogUpdateError 6499
Provider.TCustomResolver.LogUpdateRecord 6500
Provider.TCustomResolver.Provider 6501
Provider.TCustomResolver.RowRequest 6500
Provider.TCustomResolver.UpdateTree 6501
Provider.TDataPacketWriter 6501
Provider.TDataPacketWriter.AddColumn 6504
Provider.TDataPacketWriter.AddConstraints 6504
Provider.TDataPacketWriter.AddDataSetAttributes 6504
Provider.TDataPacketWriter.AddExtraFieldProps 6504
Provider.TDataPacketWriter.AddFieldLinks 6505
Provider.TDataPacketWriter.AddIndexDefs 6505
Provider.TDataPacketWriter.Constraints 6509
Provider.TDataPacketWriter.Destroy 6503
Provider.TDataPacketWriter.FreeInfoRecords 6505
Provider.TDataPacketWriter.GetDataPacket 6505
Provider.TDataPacketWriter.GetFieldIdx 6506
Provider.TDataPacketWriter.InitPutProcs 6506
Provider.TDataPacketWriter.OnGetParams 6510
Provider.TDataPacketWriter.Options 6509
Provider.TDataPacketWriter.PacketOptions 6510
Provider.TDataPacketWriter.PutADTField 6506
Provider.TDataPacketWriter.PutArrayField 6506
Provider.TDataPacketWriter.PutBlobField 6506

Provider.TDataPacketWriter.PutCalcField 6507
Provider.TDataPacketWriter.PutDataSetField 6507
Provider.TDataPacketWriter.PutField 6507
Provider.TDataPacketWriter.PutStringField 6507
Provider.TDataPacketWriter.PutVarBytesField 6508
Provider.TDataPacketWriter.PutWideStringField 6508
Provider.TDataPacketWriter.RefreshPutProcs 6508
Provider.TDataPacketWriter.Reset 6508
Provider.TDataPacketWriter.WriteDataSet 6508
Provider.TDataPacketWriter.WriteMetaData 6509
Provider.TDataRequestEvent 6590
Provider.TDataRequestEvent type 6590
Provider.TDataSetProvider 6510
Provider.TDataSetProvider.ActiveUpdateException 6524
Provider.TDataSetProvider.AfterApplyUpdates 6529
Provider.TDataSetProvider.AfterExecute 6530
Provider.TDataSetProvider.AfterGetParams 6531
Provider.TDataSetProvider.AfterGetRecords 6531
Provider.TDataSetProvider.AfterRowRequest 6532
Provider.TDataSetProvider.AfterUpdateRecord 6533
Provider.TDataSetProvider.BeforeApplyUpdates 6533
Provider.TDataSetProvider.BeforeExecute 6534
Provider.TDataSetProvider.BeforeGetParams 6534
Provider.TDataSetProvider.BeforeGetRecords 6535
Provider.TDataSetProvider.BeforeRowRequest 6536
Provider.TDataSetProvider.BeforeUpdateRecord 6537
Provider.TDataSetProvider.CheckDataSet 6514
Provider.TDataSetProvider.Constraints 6525
Provider.TDataSetProvider.Create 6513
Provider.TDataSetProvider.CreateDataPacket 6515
Provider.TDataSetProvider.CreateResolver 6515
Provider.TDataSetProvider.DataSet 6525
Provider.TDataSetProvider.Destroy 6513
Provider.TDataSetProvider.DoBeforeExecute 6516
Provider.TDataSetProvider.DoBeforeGetRecords 6516
Provider.TDataSetProvider.DoGetProviderAttributes 6517
Provider.TDataSetProvider.DoGetTableName 6518
Provider.TDataSetProvider.Exported 6526
Provider.TDataSetProvider.FetchDetails 6518
Provider.TDataSetProvider.FindRecord 6518
Provider.TDataSetProvider.GetDataSetFromDelta 6519
Provider.TDataSetProvider.InternalApplyUpdates 6519
Provider.TDataSetProvider.InternalExecute 6520
Provider.TDataSetProvider.InternalGetParams 6520
Provider.TDataSetProvider.InternalGetRecords 6521
Provider.TDataSetProvider.InternalRowRequest 6522
Provider.TDataSetProvider.LocateRecord 6522
Provider.TDataSetProvider.Notification 6522
Provider.TDataSetProvider.OnDataRequest 6537
Provider.TDataSetProvider.OnGetData 6537
Provider.TDataSetProvider.OnGetDataSetProperties 6538
Provider.TDataSetProvider.OnGetTableName 6539
Provider.TDataSetProvider.OnUpdateData 6539
Provider.TDataSetProvider.OnUpdateError 6539
Provider.TDataSetProvider.OnValidate 6540
Provider.TDataSetProvider.Options 6526
Provider.TDataSetProvider.OwnerData 6527
Provider.TDataSetProvider.Params 6527
Provider.TDataSetProvider.Reset 6523
Provider.TDataSetProvider.ResolveToDataSet 6528
Provider.TDataSetProvider.SetCommandText 6523
Provider.TDataSetProvider.SetParams 6523
Provider.TDataSetProvider.TransactionStarted 6528
Provider.TDataSetProvider.UpdateMode 6528
Provider.TDataSetProvider.UpdateRecord 6523
Provider.TDataSetResolver 6540
Provider.TDataSetResolver.BeginUpdate 6542
Provider.TDataSetResolver.Create 6541
Provider.TDataSetResolver.DoDelete 6543
Provider.TDataSetResolver.DoInsert 6543
Provider.TDataSetResolver.DoUpdate 6543
Provider.TDataSetResolver.EndUpdate 6544
Provider.TDataSetResolver.InitializeConflictBuffer 6544
Provider.TDataSetResolver.InternalBeforeResolve 6545
Provider.TDataSetResolver.Provider 6545
Provider.TGetDSProps 6591
Provider.TGetDSProps type 6591
Provider.TGetParamsEvent 6591

Provider.TGetParamsEvent type 6591
Provider.TGetRecordOption 6584
Provider.TGetRecordOption enumeration 6584
Provider.TGetRecordOptions 6591
Provider.TGetRecordOptions type 6591
Provider.TGetTableNameEvent 6592
Provider.TGetTableNameEvent type 6592
Provider.TInfoArray 6592
Provider.TInfoArray type 6592
Provider.TLocalAppServer 6546
Provider.TLocalAppServer.AS_ApplyUpdates 6548
Provider.TLocalAppServer.AS_DataRequest 6549
Provider.TLocalAppServer.AS_Execute 6549
Provider.TLocalAppServer.AS_GetParams 6550
Provider.TLocalAppServer.AS_GetProviderNames 6551
Provider.TLocalAppServer.AS_GetRecords 6552
Provider.TLocalAppServer.AS_RowRequest 6553
Provider.TLocalAppServer.Create 6547
Provider.TLocalAppServer.Destroy 6547
Provider.TLocalAppServer.GetIDsOfNames 6554
Provider.TLocalAppServer.GetTypeInfo 6554
Provider.TLocalAppServer.GetTypeInfoCount 6554
Provider.TLocalAppServer.InterfaceSupportsErrorInfo 6554
Provider.TLocalAppServer.Invoke 6555
Provider.TLocalAppServer.SafeCallException 6555
Provider.TOnValidate 6592
Provider.TOnValidate type 6592
Provider.TPacketDataSet 6555
Provider.TPacketDataSet.AssignCurValues 6558
Provider.TPacketDataSet.Create 6556
Provider.TPacketDataSet.CreateFromDelta 6558
Provider.TPacketDataSet.DataEvent 6558
Provider.TPacketDataSet.GetStateFieldValue 6559
Provider.TPacketDataSet.HasCurValues 6559
Provider.TPacketDataSet.HasMergeConflicts 6559
Provider.TPacketDataSet.InitAltRecBuffers 6560
Provider.TPacketDataSet.InternalClose 6560
Provider.TPacketDataSet.InternalInitRecord 6560
Provider.TPacketDataSet.InternalOpen 6561
Provider.TPacketDataSet.NewValuesModified 6562
Provider.TPacketDataSet.SetFieldData 6561
Provider.TPacketDataSet.SetStateFieldValue 6561
Provider.TPacketDataSet.StreamMetaData 6562
Provider.TPacketDataSet.UpdateKind 6561
Provider.TPacketDataSet.UseCurValues 6562
Provider.TPacketDataSet.WritingCurValues 6563
Provider.TProvider 6563
Provider.TProviderDataEvent 6593
Provider.TProviderDataEvent type 6593
Provider.TProviderOption 6585
Provider.TProviderOption enumeration 6585
Provider.TProviderOptions 6593
Provider.TProviderOptions type 6593
Provider.TPutFieldInfo 6587
Provider.TPutFieldInfo record 6587
Provider.TPutFieldProc 6593
Provider.TPutFieldProc type 6593
Provider.TResolverErrorEvent 6594
Provider.TResolverErrorEvent type 6594
Provider.TResolverResponse 6587
Provider.TResolverResponse enumeration 6587
Provider.TSQLResolver 6564
Provider.TSQLResolver.Create 6565
Provider.TSQLResolver.Destroy 6565
Provider.TSQLResolver.DoDelete 6566
Provider.TSQLResolver.DoExecSQL 6567
Provider.TSQLResolver.DoGetValues 6567
Provider.TSQLResolver.DoInsert 6567
Provider.TSQLResolver.DoUpdate 6568
Provider.TSQLResolver.FreeTreeData 6568
Provider.TSQLResolver.GenDeleteSQL 6568
Provider.TSQLResolver.GenInsertSQL 6569
Provider.TSQLResolver.GenSelectSQL 6569
Provider.TSQLResolver.GenUpdateSQL 6569
Provider.TSQLResolver.GenWhereSQL 6569
Provider.TSQLResolver.GetProvider 6570
Provider.TSQLResolver.InitializeConflictBuffer 6570
Provider.TSQLResolver.InitTreeData 6570

Provider.TSQLResolver.InternalDoUpdate 6570
Provider.TSQLResolver.Provider 6571
Provider.TSQLResolver.UseFieldInUpdate 6571
Provider.TSQLResolver.UseFieldInWhere 6571
Provider.TUpdateTree 6572
Provider.TUpdateTree.Clear 6574
Provider.TUpdateTree.Create 6573
Provider.TUpdateTree.Data 6576
Provider.TUpdateTree.Delta 6576
Provider.TUpdateTree.Destroy 6573
Provider.TUpdateTree.DetailCount 6577
Provider.TUpdateTree.DetailList 6577
Provider.TUpdateTree.Details 6577
Provider.TUpdateTree.DoUpdates 6574
Provider.TUpdateTree.ErrorDS 6577
Provider.TUpdateTree.HasErrors 6578
Provider.TUpdateTree.InitData 6574
Provider.TUpdateTree.InitDelta 6575
Provider.TUpdateTree.InitErrorPacket 6575
Provider.TUpdateTree.IsNested 6578
Provider.TUpdateTree.Name 6578
Provider.TUpdateTree.Opened 6578
Provider.TUpdateTree.Parent 6578
Provider.TUpdateTree.RefreshData 6576
Provider.TUpdateTree.Resolver 6579
Provider.TUpdateTree.Source 6579
Provider.VarArrayFromStrings 6582
Provider.VarArrayFromStrings function 6582
Provider.VarArrayFromWideStrings 6583
Provider.VarArrayFromWideStrings function 6583
Provider.XMLOption 6597
Provider.XMLOption constant 6597
Provider.XMLUTF8Option 6598
Provider.XMLUTF8Option constant 6598
psAlternate enumeration member 2870
psClear enumeration member 2870
psDash enumeration member 2870
psDashDot enumeration member 2870
psDashDotDot enumeration member 2870
psDot enumeration member 2870
psEnding enumeration member 2872
psHandleDC enumeration member 6439
psHandleIC enumeration member 6439
psInsideFrame enumeration member 2870
psNoHandle enumeration member 6439
psRunning enumeration member 2872
psSolid enumeration member 2870
psStarting enumeration member 2872
psUserStyle enumeration member 2870
ptEndOfPeriod enumeration member 4772
ptStartOfPeriod enumeration member 4772

Q

qpBottom enumeration member 6836
qpTop enumeration member 6836
qsCustomize enumeration member 7289
qsMoreControls enumeration member 7289

R

rdBinary enumeration member 6691
rdExpandString enumeration member 6691
rdInteger enumeration member 6691
rdString enumeration member 6691
rdUnknown enumeration member 6691
reApplicationMenu enumeration member 7383
reButton enumeration member 7383
reCheckBox enumeration member 7383
reCombo enumeration member 7383
RecoverTwoPhaseGlobal enumeration member 4024
reForm enumeration member 7383
reGallery enumeration member 7383
Registry 6599
Registry namespace 6599
    Classes 6600
    Structs, Records, Enums 6691
Registry.ERegistryException 6600
Registry.TRegDataInfo 6691
Registry.TRegDataInfo record 6691

Registry.TRegDataType 6691
Registry.TRegDataType enumeration 6691
Registry.TRegIniFile 6601
Registry.TRegIniFile.Create 6603
Registry.TRegIniFile.DeleteKey 6604
Registry.TRegIniFile.EraseSection 6605
Registry.TRegIniFile.FileName 6610
Registry.TRegIniFile.ReadBool 6605
Registry.TRegIniFile.ReadInteger 6606
Registry.TRegIniFile.ReadSection 6606
Registry.TRegIniFile.ReadSections 6607
Registry.TRegIniFile.ReadSectionValues 6607
Registry.TRegIniFile.ReadString 6608
Registry.TRegIniFile.WriteBool 6608
Registry.TRegIniFile.WriteInteger 6609
Registry.TRegIniFile.WriteString 6609
Registry.TRegistry 6610
Registry.TRegistry.Access 6657
Registry.TRegistry.ChangeKey 6619
Registry.TRegistry.CloseKey 6619
Registry.TRegistry.Create 6615
Registry.TRegistry.CreateKey 6622
Registry.TRegistry.CurrentKey 6660
Registry.TRegistry.CurrentPath 6660
Registry.TRegistry.DeleteKey 6623
Registry.TRegistry.DeleteValue 6625
Registry.TRegistry.Destroy 6617
Registry.TRegistry.GetBaseKey 6626
Registry.TRegistry.GetData 6626
Registry.TRegistry.GetDataAsString 6626
Registry.TRegistry.GetDataInfo 6627
Registry.TRegistry.GetDataSize 6627
Registry.TRegistry.GetDataType 6628
Registry.TRegistry.GetKey 6628
Registry.TRegistry.GetKeyInfo 6628
Registry.TRegistry.GetKeyNames 6629
Registry.TRegistry.GetValueNames 6629
Registry.TRegistry.HasSubKeys 6630
Registry.TRegistry.KeyExists 6630
Registry.TRegistry.LazyWrite 6660
Registry.TRegistry.LoadKey 6632
Registry.TRegistry.MoveKey 6633
Registry.TRegistry.OpenKey 6634
Registry.TRegistry.OpenKeyReadOnly 6636
Registry.TRegistry.PutData 6637
Registry.TRegistry.ReadBinaryData 6637
Registry.TRegistry.ReadBool 6638
Registry.TRegistry.ReadCurrency 6638
Registry.TRegistry.ReadDate 6638
Registry.TRegistry.ReadDateTime 6639
Registry.TRegistry.ReadFloat 6639
Registry.TRegistry.ReadInteger 6640
Registry.TRegistry.ReadString 6640
Registry.TRegistry.ReadTime 6641
Registry.TRegistry.RegistryConnect 6641
Registry.TRegistry.RenameValue 6642
Registry.TRegistry.ReplaceKey 6642
Registry.TRegistry.RestoreKey 6643
Registry.TRegistry.RootKey 6661
Registry.TRegistry.SaveKey 6643
Registry.TRegistry.SetCurrentKey 6644
Registry.TRegistry.UnLoadKey 6644
Registry.TRegistry.ValueExists 6645
Registry.TRegistry.WriteBinaryData 6645
Registry.TRegistry.WriteBool 6646
Registry.TRegistry.WriteCurrency 6646
Registry.TRegistry.WriteDate 6646
Registry.TRegistry.WriteDateTime 6647
Registry.TRegistry.WriteExpandString 6650
Registry.TRegistry.WriteFloat 6652
Registry.TRegistry.WriteInteger 6653
Registry.TRegistry.WriteString 6653
Registry.TRegistry.WriteTime 6656
Registry.TRegistryIniFile 6663
Registry.TRegistryIniFile.Create 6668
Registry.TRegistryIniFile.DeleteKey 6669
Registry.TRegistryIniFile.Destroy 6668
Registry.TRegistryIniFile.EraseSection 6670

Registry.TRegistryIniFile.ReadBinaryStream 6670
Registry.TRegistryIniFile.ReadDate 6671
Registry.TRegistryIniFile.ReadDateTime 6671
Registry.TRegistryIniFile.ReadFloat 6672
Registry.TRegistryIniFile.ReadInteger 6672
Registry.TRegistryIniFile.ReadSection 6676
Registry.TRegistryIniFile.ReadSections 6676
Registry.TRegistryIniFile.ReadSectionValues 6677
Registry.TRegistryIniFile.ReadString 6677
Registry.TRegistryIniFile.ReadTime 6680
Registry.TRegistryIniFile.RegIniFile 6690
Registry.TRegistryIniFile.UpdateFile 6681
Registry.TRegistryIniFile.WriteBinaryStream 6681
Registry.TRegistryIniFile.WriteDate 6682
Registry.TRegistryIniFile.WriteDateTime 6682
Registry.TRegistryIniFile.WriteFloat 6683
Registry.TRegistryIniFile.WriteInteger 6683
Registry.TRegistryIniFile.WriteString 6686
Registry.TRegistryIniFile.WriteTime 6689
Registry.TRegKeyInfo 6692
Registry.TRegKeyInfo record 6692
reGroup enumeration member 7383
reMenu enumeration member 7383
Replace enumeration member 4021
ReqFiles 6695
ReqFiles namespace 6695
Classes 6696
ReqFiles.TWebRequestFile 6696
ReqFiles.TWebRequestFile.Create 6697
ReqFiles.TWebRequestFile.Destroy 6698
ReqFiles.TWebRequestFile.GetContentType 6698
ReqFiles.TWebRequestFile.GetFieldName 6699
ReqFiles.TWebRequestFile.GetFileName 6699
ReqFiles.TWebRequestFile.GetStream 6699
ReqFiles.TWebRequestFiles 6700
ReqFiles.TWebRequestFiles.Add 6702
ReqFiles.TWebRequestFiles.Create 6701
ReqFiles.TWebRequestFiles.Destroy 6701
ReqFiles.TWebRequestFiles.GetCount 6703
ReqFiles.TWebRequestFiles.GetItem 6703
ReqFiles.TWebRequestFileStream 6703
ReqFiles.TWebRequestFileStream.Create 6704
ReqFiles.TWebRequestFileStream.Write 6704
ReqMulti 6705
ReqMulti namespace 6705
Classes 6706
ReqMulti.TMultipartContentParser 6706
ReqMulti.TMultipartContentParser.BufferContent 6708
ReqMulti.TMultipartContentParser.CanParse 6708
ReqMulti.TMultipartContentParser.Destroy 6707
ReqMulti.TMultipartContentParser.ExtractContentTypeFields 6709
ReqMulti.TMultipartContentParser.ExtractMultipartContentFields 6709
ReqMulti.TMultipartContentParser.GetContentFields 6709
ReqMulti.TMultipartContentParser.GetFiles 6709
ReqMulti.TMultipartContentParser.ParseMultiPart 6710
ReqMulti.TMultipartContentParser.ParseMultiPartContent 6710
ReqMulti.TMultipartContentParser.ParseMultipartHeaders 6710
reQuickAccessToolbar enumeration member 7383
reRadioButton enumeration member 7383
reRibbon enumeration member 7383
reScrollBar enumeration member 7383
reSpinner enumeration member 7383
reStatusBar enumeration member 7383
reTab enumeration member 7383
rgvCollapsed enumeration member 6937
rgvLarge enumeration member 6937
rgvMedium enumeration member 6937
rgvSmall enumeration member 6937
Ribbon 6713
Ribbon namespace 6713
Classes 6715
Interfaces 6931
Structs, Records, Enums 6936
Types 6940
Ribbon.ERibbonException 6715
Ribbon.IRibbon 6931

Ribbon.IRibbon.GetRibbon 6932
Ribbon.IRibbon.Ribbon 6932
Ribbon.IRibbon.SetRibbon 6932
Ribbon.IRibbonGroup 6933
Ribbon.IRibbonGroup.AlignGroupControls 6934
Ribbon.IRibbonGroup.CreateControl 6934
Ribbon.IRibbonGroup.StyleChanged 6934
Ribbon.IRibbonQuickAccessToolbar 6934
Ribbon.IRibbonQuickAccessToolbar.DoUpdateQATWidth 6935
Ribbon.TApplicationMenu 6716
Ribbon.TApplicationMenu.Assign 6718
Ribbon.TApplicationMenu.Caption 6719
Ribbon.TApplicationMenu.CommandType 6720
Ribbon.TApplicationMenu.Create 6717
Ribbon.TApplicationMenu.Destroy 6717
Ribbon.TApplicationMenu.GetActionClientItem 6718
Ribbon.TApplicationMenu.Icon 6720
Ribbon.TApplicationMenu.IconSize 6720
Ribbon.TApplicationMenu.Initialize 6719
Ribbon.TApplicationMenu.KeyTip 6720
Ribbon.TApplicationMenu.Menu 6721
Ribbon.TApplicationMenu.ScreenTip 6721
Ribbon.TApplicationMenu.Show 6719
Ribbon.TCommandType 6936
Ribbon.TCommandType enumeration 6936
Ribbon.TCustomActionControlBar 6721
Ribbon.TCustomActionControlBar.CMControlChange 6722
Ribbon.TCustomActionControlBar.Create 6722
Ribbon.TCustomActionControlBar.CreateControl 6722
Ribbon.TCustomActionControlBar.CreateControls 6723
Ribbon.TCustomActionControlBar.CreatingControls 6723
Ribbon.TCustomActionControlBar.GetControlClass 6723
Ribbon.TCustomRibbon 6724
Ribbon.TCustomRibbon.ActionManager 6759
Ribbon.TCustomRibbon.Active 6759
Ribbon.TCustomRibbon.ActivePage 6760
Ribbon.TCustomRibbon.AddRecentItem 6735
Ribbon.TCustomRibbon.AddTab 6736
Ribbon.TCustomRibbon.AllowKeyboardNavigation 6760
Ribbon.TCustomRibbon.ApplicationMenu 6760
Ribbon.TCustomRibbon.BeginUpdate 6736
Ribbon.TCustomRibbon.CheckRibbonSize 6736
Ribbon.TCustomRibbon.ClearRecentItems 6736
Ribbon.TCustomRibbon.CMBidimodechanged 6736
Ribbon.TCustomRibbon.CMDesignHitTest 6737
Ribbon.TCustomRibbon.CMFontchanged 6737
Ribbon.TCustomRibbon.CMMouseenter 6737
Ribbon.TCustomRibbon.CMMouseleave 6737
Ribbon.TCustomRibbon.CMTextchanged 6738
Ribbon.TCustomRibbon.CMThemeChanged 6738
Ribbon.TCustomRibbon.ColorMap 6761
Ribbon.TCustomRibbon.ContextMenuPopupHandler 6738
Ribbon.TCustomRibbon.Create 6728
Ribbon.TCustomRibbon.CreateContextMenu 6738
Ribbon.TCustomRibbon.cRibbonFirstTabOffSet 6730
Ribbon.TCustomRibbon.cRibbonGroupCaptionHeight 6730
Ribbon.TCustomRibbon.cRibbonGroupHeight 6730
Ribbon.TCustomRibbon.cRibbonHeight 6730
Ribbon.TCustomRibbon.cRibbonHideHeight 6730
Ribbon.TCustomRibbon.cRibbonHideWidth 6731
Ribbon.TCustomRibbon.cRibbonMinimizedHeight 6731
Ribbon.TCustomRibbon.cRibbonMinimumCaptionWidth 6731
Ribbon.TCustomRibbon.cRibbonPageHeight 6731
Ribbon.TCustomRibbon.cRibbonQATHeight 6732
Ribbon.TCustomRibbon.cRibbonQuickAccessToolbarLeft 6732
Ribbon.TCustomRibbon.cRibbonQuickAccessToolbarTop 6732
Ribbon.TCustomRibbon.cRibbonTabHeight 6732
Ribbon.TCustomRibbon.cRibbonTabScrollButtonWidth 6732
Ribbon.TCustomRibbon.cRibbonTabSpacing 6733
Ribbon.TCustomRibbon.cRibbonUnthemedCaptionHeight 6733
Ribbon.TCustomRibbon.CustomFrameSupported 6738
Ribbon.TCustomRibbon.CustomFrameWndProc 6739
Ribbon.TCustomRibbon.DblClick 6739
Ribbon.TCustomRibbon.DefineProperties 6739
Ribbon.TCustomRibbon.DeleteTab 6740
Ribbon.TCustomRibbon.Destroy 6728
Ribbon.TCustomRibbon.DisplayKeyTips 6740
Ribbon.TCustomRibbon.DoAddToQuickAccessToolbar 6740

Ribbon.TCustomRibbon.DoChangeQuickAccessToolbarPosition 6740
Ribbon.TCustomRibbon.DoContextPopup 6741
Ribbon.TCustomRibbon.DocumentName 6761
Ribbon.TCustomRibbon.DoCustomizeQuickAccessToolbar 6741
Ribbon.TCustomRibbon.DoGroupStyleChanged 6741
Ribbon.TCustomRibbon.DoHelpButtonClick 6741
Ribbon.TCustomRibbon.DoMinimizeRibbon 6742
Ribbon.TCustomRibbon.DoMouseWheelDown 6742
Ribbon.TCustomRibbon.DoMouseWheelEx 6742
Ribbon.TCustomRibbon.DoMouseWheelUp 6743
Ribbon.TCustomRibbon.DoRecentItemClick 6743
Ribbon.TCustomRibbon.DoScrollTabsLeft 6743
Ribbon.TCustomRibbon.DoScrollTabsRight 6743
Ribbon.TCustomRibbon.DoTabChange 6744
Ribbon.TCustomRibbon.DrawCaption 6744
Ribbon.TCustomRibbon.DrawQuickAccessToolbar 6744
Ribbon.TCustomRibbon.DrawSystemButtons 6744
Ribbon.TCustomRibbon.DrawTab 6745
Ribbon.TCustomRibbon.EndUpdate 6745
Ribbon.TCustomRibbon.GetContextMenuOptions 6745
Ribbon.TCustomRibbon.GetHelpButtonRect 6745
Ribbon.TCustomRibbon.GetIcon 6745
Ribbon.TCustomRibbon.GetRibbonHeight 6746
Ribbon.TCustomRibbon.GetRibbonMetric 6746
Ribbon.TCustomRibbon.GetTabAt 6746
Ribbon.TCustomRibbon.GetTabRect 6746
Ribbon.TCustomRibbon.GetTabWidth 6747
Ribbon.TCustomRibbon.HasApplicationMenu 6747
Ribbon.TCustomRibbon.HasQuickAccessToolbar 6747
Ribbon.TCustomRibbon.HideRibbon 6761
Ribbon.TCustomRibbon.HideTabs 6762
Ribbon.TCustomRibbon.HotTabIndex 6762
Ribbon.TCustomRibbon.Idle 6747
Ribbon.TCustomRibbon.InvalidateHeaderArea 6747
Ribbon.TCustomRibbon.InvalidateScrollButtons 6748
Ribbon.TCustomRibbon.InvalidateSystemMenuArea 6748
Ribbon.TCustomRibbon.InvalidateTabArea 6748
Ribbon.TCustomRibbon.KeyDown 6749
Ribbon.TCustomRibbon.KeyTipsVisible 6762
Ribbon.TCustomRibbon.Loaded 6749
Ribbon.TCustomRibbon.Menu 6762
Ribbon.TCustomRibbon.Minimized 6763
Ribbon.TCustomRibbon.MouseDown 6750
Ribbon.TCustomRibbon.MouseInSystemMenu 6750
Ribbon.TCustomRibbon.MouseMove 6750
Ribbon.TCustomRibbon.MouseUp 6751
Ribbon.TCustomRibbon.MoveGroup 6751
Ribbon.TCustomRibbon.Notification 6752
Ribbon.TCustomRibbon.OnHelpButtonClick 6766
Ribbon.TCustomRibbon.OnLeftTabScrollButtonClick 6766
Ribbon.TCustomRibbon.OnRecentItemClick 6766
Ribbon.TCustomRibbon.OnRightTabScrollButtonClick 6767
Ribbon.TCustomRibbon.OnTabChange 6767
Ribbon.TCustomRibbon.Paint 6752
Ribbon.TCustomRibbon.QuickAccessToolbar 6763
Ribbon.TCustomRibbon.RebuildRibbonMetrics 6753
Ribbon.TCustomRibbon.RemoveAllTabs 6753
Ribbon.TCustomRibbon.RemoveRecentItem 6753
Ribbon.TCustomRibbon.RemoveTab 6753
Ribbon.TCustomRibbon.Resize 6754
Ribbon.TCustomRibbon.ScaleFactor 6763
Ribbon.TCustomRibbon.ScreenTips 6763
Ribbon.TCustomRibbon.SetBounds 6754
Ribbon.TCustomRibbon.SetParent 6754
Ribbon.TCustomRibbon.ShowHelpButton 6764
Ribbon.TCustomRibbon.Style 6764
Ribbon.TCustomRibbon.TabHandledKeyTips 6755
Ribbon.TCustomRibbon.TabHideKeyTips 6755
Ribbon.TCustomRibbon.TabIndex 6764
Ribbon.TCustomRibbon.Tabs 6764
Ribbon.TCustomRibbon.TabShowKeyTips 6756
Ribbon.TCustomRibbon.TTabChangeEvent 6765
Ribbon.TCustomRibbon.UM_CHECKSIZE 6733
Ribbon.TCustomRibbon.UM_DISPLAYKEYTIPS 6733
Ribbon.TCustomRibbon.UMCheckResize 6756
Ribbon.TCustomRibbon.UMDisplayKeyTips 6756
Ribbon.TCustomRibbon.UpdateQATPosition 6757

Ribbon.TCustomRibbon.UpdateSystemMenuInfo 6757
Ribbon.TCustomRibbon.UseCustomFrame 6765
Ribbon.TCustomRibbon.ValidTab 6757
Ribbon.TCustomRibbon.WMEraseBkgnd 6757
Ribbon.TCustomRibbon.WMGetDlgCode 6758
Ribbon.TCustomRibbon.WMKillFocus 6758
Ribbon.TCustomRibbon.WMSettingChange 6758
Ribbon.TCustomRibbon.WMWindowPosChanged 6758
Ribbon.TCustomRibbonClass 6940
Ribbon.TCustomRibbonClass type 6940
Ribbon.TCustomRibbonColorMap 6767
Ribbon.TCustomRibbonColorMap.ActiveTabFontColor 6768
Ribbon.TCustomRibbonColorMap.BevelMinimizedInner 6768
Ribbon.TCustomRibbonColorMap.BevelMinimizedOuter 6769
Ribbon.TCustomRibbonColorMap.CaptionFontColor 6769
Ribbon.TCustomRibbonColorMap.ControlColor 6769
Ribbon.TCustomRibbonColorMap.ControlFrameColor 6769
Ribbon.TCustomRibbonColorMap.ControlInactiveColor 6769
Ribbon.TCustomRibbonColorMap.DocumentFontColor 6770
Ribbon.TCustomRibbonColorMap.GroupFontColor 6770
Ribbon.TCustomRibbonColorMap.InactiveTabFontColor 6770
Ribbon.TCustomRibbonColorMap.QuickAccessToolbarMoreBorderColor 6770
Ribbon.TCustomRibbonColorMap.QuickAccessToolbarMoreColor 6771
Ribbon.TCustomRibbonGroup 6771
Ribbon.TCustomRibbonGroup.AlignControls 6775
Ribbon.TCustomRibbonGroup.AlignGroupControls 6775
Ribbon.TCustomRibbonGroup.Assign 6775
Ribbon.TCustomRibbonGroup.AutoSizingChanged 6776
Ribbon.TCustomRibbonGroup.BarHandleKeyTips 6776
Ribbon.TCustomRibbonGroup.BarHideKeyTips 6776
Ribbon.TCustomRibbonGroup.BarShowKeyTips 6777
Ribbon.TCustomRibbonGroup.CanCollapse 6786
Ribbon.TCustomRibbonGroup.Clone 6777
Ribbon.TCustomRibbonGroup.CMFontchanged 6777
Ribbon.TCustomRibbonGroup.CMItemDropPoint 6777
Ribbon.TCustomRibbonGroup.CMMouseenter 6778
Ribbon.TCustomRibbonGroup.CMMouseleave 6778
Ribbon.TCustomRibbonGroup.CMTextchanged 6778
Ribbon.TCustomRibbonGroup.CollapsedControl 6786
Ribbon.TCustomRibbonGroup.Create 6773
Ribbon.TCustomRibbonGroup.CreateControl 6778
Ribbon.TCustomRibbonGroup.CreateControls 6779
Ribbon.TCustomRibbonGroup.Destroy 6773
Ribbon.TCustomRibbonGroup.DialogAction 6786
Ribbon.TCustomRibbonGroup.DockDrop 6779
Ribbon.TCustomRibbonGroup.DoDropItem 6779
Ribbon.TCustomRibbonGroup.DoEndDrag 6779
Ribbon.TCustomRibbonGroup.DragDrop 6779
Ribbon.TCustomRibbonGroup.DrawShowDialogButton 6780
Ribbon.TCustomRibbonGroup.GetCaptionHeight 6780
Ribbon.TCustomRibbonGroup.GetGroupHeight 6780
Ribbon.TCustomRibbonGroup.GetKeyTip 6780
Ribbon.TCustomRibbonGroup.GroupAlign 6787
Ribbon.TCustomRibbonGroup.GroupIndex 6787
Ribbon.TCustomRibbonGroup.GroupVariant 6787
Ribbon.TCustomRibbonGroup.GroupWidth 6788
Ribbon.TCustomRibbonGroup.HandleKeyTip 6781
Ribbon.TCustomRibbonGroup.HasGallery 6788
Ribbon.TCustomRibbonGroup.HideControls 6781
Ribbon.TCustomRibbonGroup.HideKeyTip 6781
Ribbon.TCustomRibbonGroup.Items 6788
Ribbon.TCustomRibbonGroup.KeyTip 6788
Ribbon.TCustomRibbonGroup.KeyTipClick 6782
Ribbon.TCustomRibbonGroup.MouseDown 6782
Ribbon.TCustomRibbonGroup.MouseMove 6782
Ribbon.TCustomRibbonGroup.MouseUp 6783
Ribbon.TCustomRibbonGroup.Notification 6783
Ribbon.TCustomRibbonGroup.Page 6789
Ribbon.TCustomRibbonGroup.Paint 6783
Ribbon.TCustomRibbonGroup.RequiredWidth 6789
Ribbon.TCustomRibbonGroup.Resize 6783
Ribbon.TCustomRibbonGroup.Rows 6789
Ribbon.TCustomRibbonGroup.SetAutoSizing 6783
Ribbon.TCustomRibbonGroup.SetBounds 6784
Ribbon.TCustomRibbonGroup.SetKeyTip 6784
Ribbon.TCustomRibbonGroup.SetParent 6784

Ribbon.TCustomRibbonGroup.SetupDropDownBtn 6785
Ribbon.TCustomRibbonGroup.ShowControls 6785
Ribbon.TCustomRibbonGroup.ShowKeyTip 6785
Ribbon.TCustomRibbonGroup.StyleChanged 6785
Ribbon.TCustomRibbonGroup.TGroupAlign 6774
Ribbon.TCustomRibbonGroup.TGroupAlign enumeration 6774
Ribbon.TCustomRibbonGroup.TRowPosition 6790
Ribbon.TCustomRibbonPage 6790
Ribbon.TCustomRibbonPage.AddGroup 6794
Ribbon.TCustomRibbonPage.AlignGroups 6794
Ribbon.TCustomRibbonPage.CMControlListChanging 6795
Ribbon.TCustomRibbonPage.CMTextchanged 6795
Ribbon.TCustomRibbonPage.CollapseGroup 6795
Ribbon.TCustomRibbonPage.CollapseGroupCount 6803
Ribbon.TCustomRibbonPage.Create 6792
Ribbon.TCustomRibbonPage.CreateScrollButton 6795
Ribbon.TCustomRibbonPage.cRibbonGroupLeftOffset 6793
Ribbon.TCustomRibbonPage.cRibbonGroupSpacing 6793
Ribbon.TCustomRibbonPage.Destroy 6792
Ribbon.TCustomRibbonPage.DoScrollButtonClick 6795
Ribbon.TCustomRibbonPage.FindGroup 6796
Ribbon.TCustomRibbonPage.FindGroupAt 6796
Ribbon.TCustomRibbonPage.GetGroupPos 6796
Ribbon.TCustomRibbonPage.GetNextVisibleGroup 6796
Ribbon.TCustomRibbonPage.GetPageHeight 6797
Ribbon.TCustomRibbonPage.GetPageTop 6797
Ribbon.TCustomRibbonPage.GroupCount 6803
Ribbon.TCustomRibbonPage.GroupOffset 6803
Ribbon.TCustomRibbonPage.Groups 6803
Ribbon.TCustomRibbonPage.HandleKeyTips 6797
Ribbon.TCustomRibbonPage.HideKeyTips 6797
Ribbon.TCustomRibbonPage.Index 6803
Ribbon.TCustomRibbonPage.Invalidate 6797
Ribbon.TCustomRibbonPage.KeyTip 6804
Ribbon.TCustomRibbonPage.Loaded 6798
Ribbon.TCustomRibbonPage.Notification 6798
Ribbon.TCustomRibbonPage.Paint 6799
Ribbon.TCustomRibbonPage.RequestAlign 6799
Ribbon.TCustomRibbonPage.Resize 6800
Ribbon.TCustomRibbonPage.Ribbon 6804
Ribbon.TCustomRibbonPage.RibbonStyle 6800
Ribbon.TCustomRibbonPage.SetBounds 6800
Ribbon.TCustomRibbonPage.SetParent 6801
Ribbon.TCustomRibbonPage.ShowKeyTips 6802
Ribbon.TCustomRibbonPage.UpdateActionManager 6802
Ribbon.TCustomRibbonPage.UpdateScrollButtons 6802
Ribbon.TCustomRibbonQuickAccessPopupToolbar 6804
Ribbon.TCustomRibbonQuickAccessPopupToolbar.Create 6805
Ribbon.TCustomRibbonQuickAccessPopupToolbar.CreateControl 6806
Ribbon.TCustomRibbonQuickAccessPopupToolbar.CreateWnd 6806
Ribbon.TCustomRibbonQuickAccessPopupToolbar.DrawBackground 6806
Ribbon.TCustomRibbonQuickAccessPopupToolbar.GetControlClass 6806
Ribbon.TCustomRibbonQuickAccessPopupToolbar.NCPaint 6807
Ribbon.TCustomRibbonQuickAccessPopupToolbar.ValidOrientations 6807
Ribbon.TCustomRibbonQuickAccessToolbar 6807
Ribbon.TCustomRibbonQuickAccessToolbar.BarHandleKeyTips 6810
Ribbon.TCustomRibbonQuickAccessToolbar.BarHideKeyTips 6810
Ribbon.TCustomRibbonQuickAccessToolbar.BarShowKeyTips 6811
Ribbon.TCustomRibbonQuickAccessToolbar.CMFontchanged 6811
Ribbon.TCustomRibbonQuickAccessToolbar.Create 6809
Ribbon.TCustomRibbonQuickAccessToolbar.CreateControl 6811
Ribbon.TCustomRibbonQuickAccessToolbar.CreateControls 6811
Ribbon.TCustomRibbonQuickAccessToolbar.Customize 6812
Ribbon.TCustomRibbonQuickAccessToolbar.DoContextPopup 6812
Ribbon.TCustomRibbonQuickAccessToolbar.DoDropActions 6812
Ribbon.TCustomRibbonQuickAccessToolbar.DoDropCategory

6813
Ribbon.TCustomRibbonQuickAccessToolbar.DoDropItem 6813
Ribbon.TCustomRibbonQuickAccessToolbar.DoMoreCommands 6813
Ribbon.TCustomRibbonQuickAccessToolbar.DoScrollBtnClick 6813
Ribbon.TCustomRibbonQuickAccessToolbar.DoUpdateQATWidth 6813
Ribbon.TCustomRibbonQuickAccessToolbar.DragDrop 6814
Ribbon.TCustomRibbonQuickAccessToolbar.DrawBackground 6814
Ribbon.TCustomRibbonQuickAccessToolbar.GetBarHeight 6814
Ribbon.TCustomRibbonQuickAccessToolbar.GetBarWidth 6815
Ribbon.TCustomRibbonQuickAccessToolbar.GetScrollBtnClass 6815
Ribbon.TCustomRibbonQuickAccessToolbar.Items 6818
Ribbon.TCustomRibbonQuickAccessToolbar.Notification 6815
Ribbon.TCustomRibbonQuickAccessToolbar.OnMoreCommands 6818
Ribbon.TCustomRibbonQuickAccessToolbar.RequestAlign 6815
Ribbon.TCustomRibbonQuickAccessToolbar.Ribbon 6818
Ribbon.TCustomRibbonQuickAccessToolbar.ScrollMode 6818
Ribbon.TCustomRibbonQuickAccessToolbar.SetParent 6816
Ribbon.TCustomRibbonQuickAccessToolbar.SetupDropDownBtn 6816
Ribbon.TCustomRibbonQuickAccessToolbar.ShowBorder 6819
Ribbon.TCustomRibbonQuickAccessToolbar.ShowCustomizeMenu 6817
Ribbon.TCustomRibbonQuickAccessToolbar.ShowMoreControls 6817
Ribbon.TCustomRibbonQuickAccessToolbar.TMoreCommandsEvent 6819
Ribbon.TCustomRibbonQuickAccessToolbar.UpdateQATWidth 6817
Ribbon.TCustomTabData 6819
Ribbon.TCustomTabData.Add 6821
Ribbon.TCustomTabData.AvailableSpace 6822
Ribbon.TCustomTabData.CalculateTabWidths 6821
Ribbon.TCustomTabData.Clear 6821
Ribbon.TCustomTabData.Create 6820
Ribbon.TCustomTabData.Destroy 6821
Ribbon.TCustomTabData.FirstVisibleTabIndex 6822
Ribbon.TCustomTabData.FullWidth 6822
Ribbon.TCustomTabData.ItemCount 6823
Ribbon.TCustomTabData.LastVisibleTabIndex 6823
Ribbon.TCustomTabData.LeftTabScrollButtonRect 6823
Ribbon.TCustomTabData.NewWidth 6823
Ribbon.TCustomTabData.Owner 6824
Ribbon.TCustomTabData.RightTabScrollButtonRect 6824
Ribbon.TCustomTabData.ScrollButtons 6824
Ribbon.TCustomTabData.SeparatorOpacity 6824
Ribbon.TCustomTabData.TabSeparatorState 6824
Ribbon.TCustomTabData.TabUpdateRect 6825
Ribbon.THelpButton 6825
Ribbon.THelpButton.DrawImage 6826
Ribbon.TIconSize 6936
Ribbon.TIconSize enumeration 6936
Ribbon.TOffice2007Button 6826
Ribbon.TOffice2007Button.CMMouseenter 6829
Ribbon.TOffice2007Button.CMMouseleave 6829
Ribbon.TOffice2007Button.Create 6827
Ribbon.TOffice2007Button.Destroy 6828
Ribbon.TOffice2007Button.DoClick 6829
Ribbon.TOffice2007Button.DrawBackground 6829
Ribbon.TOffice2007Button.DrawImage 6830
Ribbon.TOffice2007Button.MouseDown 6830
Ribbon.TOffice2007Button.MouseInControl 6832
Ribbon.TOffice2007Button.MouseMove 6830
Ribbon.TOffice2007Button.MouseUp 6831
Ribbon.TOffice2007Button.OnClick 6833
Ribbon.TOffice2007Button.Paint 6831
Ribbon.TOffice2007Button.RibbonStyle 6833
Ribbon.TOffice2007Button.SetParent 6832
Ribbon.TQuickAccessToolbar 6835
Ribbon.TQuickAccessToolbar.ActionBar 6839
Ribbon.TQuickAccessToolbar.Assign 6837
Ribbon.TQuickAccessToolbar.BarWidth 6839

Ribbon.TQuickAccessToolbar.Create 6836
Ribbon.TQuickAccessToolbar.DoChange 6838
Ribbon.TQuickAccessToolbar.OnChange 6840
Ribbon.TQuickAccessToolbar.Position 6839
Ribbon.TQuickAccessToolbar.Show 6838
Ribbon.TQuickAccessToolbar.ShowMinimize 6839
Ribbon.TQuickAccessToolbar.ShowMoreCommands 6840
Ribbon.TQuickAccessToolbar.ShowPosition 6840
Ribbon.TQuickAccessToolbar.TogglePosition 6838
Ribbon.TQuickAccessToolbar.TQATPosition 6836
Ribbon.TQuickAccessToolbar.TQATPosition enumeration 6836
Ribbon.TQuickAccessToolbarGroupItem 6841
Ribbon.TQuickAccessToolbarGroupItem.Action 6842
Ribbon.TQuickAccessToolbarGroupItem.Caption 6842
Ribbon.TQuickAccessToolbarGroupItem.Create 6841
Ribbon.TQuickAccessToolbarGroupItem.Destroy 6842
Ribbon.TQuickAccessToolbarGroupItem.Group 6842
Ribbon.TQuickAccessToolbarGroups 6843
Ribbon.TQuickAccessToolbarGroups.Add 6844
Ribbon.TQuickAccessToolbarGroups.Clean 6844
Ribbon.TQuickAccessToolbarGroups.Create 6843
Ribbon.TQuickAccessToolbarGroups.Destroy 6844
Ribbon.TQuickAccessToolbarGroups.Exists 6844
Ribbon.TQuickAccessToolbarGroups.Group 6845
Ribbon.TQuickAccessToolbarGroups.Remove 6845
Ribbon.TRecentItemClickEvent 6940
Ribbon.TRecentItemClickEvent type 6940
Ribbon.TRibbon 6845
Ribbon.TRibbon.ActionManager 6849
Ribbon.TRibbon.Align 6850
Ribbon.TRibbon.Anchors 6853
Ribbon.TRibbon.ApplicationMenu 6854
Ribbon.TRibbon.BiDiMode 6854
Ribbon.TRibbon.Caption 6854
Ribbon.TRibbon.DocumentName 6857
Ribbon.TRibbon.Enabled 6857
Ribbon.TRibbon.Font 6858
Ribbon.TRibbon.Height 6859
Ribbon.TRibbon.HideTabs 6860
Ribbon.TRibbon.OnHelpButtonClick 6863
Ribbon.TRibbon.OnRecentItemClick 6863
Ribbon.TRibbon.OnTabChange 6864
Ribbon.TRibbon.ParentBiDiMode 6860
Ribbon.TRibbon.ParentFont 6861
Ribbon.TRibbon.QuickAccessToolbar 6861
Ribbon.TRibbon.ScreenTips 6861
Ribbon.TRibbon.ShowHelpButton 6862
Ribbon.TRibbon.Style 6862
Ribbon.TRibbon.TabIndex 6862
Ribbon.TRibbon.Tabs 6862
Ribbon.TRibbon.UseCustomFrame 6863
Ribbon.TRibbonCustomizeDlg 6864
Ribbon.TRibbonCustomizeDlg.SetupDlg 6865
Ribbon.TRibbonGroup 6865
Ribbon.TRibbonGroup.ActionManager 6868
Ribbon.TRibbonGroup.BiDiMode 6868
Ribbon.TRibbonGroup.Caption 6869
Ribbon.TRibbonGroup.DialogAction 6869
Ribbon.TRibbonGroup.GroupAlign 6869
Ribbon.TRibbonGroup.GroupIndex 6869
Ribbon.TRibbonGroup.Height 6870
Ribbon.TRibbonGroup.KeyTip 6870
Ribbon.TRibbonGroup.ParentBiDiMode 6870
Ribbon.TRibbonGroup.Rows 6870
Ribbon.TRibbonGroupCollapsedControl 6871
Ribbon.TRibbonGroupCollapsedControl.ActionManager 6880
Ribbon.TRibbonGroupCollapsedControl.AdjustBounds 6874
Ribbon.TRibbonGroupCollapsedControl.Caption 6880
Ribbon.TRibbonGroupCollapsedControl.Click 6874
Ribbon.TRibbonGroupCollapsedControl.CloseUp 6875
Ribbon.TRibbonGroupCollapsedControl.CMFontchanged 6875
Ribbon.TRibbonGroupCollapsedControl.CMMouseenter 6875
Ribbon.TRibbonGroupCollapsedControl.CMMouseleave 6875
Ribbon.TRibbonGroupCollapsedControl.CMTextchanged 6875
Ribbon.TRibbonGroupCollapsedControl.Create 6872
Ribbon.TRibbonGroupCollapsedControl.Destroy 6873
Ribbon.TRibbonGroupCollapsedControl.GetKeyTip 6876
Ribbon.TRibbonGroupCollapsedControl.GetParentControl 6876

Ribbon.TRibbonGroupCollapsedControl.GetTopPosition 6876
Ribbon.TRibbonGroupCollapsedControl.Group 6883
Ribbon.TRibbonGroupCollapsedControl.HandleKeyTip 6876
Ribbon.TRibbonGroupCollapsedControl.HideKeyTip 6877
Ribbon.TRibbonGroupCollapsedControl.KeyTip 6883
Ribbon.TRibbonGroupCollapsedControl.KeyTipClick 6877
Ribbon.TRibbonGroupCollapsedControl.MouseInControl 6883
Ribbon.TRibbonGroupCollapsedControl.Notification 6878
Ribbon.TRibbonGroupCollapsedControl.OldWidth 6883
Ribbon.TRibbonGroupCollapsedControl.Paint 6878
Ribbon.TRibbonGroupCollapsedControl.SetKeyTip 6878
Ribbon.TRibbonGroupCollapsedControl.SetupGroupImage 6879
Ribbon.TRibbonGroupCollapsedControl.ShowKeyTip 6879
Ribbon.TRibbonGroupDockForm 6884
Ribbon.TRibbonGroupDockForm.Control 6886
Ribbon.TRibbonGroupDockForm.Create 6885
Ribbon.TRibbonGroupDockForm.CreateParams 6885
Ribbon.TRibbonGroupDockForm.ParentControl 6886
Ribbon.TRibbonGroupDockForm.WMActivate 6885
Ribbon.TRibbonGroupScrollButton 6886
Ribbon.TRibbonGroupScrollButton.CMDesignHitTest 6888
Ribbon.TRibbonGroupScrollButton.CMMouseenter 6888
Ribbon.TRibbonGroupScrollButton.CMMouseleave 6888
Ribbon.TRibbonGroupScrollButton.Create 6887
Ribbon.TRibbonGroupScrollButton.Direction 6889
Ribbon.TRibbonGroupScrollButton.MouseInControl 6889
Ribbon.TRibbonGroupScrollButton.Paint 6888
Ribbon.TRibbonGroupScrollButton.RibbonStyle 6890
Ribbon.TRibbonGroupVariant 6937
Ribbon.TRibbonGroupVariant enumeration 6937
Ribbon.TRibbonLunaColorMap 6890
Ribbon.TRibbonLunaColorMap.ActiveTabFontColor 6892
Ribbon.TRibbonLunaColorMap.BevelMinimizedInner 6892
Ribbon.TRibbonLunaColorMap.BevelMinimizedOuter 6892
Ribbon.TRibbonLunaColorMap.BtnSelectedFont 6892
Ribbon.TRibbonLunaColorMap.CaptionFontColor 6893
Ribbon.TRibbonLunaColorMap.ControlColor 6893
Ribbon.TRibbonLunaColorMap.ControlFrameColor 6893
Ribbon.TRibbonLunaColorMap.ControlInactiveColor 6893
Ribbon.TRibbonLunaColorMap.DisabledFontColor 6894
Ribbon.TRibbonLunaColorMap.DocumentFontColor 6894
Ribbon.TRibbonLunaColorMap.FontColor 6894
Ribbon.TRibbonLunaColorMap.GroupFontColor 6894
Ribbon.TRibbonLunaColorMap.HighlightColor 6895
Ribbon.TRibbonLunaColorMap.InactiveTabFontColor 6895
Ribbon.TRibbonLunaColorMap.QuickAccessToolbarMoreBorder Color
6895
Ribbon.TRibbonLunaColorMap.QuickAccessToolbarMoreColor
6895
Ribbon.TRibbonLunaColorMap.UpdateColors 6891
Ribbon.TRibbonMetric 6937
Ribbon.TRibbonMetric enumeration 6937
Ribbon.TRibbonObsidianColorMap 6896
Ribbon.TRibbonObsidianColorMap.ActiveTabFontColor 6898
Ribbon.TRibbonObsidianColorMap.BevelMinimizedInner 6898
Ribbon.TRibbonObsidianColorMap.BevelMinimizedOuter 6898
Ribbon.TRibbonObsidianColorMap.BtnSelectedFont 6898
Ribbon.TRibbonObsidianColorMap.CaptionFontColor 6898
Ribbon.TRibbonObsidianColorMap.ControlColor 6899
Ribbon.TRibbonObsidianColorMap.ControlFrameColor 6899
Ribbon.TRibbonObsidianColorMap.ControlInactiveColor 6899
Ribbon.TRibbonObsidianColorMap.DisabledFontColor 6899
Ribbon.TRibbonObsidianColorMap.DocumentFontColor 6900
Ribbon.TRibbonObsidianColorMap.FontColor 6900
Ribbon.TRibbonObsidianColorMap.GroupFontColor 6900
Ribbon.TRibbonObsidianColorMap.HighlightColor 6900
Ribbon.TRibbonObsidianColorMap.InactiveTabFontColor 6901
Ribbon.TRibbonObsidianColorMap.QuickAccessToolbarMoreBo rderColor
6901
Ribbon.TRibbonObsidianColorMap.QuickAccessToolbarMoreCo lor
6901
Ribbon.TRibbonObsidianColorMap.UpdateColors 6897
Ribbon.TRibbonPage 6901
Ribbon.TRibbonPage.BiDiMode 6903
Ribbon.TRibbonPage.Caption 6904
Ribbon.TRibbonPage.Height 6906
Ribbon.TRibbonPage.Index 6908
Ribbon.TRibbonPage.KeyTip 6908

Ribbon.TRibbonPage.Name 6908
Ribbon.TRibbonPage.ParentBiDiMode 6910
Ribbon.TRibbonQuickAccessPopupToolbar 6911
Ribbon.TRibbonQuickAccessToolbar 6911
Ribbon.TRibbonQuickAccessToolbar.ActionManager 6913
Ribbon.TRibbonQuickAccessToolbar.OnMoreCommands 6913
Ribbon.TRibbonQuickAccessToolbar.VertMargin 6914
Ribbon.TRibbonScrollButton 6938
Ribbon.TRibbonScrollButton enumeration 6938
Ribbon.TRibbonScrollButtons 6941
Ribbon.TRibbonScrollButtons type 6941
Ribbon.TRibbonSilverColorMap 6914
Ribbon.TRibbonSilverColorMap.ActiveTabFontColor 6916
Ribbon.TRibbonSilverColorMap.BevelMinimizedInner 6916
Ribbon.TRibbonSilverColorMap.BevelMinimizedOuter 6916
Ribbon.TRibbonSilverColorMap.BtnSelectedFont 6917
Ribbon.TRibbonSilverColorMap.CaptionFontColor 6917
Ribbon.TRibbonSilverColorMap.ControlColor 6917
Ribbon.TRibbonSilverColorMap.ControlFrameColor 6917
Ribbon.TRibbonSilverColorMap.ControlInactiveColor 6917
Ribbon.TRibbonSilverColorMap.DisabledFontColor 6918
Ribbon.TRibbonSilverColorMap.DocumentFontColor 6918
Ribbon.TRibbonSilverColorMap.FontColor 6918
Ribbon.TRibbonSilverColorMap.GroupFontColor 6918
Ribbon.TRibbonSilverColorMap.HighlightColor 6919
Ribbon.TRibbonSilverColorMap.InactiveTabFontColor 6919
Ribbon.TRibbonSilverColorMap.QuickAccessToolbarMoreBorderColor 6919
Ribbon.TRibbonSilverColorMap.QuickAccessToolbarMoreColor 6919
Ribbon.TRibbonSilverColorMap.UpdateColors 6915
Ribbon.TRibbonTabItem 6920
Ribbon.TRibbonTabItem.Assign 6921
Ribbon.TRibbonTabItem.BestTabWidth 6923
Ribbon.TRibbonTabItem.Caption 6923
Ribbon.TRibbonTabItem.Destroy 6920
Ribbon.TRibbonTabItem.GetDisplayName 6922
Ribbon.TRibbonTabItem.KeyTip 6923
Ribbon.TRibbonTabItem.KeyTipForm 6923
Ribbon.TRibbonTabItem.MinTabwidth 6923
Ribbon.TRibbonTabItem.MinTruncatedWidth 6924
Ribbon.TRibbonTabItem.Modified 6924
Ribbon.TRibbonTabItem.Page 6924
Ribbon.TRibbonTabItem.X 6924
Ribbon.TRibbonTabItem.Y 6925
Ribbon.TRibbonTabs 6925
Ribbon.TRibbonTabs.Add 6926
Ribbon.TRibbonTabs.Create 6926
Ribbon.TRibbonTabs.DesignerAdd 6929
Ribbon.TRibbonTabs.FindGroup 6927
Ribbon.TRibbonTabs.GetPageFromPageCaption 6927
Ribbon.TRibbonTabs.IndexOfCaption 6927
Ribbon.TRibbonTabs.Items 6929
Ribbon.TRibbonTabs.Notify 6927
Ribbon.TRibbonTabs.Ribbon 6930
Ribbon.TRibbonTabs.SelectNext 6928
Ribbon.TRibbonTabs.SelectPrior 6928
Ribbon.TRibbonTabs.TabExists 6929
Ribbon.TTabActiveScrollButton 6938
Ribbon.TTabActiveScrollButton enumeration 6938
Ribbon.TTabSeparatorState 6939
Ribbon.TTabSeparatorState enumeration 6939
RibbonActnCtrls 6943
RibbonActnCtrls namespace 6943
- Classes 6946
- Constants 7292
- Functions 7284
- Interfaces 7272
- Structs, Records, Enums 7286
- Types 7291

RibbonActnCtrls.DrawLine 7284
RibbonActnCtrls.DrawLine function 7284
RibbonActnCtrls.IRibbonBarKeyTip 7272
RibbonActnCtrls.IRibbonBarKeyTip.BarHandleKeyTips 7273
RibbonActnCtrls.IRibbonBarKeyTip.BarHideKeyTips 7273
RibbonActnCtrls.IRibbonBarKeyTip.BarShowKeyTips 7274
RibbonActnCtrls.IRibbonContainedControl 7274
RibbonActnCtrls.IRibbonContainedControl.ContainedControl 7275

RibbonActnCtrls.IRibbonContainedControl.GetContainedControl 7275
RibbonActnCtrls.IRibbonContainedControl.SetContainedControl 7275
RibbonActnCtrls.IRibbonControl 7276
RibbonActnCtrls.IRibbonControl.CanAddToQuickAccessToolbar 7276
RibbonActnCtrls.IRibbonGroupDropdown 7277
RibbonActnCtrls.IRibbonGroupDropdown.CloseUp 7277
RibbonActnCtrls.IRibbonGroupDropdown.GetParentControl 7278
RibbonActnCtrls.IRibbonGroupDropdown.GetTopPosition 7278
RibbonActnCtrls.IRibbonKeyTip 7278
RibbonActnCtrls.IRibbonKeyTip.GetKeyTip 7279
RibbonActnCtrls.IRibbonKeyTip.HandleKeyTip 7280
RibbonActnCtrls.IRibbonKeyTip.HideKeyTip 7280
RibbonActnCtrls.IRibbonKeyTip.KeyTip 7282
RibbonActnCtrls.IRibbonKeyTip.KeyTipClick 7281
RibbonActnCtrls.IRibbonKeyTip.SetKeyTip 7281
RibbonActnCtrls.IRibbonKeyTip.ShowKeyTip 7281
RibbonActnCtrls.IRibbonSplitButton 7282
RibbonActnCtrls.IRibbonSplitButton.UpdateSplitBounds 7283
RibbonActnCtrls.IsLeftMouseButtonPressed 7284
RibbonActnCtrls.IsLeftMouseButtonPressed function 7284
RibbonActnCtrls.ShowKeyTip 7284
RibbonActnCtrls.ShowKeyTip function 7284
RibbonActnCtrls.TCustomRibbonComboBox 6947
RibbonActnCtrls.TCustomRibbonComboBox.BeginAutoDrag 6949
RibbonActnCtrls.TCustomRibbonComboBox.CMMouseenter 6949
RibbonActnCtrls.TCustomRibbonComboBox.CMMouseleave 6950
RibbonActnCtrls.TCustomRibbonComboBox.Color 6955
RibbonActnCtrls.TCustomRibbonComboBox.Create 6948
RibbonActnCtrls.TCustomRibbonComboBox.Destroy 6948
RibbonActnCtrls.TCustomRibbonComboBox.DragDrop 6950
RibbonActnCtrls.TCustomRibbonComboBox.DragOver 6950
RibbonActnCtrls.TCustomRibbonComboBox.Items 6956
RibbonActnCtrls.TCustomRibbonComboBox.KeyDown 6951
RibbonActnCtrls.TCustomRibbonComboBox.KeyPress 6951
RibbonActnCtrls.TCustomRibbonComboBox.MouseInControl 6956
RibbonActnCtrls.TCustomRibbonComboBox.MouseMove 6952
RibbonActnCtrls.TCustomRibbonComboBox.Notification 6952
RibbonActnCtrls.TCustomRibbonComboBox.SetBounds 6953
RibbonActnCtrls.TCustomRibbonComboBox.SetParent 6953
RibbonActnCtrls.TCustomRibbonComboBox.WMKillFocus 6954
RibbonActnCtrls.TCustomRibbonComboBox.WMSetFocus 6954
RibbonActnCtrls.TKeyTipHandled 7286
RibbonActnCtrls.TKeyTipHandled enumeration 7286
RibbonActnCtrls.TKeyTipList 6956
RibbonActnCtrls.TKeyTipList.Add 6958
RibbonActnCtrls.TKeyTipList.Clear 6958
RibbonActnCtrls.TKeyTipList.Count 6958
RibbonActnCtrls.TKeyTipList.Create 6957
RibbonActnCtrls.TKeyTipList.Current 6959
RibbonActnCtrls.TKeyTipList.Destroy 6957
RibbonActnCtrls.TKeyTipList.Full 6959
RibbonActnCtrls.TKeyTipList.Items 6959
RibbonActnCtrls.TKeyTipList.Partial 6960
RibbonActnCtrls.TKeyTipListEnumerator 6960
RibbonActnCtrls.TKeyTipListEnumerator.Create 6961
RibbonActnCtrls.TKeyTipListEnumerator.Current 6962
RibbonActnCtrls.TKeyTipListEnumerator.GetCurrent 6961
RibbonActnCtrls.TKeyTipListEnumerator.MoveNext 6961
RibbonActnCtrls.TKeyTipMatchItem 6962
RibbonActnCtrls.TKeyTipMatchItem.Control 6963
RibbonActnCtrls.TKeyTipMatchItem.Create 6963
RibbonActnCtrls.TKeyTipMatchItem.Handled 6963
RibbonActnCtrls.TKeyTipMatchItem.Index 6964
RibbonActnCtrls.TKeyTipMatchItem.MatchType 6964
RibbonActnCtrls.TKeyTipMatchType 7287
RibbonActnCtrls.TKeyTipMatchType enumeration 7287
RibbonActnCtrls.TKeyTipPosition 7287
RibbonActnCtrls.TKeyTipPosition enumeration 7287
RibbonActnCtrls.TKeyTipState 7288
RibbonActnCtrls.TKeyTipState enumeration 7288
RibbonActnCtrls.TKeyTipValidator 6964
RibbonActnCtrls.TKeyTipValidator.Validate 6965

RibbonActnCtrls.TNumGlyphs 7291
RibbonActnCtrls.TNumGlyphs type 7291
RibbonActnCtrls.TQATScrollMode 7289
RibbonActnCtrls.TQATScrollMode enumeration 7289
RibbonActnCtrls.TRibbonAddRemoveItem 6965
RibbonActnCtrls.TRibbonAddRemoveItem.DrawBackground 6966
RibbonActnCtrls.TRibbonAddRemoveItem.DrawGlyph 6966
RibbonActnCtrls.TRibbonApplicationButton 6967
RibbonActnCtrls.TRibbonApplicationButton.CalcBounds 6967
RibbonActnCtrls.TRibbonApplicationMenuButton 6968
RibbonActnCtrls.TRibbonApplicationMenuButton.CalcBounds 6970
RibbonActnCtrls.TRibbonApplicationMenuButton.CMMouseEnter 6971
RibbonActnCtrls.TRibbonApplicationMenuButton.CMMouseLeave 6971
RibbonActnCtrls.TRibbonApplicationMenuButton.Create 6969
RibbonActnCtrls.TRibbonApplicationMenuButton.Destroy 6969
RibbonActnCtrls.TRibbonApplicationMenuButton.DragOver 6971
RibbonActnCtrls.TRibbonApplicationMenuButton.GetKeyTip 6971
RibbonActnCtrls.TRibbonApplicationMenuButton.HandleKeyTip 6972
RibbonActnCtrls.TRibbonApplicationMenuButton.HideKeyTip 6972
RibbonActnCtrls.TRibbonApplicationMenuButton.KeyTip 6976
RibbonActnCtrls.TRibbonApplicationMenuButton.KeyTipClick 6973
RibbonActnCtrls.TRibbonApplicationMenuButton.MouseInControl 6976
RibbonActnCtrls.TRibbonApplicationMenuButton.Paint 6973
RibbonActnCtrls.TRibbonApplicationMenuButton.RibbonStyle 6973
RibbonActnCtrls.TRibbonApplicationMenuButton.SetKeyTip 6974
RibbonActnCtrls.TRibbonApplicationMenuButton.SetParent 6974
RibbonActnCtrls.TRibbonApplicationMenuButton.ShowKeyTip 6975
RibbonActnCtrls.TRibbonApplicationMenuButton.UMFadeControl 6975
RibbonActnCtrls.TRibbonApplicationMenuButton.WMLButtonDblClk 6975
RibbonActnCtrls.TRibbonApplicationMenuItem 6976
RibbonActnCtrls.TRibbonApplicationMenuItem.CalcBounds 6978
RibbonActnCtrls.TRibbonApplicationMenuItem.Create 6977
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawGlyph 6978
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawSelectedBackground 6979
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawSeparator 6979
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawShadowedText 6979
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawSubMenuGlyph 6979
RibbonActnCtrls.TRibbonApplicationMenuItem.DrawText 6980
RibbonActnCtrls.TRibbonApplicationMenuItem.GetImageSize 6980
RibbonActnCtrls.TRibbonBaseButtonControl 6980
RibbonActnCtrls.TRibbonBaseButtonControl.AllowHotKeys 6983
RibbonActnCtrls.TRibbonBaseButtonControl.CalcBounds 6983
RibbonActnCtrls.TRibbonBaseButtonControl.CanAddToQuickAccessToolbar 6984
RibbonActnCtrls.TRibbonBaseButtonControl.Click 6984
RibbonActnCtrls.TRibbonBaseButtonControl.Create 6981
RibbonActnCtrls.TRibbonBaseButtonControl.Destroy 6982
RibbonActnCtrls.TRibbonBaseButtonControl.DragDrop 6984
RibbonActnCtrls.TRibbonBaseButtonControl.DrawBackground 6985
RibbonActnCtrls.TRibbonBaseButtonControl.DrawDown 6990
RibbonActnCtrls.TRibbonBaseButtonControl.DrawGlyph 6985
RibbonActnCtrls.TRibbonBaseButtonControl.DrawSeparator 6985
RibbonActnCtrls.TRibbonBaseButtonControl.DrawShadowedText 6986
RibbonActnCtrls.TRibbonBaseButtonControl.DrawText 6986
RibbonActnCtrls.TRibbonBaseButtonControl.GetImageSize 6986
RibbonActnCtrls.TRibbonBaseButtonControl.GetKeyTip 6986

RibbonActnCtrls.TRibbonBaseButtonControl.HandleKeyTip 6987
RibbonActnCtrls.TRibbonBaseButtonControl.HideKeyTip 6987
RibbonActnCtrls.TRibbonBaseButtonControl.IsButton 6988
RibbonActnCtrls.TRibbonBaseButtonControl.IsRibbonAvailable 6988
RibbonActnCtrls.TRibbonBaseButtonControl.KeyTipClick 6988
RibbonActnCtrls.TRibbonBaseButtonControl.RibbonStyle 6988
RibbonActnCtrls.TRibbonBaseButtonControl.SetKeyTip 6989
RibbonActnCtrls.TRibbonBaseButtonControl.ShowKeyTip 6989
RibbonActnCtrls.TRibbonCheckBoxControl 6990
RibbonActnCtrls.TRibbonCheckBoxControl.CalcBounds 6991
RibbonActnCtrls.TRibbonCheckBoxControl.DrawBackground 6992
RibbonActnCtrls.TRibbonCheckBoxControl.DrawGlyph 6992
RibbonActnCtrls.TRibbonComboBox 6992
RibbonActnCtrls.TRibbonComboBox.Alignment 6995
RibbonActnCtrls.TRibbonComboBox.AutoSelect 6995
RibbonActnCtrls.TRibbonComboBox.CharCase 6996
RibbonActnCtrls.TRibbonComboBox.Font 6996
RibbonActnCtrls.TRibbonComboBox.HideSelection 6997
RibbonActnCtrls.TRibbonComboBox.ImeMode 6998
RibbonActnCtrls.TRibbonComboBox.ImeName 6998
RibbonActnCtrls.TRibbonComboBox.Items 6999
RibbonActnCtrls.TRibbonComboBox.MaxLength 6999
RibbonActnCtrls.TRibbonComboBox.NumbersOnly 7000
RibbonActnCtrls.TRibbonComboBox.OnChange 7012
RibbonActnCtrls.TRibbonComboBox.OnClick 7013
RibbonActnCtrls.TRibbonComboBox.OnContextPopup 7015
RibbonActnCtrls.TRibbonComboBox.OnDblClick 7015
RibbonActnCtrls.TRibbonComboBox.OnDragDrop 7016
RibbonActnCtrls.TRibbonComboBox.OnDragOver 7019
RibbonActnCtrls.TRibbonComboBox.OnEndDock 7022
RibbonActnCtrls.TRibbonComboBox.OnEndDrag 7022
RibbonActnCtrls.TRibbonComboBox.OnEnter 7023
RibbonActnCtrls.TRibbonComboBox.OnExit 7026
RibbonActnCtrls.TRibbonComboBox.OnKeyDown 7029
RibbonActnCtrls.TRibbonComboBox.OnKeyPress 7032
RibbonActnCtrls.TRibbonComboBox.OnKeyUp 7033
RibbonActnCtrls.TRibbonComboBox.OnMouseActivate 7035
RibbonActnCtrls.TRibbonComboBox.OnMouseDown 7036
RibbonActnCtrls.TRibbonComboBox.OnMouseEnter 7038
RibbonActnCtrls.TRibbonComboBox.OnMouseLeave 7038
RibbonActnCtrls.TRibbonComboBox.OnMouseMove 7039
RibbonActnCtrls.TRibbonComboBox.OnMouseUp 7041
RibbonActnCtrls.TRibbonComboBox.OnStartDock 7043
RibbonActnCtrls.TRibbonComboBox.OnStartDrag 7044
RibbonActnCtrls.TRibbonComboBox.ParentFont 7000
RibbonActnCtrls.TRibbonComboBox.PasswordChar 7000
RibbonActnCtrls.TRibbonComboBox.PopupMenu 7001
RibbonActnCtrls.TRibbonComboBox.ReadOnly 7006
RibbonActnCtrls.TRibbonComboBox.ShowHint 7006
RibbonActnCtrls.TRibbonComboBox.TabOrder 7008
RibbonActnCtrls.TRibbonComboBox.TabStop 7008
RibbonActnCtrls.TRibbonComboBox.Text 7009
RibbonActnCtrls.TRibbonComboBox.TextHint 7010
RibbonActnCtrls.TRibbonComboBox.Visible 7010
RibbonActnCtrls.TRibbonComboBoxItem 7045
RibbonActnCtrls.TRibbonComboBoxItem.CalcBounds 7046
RibbonActnCtrls.TRibbonComboBoxItem.DrawBackground 7046
RibbonActnCtrls.TRibbonComboBoxItem.DrawText 7046
RibbonActnCtrls.TRibbonComboControl 7047
RibbonActnCtrls.TRibbonComboControl.CalcBounds 7050
RibbonActnCtrls.TRibbonComboControl.CanAddToQuickAccessToolbar 7050
RibbonActnCtrls.TRibbonComboControl.Click 7051
RibbonActnCtrls.TRibbonComboControl.CMBidimodechanged 7051
RibbonActnCtrls.TRibbonComboControl.CMMouseenter 7051
RibbonActnCtrls.TRibbonComboControl.CMMouseleave 7051
RibbonActnCtrls.TRibbonComboControl.CMVisiblechanged 7052
RibbonActnCtrls.TRibbonComboControl.ComboBox 7060
RibbonActnCtrls.TRibbonComboControl.ComboClick 7052
RibbonActnCtrls.TRibbonComboControl.ContainedControl 7060
RibbonActnCtrls.TRibbonComboControl.Create 7048
RibbonActnCtrls.TRibbonComboControl.Destroy 7049
RibbonActnCtrls.TRibbonComboControl.DrawBackground 7052
RibbonActnCtrls.TRibbonComboControl.DrawFrame 7052
RibbonActnCtrls.TRibbonComboControl.DropDownClick 7053

RibbonActnCtrls.TRibbonComboControl.GetCaptionWidth 7053
RibbonActnCtrls.TRibbonComboControl.GetContainedControl 7053
RibbonActnCtrls.TRibbonComboControl.GetGlyphWidth 7053
RibbonActnCtrls.TRibbonComboControl.GetKeyTip 7053
RibbonActnCtrls.TRibbonComboControl.GetLabelWidth 7054
RibbonActnCtrls.TRibbonComboControl.HandleKeyTip 7054
RibbonActnCtrls.TRibbonComboControl.HideKeyTip 7055
RibbonActnCtrls.TRibbonComboControl.KeyTipClick 7055
RibbonActnCtrls.TRibbonComboControl.MouseDown 7055
RibbonActnCtrls.TRibbonComboControl.MouseInControl 7060
RibbonActnCtrls.TRibbonComboControl.Notification 7056
RibbonActnCtrls.TRibbonComboControl.Paint 7056
RibbonActnCtrls.TRibbonComboControl.RibbonStyle 7057
RibbonActnCtrls.TRibbonComboControl.SetActionClient 7057
RibbonActnCtrls.TRibbonComboControl.SetBounds 7057
RibbonActnCtrls.TRibbonComboControl.SetContainedControl 7057
RibbonActnCtrls.TRibbonComboControl.SetDragMode 7058
RibbonActnCtrls.TRibbonComboControl.SetKeyTip 7058
RibbonActnCtrls.TRibbonComboControl.SetParent 7058
RibbonActnCtrls.TRibbonComboControl.ShowComboItems 7059
RibbonActnCtrls.TRibbonComboControl.ShowKeyTip 7059
RibbonActnCtrls.TRibbonCustomControl 7061
RibbonActnCtrls.TRibbonCustomControl.CalcBounds 7063
RibbonActnCtrls.TRibbonCustomControl.CanAddToQuickAccessToolbar 7064
RibbonActnCtrls.TRibbonCustomControl.Click 7064
RibbonActnCtrls.TRibbonCustomControl.CMVisiblechanged 7064
RibbonActnCtrls.TRibbonCustomControl.ContainedControl 7073
RibbonActnCtrls.TRibbonCustomControl.Create 7062
RibbonActnCtrls.TRibbonCustomControl.Destroy 7062
RibbonActnCtrls.TRibbonCustomControl.DrawBackground 7065
RibbonActnCtrls.TRibbonCustomControl.GetCaptionWidth 7065
RibbonActnCtrls.TRibbonCustomControl.GetContainedControl 7065
RibbonActnCtrls.TRibbonCustomControl.GetGlyphWidth 7065
RibbonActnCtrls.TRibbonCustomControl.GetKeyTip 7066
RibbonActnCtrls.TRibbonCustomControl.GetLabelWidth 7066
RibbonActnCtrls.TRibbonCustomControl.HandleKeyTip 7066
RibbonActnCtrls.TRibbonCustomControl.HideKeyTip 7067
RibbonActnCtrls.TRibbonCustomControl.KeyTipClick 7067
RibbonActnCtrls.TRibbonCustomControl.Notification 7067
RibbonActnCtrls.TRibbonCustomControl.Paint 7068
RibbonActnCtrls.TRibbonCustomControl.SetActionClient 7068
RibbonActnCtrls.TRibbonCustomControl.SetBounds 7068
RibbonActnCtrls.TRibbonCustomControl.SetContainedControl 7071
RibbonActnCtrls.TRibbonCustomControl.SetDragMode 7071
RibbonActnCtrls.TRibbonCustomControl.SetKeyTip 7071
RibbonActnCtrls.TRibbonCustomControl.SetParent 7072
RibbonActnCtrls.TRibbonCustomControl.ShowKeyTip 7072
RibbonActnCtrls.TRibbonCustomizeItem 7073
RibbonActnCtrls.TRibbonCustomizeItem.CalcBounds 7074
RibbonActnCtrls.TRibbonCustomizeItem.DrawBackground 7075
RibbonActnCtrls.TRibbonCustomizeItem.DrawGlyph 7075
RibbonActnCtrls.TRibbonCustomizeItem.RibbonStyle 7075
RibbonActnCtrls.TRibbonCustomStatusControl 7076
RibbonActnCtrls.TRibbonCustomStatusControl.CalcBounds 7077
RibbonActnCtrls.TRibbonCustomStatusControl.DrawBackground 7078
RibbonActnCtrls.TRibbonDropDownButton 7078
RibbonActnCtrls.TRibbonDropDownButton.CalcBounds 7080
RibbonActnCtrls.TRibbonDropDownButton.CalcLayout 7080
RibbonActnCtrls.TRibbonDropDownButton.Click 7080
RibbonActnCtrls.TRibbonDropDownButton.DoDropDown 7081
RibbonActnCtrls.TRibbonDropDownButton.DrawArrow 7081
RibbonActnCtrls.TRibbonDropDownButton.DrawBackground 7081
RibbonActnCtrls.TRibbonDropDownButton.DrawShadowedText 7081
RibbonActnCtrls.TRibbonDropDownButton.DrawText 7082
RibbonActnCtrls.TRibbonDropDownButton.GetArrowPosition 7082
RibbonActnCtrls.TRibbonDropDownButton.GetPopupClass 7082
RibbonActnCtrls.TRibbonDropDownButton.KeyTipClick 7082
RibbonActnCtrls.TRibbonGallery 7083
RibbonActnCtrls.TRibbonGalleryControl 7083

RibbonActnCtrls.TRibbonGalleryControl.CanAddToQuickAccessToolbar 7084
RibbonActnCtrls.TRibbonGalleryControl.ContainedControl 7085
RibbonActnCtrls.TRibbonGalleryControl.Gallery 7085
RibbonActnCtrls.TRibbonGalleryControl.GetContainedControl 7084
RibbonActnCtrls.TRibbonGalleryControl.SetContainedControl 7084
RibbonActnCtrls.TRibbonGalleryControlMenuItem 7085
RibbonActnCtrls.TRibbonGalleryControlMenuItem.CalcBounds 7086
RibbonActnCtrls.TRibbonGalleryControlMenuItem.CanAddToQuickAccessToolbar 7087
RibbonActnCtrls.TRibbonGalleryControlMenuItem.ContainedControl 7091
RibbonActnCtrls.TRibbonGalleryControlMenuItem.GetContainedControl 7087
RibbonActnCtrls.TRibbonGalleryControlMenuItem.Paint 7087
RibbonActnCtrls.TRibbonGalleryControlMenuItem.SetBounds 7088
RibbonActnCtrls.TRibbonGalleryControlMenuItem.SetContainedControl 7090
RibbonActnCtrls.TRibbonGalleryControlMenuItem.SetParent 7090
RibbonActnCtrls.TRibbonGalleryItem 7091
RibbonActnCtrls.TRibbonGalleryItem.CalcBounds 7093
RibbonActnCtrls.TRibbonGalleryItem.CalcLayout 7093
RibbonActnCtrls.TRibbonGalleryMenuItem 7093
RibbonActnCtrls.TRibbonGalleryMenuItem.CalcBounds 7094
RibbonActnCtrls.TRibbonGalleryMenuItem.DrawBackground 7095
RibbonActnCtrls.TRibbonGalleryMenuItem.DrawGlyph 7095
RibbonActnCtrls.TRibbonGalleryMenuItem.DrawShadowedText 7095
RibbonActnCtrls.TRibbonGalleryMenuItem.DrawText 7096
RibbonActnCtrls.TRibbonGallerySplitButton 7096
RibbonActnCtrls.TRibbonGallerySplitButton.GetPopupClass 7098
RibbonActnCtrls.TRibbonGripItem 7098
RibbonActnCtrls.TRibbonGripItem.CalcBounds 7100
RibbonActnCtrls.TRibbonGripItem.Create 7099
RibbonActnCtrls.TRibbonGripItem.GalleryResize 7101
RibbonActnCtrls.TRibbonGripItem.Paint 7100
RibbonActnCtrls.TRibbonGripItem.RibbonStyle 7101
RibbonActnCtrls.TRibbonGroupDropDown 7101
RibbonActnCtrls.TRibbonGroupDropDown.CloseUp 7103
RibbonActnCtrls.TRibbonGroupDropDown.DoDropDown 7103
RibbonActnCtrls.TRibbonGroupDropDown.DrawArrow 7104
RibbonActnCtrls.TRibbonGroupDropDown.DrawBackground 7104
RibbonActnCtrls.TRibbonGroupDropDown.GetParentControl 7104
RibbonActnCtrls.TRibbonGroupDropDown.GetTopPosition 7104
RibbonActnCtrls.TRibbonGroupDropDown.Group 7105
RibbonActnCtrls.TRibbonLargeButtonControl 7105
RibbonActnCtrls.TRibbonLargeButtonControl.CalcBounds 7107
RibbonActnCtrls.TRibbonLargeButtonControl.CalcLayout 7107
RibbonActnCtrls.TRibbonLargeButtonControl.DrawBackground 7107
RibbonActnCtrls.TRibbonLargeButtonControl.DrawShadowedText 7108
RibbonActnCtrls.TRibbonLargeButtonControl.DrawText 7108
RibbonActnCtrls.TRibbonLargeSplitButton 7108
RibbonActnCtrls.TRibbonLargeSplitButton.CalcBounds 7110
RibbonActnCtrls.TRibbonLargeSplitButton.CalcLayout 7111
RibbonActnCtrls.TRibbonLargeSplitButton.Click 7111
RibbonActnCtrls.TRibbonLargeSplitButton.CMMouseleave 7111
RibbonActnCtrls.TRibbonLargeSplitButton.CursorInSplit 7115
RibbonActnCtrls.TRibbonLargeSplitButton.DoSplitButtonClick 7111
RibbonActnCtrls.TRibbonLargeSplitButton.DrawArrow 7112
RibbonActnCtrls.TRibbonLargeSplitButton.DrawBackground 7112
RibbonActnCtrls.TRibbonLargeSplitButton.GetArrowPosition 7112
RibbonActnCtrls.TRibbonLargeSplitButton.InSplit 7112
RibbonActnCtrls.TRibbonLargeSplitButton.KeyTipClick 7113
RibbonActnCtrls.TRibbonLargeSplitButton.MouseDown 7113
RibbonActnCtrls.TRibbonLargeSplitButton.MouseMove 7113
RibbonActnCtrls.TRibbonLargeSplitButton.MouseUp 7114
RibbonActnCtrls.TRibbonLargeSplitButton.UpdateSplitBounds 7114

RibbonActnCtrls.TRibbonMainMenuItem 7115
RibbonActnCtrls.TRibbonMainMenuItem.DrawShadowedText 7116
RibbonActnCtrls.TRibbonMainMenuItem.DrawText 7116
RibbonActnCtrls.TRibbonMainMenuSeparator 7117
RibbonActnCtrls.TRibbonMainMenuSeparator.CalcBounds 7118
RibbonActnCtrls.TRibbonMenuItem 7118
RibbonActnCtrls.TRibbonMenuItem.CalcBounds 7120
RibbonActnCtrls.TRibbonMenuItem.CalcLayout 7120
RibbonActnCtrls.TRibbonMenuItem.Click 7120
RibbonActnCtrls.TRibbonMenuItem.Create 7119
RibbonActnCtrls.TRibbonMenuItem.DrawBackground 7121
RibbonActnCtrls.TRibbonMenuItem.DrawBanner 7123
RibbonActnCtrls.TRibbonMenuItem.DrawGlyph 7121
RibbonActnCtrls.TRibbonMenuItem.DrawSelectedBackground 7121
RibbonActnCtrls.TRibbonMenuItem.DrawSeparator 7122
RibbonActnCtrls.TRibbonMenuItem.DrawShadowedText 7122
RibbonActnCtrls.TRibbonMenuItem.DrawText 7122
RibbonActnCtrls.TRibbonMenuItem.GetShowShortCut 7122
RibbonActnCtrls.TRibbonMenuItem.RibbonStyle 7123
RibbonActnCtrls.TRibbonMenuSeparatorItem 7123
RibbonActnCtrls.TRibbonMenuSeparatorItem.CalcBounds 7124
RibbonActnCtrls.TRibbonMenuSeparatorItem.DrawSeparator 7125
RibbonActnCtrls.TRibbonMenuSmallButtonControl 7125
RibbonActnCtrls.TRibbonMenuSmallButtonControl.Create 7126
RibbonActnCtrls.TRibbonMenuSmallButtonControl.MouseUp 7127
RibbonActnCtrls.TRibbonOptionButtonControl 7128
RibbonActnCtrls.TRibbonOptionButtonControl.DrawBackground 7129
RibbonActnCtrls.TRibbonQATScrollButton 7130
RibbonActnCtrls.TRibbonQATScrollButton.CalcBounds 7131
RibbonActnCtrls.TRibbonQATScrollButton.Click 7131
RibbonActnCtrls.TRibbonQATScrollButton.CMMouseEnter 7132
RibbonActnCtrls.TRibbonQATScrollButton.CMMouseLeave 7132
RibbonActnCtrls.TRibbonQATScrollButton.CursorInBtn 7134
RibbonActnCtrls.TRibbonQATScrollButton.DrawArrows 7132
RibbonActnCtrls.TRibbonQATScrollButton.DrawBackground 7132
RibbonActnCtrls.TRibbonQATScrollButton.DrawDown 7134
RibbonActnCtrls.TRibbonQATScrollButton.MouseMove 7133
RibbonActnCtrls.TRibbonQATScrollButton.RibbonStyle 7133
RibbonActnCtrls.TRibbonQATScrollButton.ScrollMode 7134
RibbonActnCtrls.TRibbonRadioButtonControl 7134
RibbonActnCtrls.TRibbonRadioButtonControl.CalcBounds 7136
RibbonActnCtrls.TRibbonRadioButtonControl.DrawBackground 7136
RibbonActnCtrls.TRibbonRadioButtonControl.DrawGlyph 7136
RibbonActnCtrls.TRibbonRecentButtonControl 7137
RibbonActnCtrls.TRibbonRecentButtonControl.AllowHotKeys 7138
RibbonActnCtrls.TRibbonRichContentItem 7139
RibbonActnCtrls.TRibbonRichContentItem.CalcBounds 7140
RibbonActnCtrls.TRibbonRichContentItem.CanCalculateHeight 7141
RibbonActnCtrls.TRibbonRichContentItem.Create 7140
RibbonActnCtrls.TRibbonRichContentItem.DrawBackground 7141
RibbonActnCtrls.TRibbonRichContentItem.DrawGlyph 7141
RibbonActnCtrls.TRibbonRichContentItem.DrawShadowedText 7142
RibbonActnCtrls.TRibbonRichContentItem.DrawText 7142
RibbonActnCtrls.TRibbonRichContentItem.GetItemHeight 7142
RibbonActnCtrls.TRibbonRichContentItem.GetItemWidth 7143
RibbonActnCtrls.TRibbonRichContentItem.GetMenuProps 7143
RibbonActnCtrls.TRibbonRichContentMainMenuItem 7143
RibbonActnCtrls.TRibbonRichContentMainMenuItem.CalcBounds 7145
RibbonActnCtrls.TRibbonRichContentMainMenuItem.CalcLayout 7145
RibbonActnCtrls.TRibbonRichContentMainMenuItem.Create 7144
RibbonActnCtrls.TRibbonRichContentMainMenuItem.DrawGlyph 7145
RibbonActnCtrls.TRibbonRichContentMainMenuItem.DrawText 7146
RibbonActnCtrls.TRibbonRichContentMainMenuItem.GetImageSize 7146
RibbonActnCtrls.TRibbonRichContentMainMenuItem.GetShowShortCut

7146
RibbonActnCtrls.TRibbonSeparator 7147
RibbonActnCtrls.TRibbonSeparator.CalcBounds 7148
RibbonActnCtrls.TRibbonSeparator.Create 7147
RibbonActnCtrls.TRibbonSeparator.DrawBackground 7148
RibbonActnCtrls.TRibbonSeparator.DrawGlyph 7149
RibbonActnCtrls.TRibbonSeparator.DrawShadowedText 7149
RibbonActnCtrls.TRibbonSeparator.DrawText 7149
RibbonActnCtrls.TRibbonSeparator.RibbonStyle 7150
RibbonActnCtrls.TRibbonSmallButtonControl 7150
RibbonActnCtrls.TRibbonSmallButtonControl.CalcBounds 7151
RibbonActnCtrls.TRibbonSmallSplitButton 7152
RibbonActnCtrls.TRibbonSmallSplitButton.CalcBounds 7155
RibbonActnCtrls.TRibbonSmallSplitButton.CalcLayout 7155
RibbonActnCtrls.TRibbonSmallSplitButton.Click 7156
RibbonActnCtrls.TRibbonSmallSplitButton.CMMouseLeave 7156
RibbonActnCtrls.TRibbonSmallSplitButton.Create 7153
RibbonActnCtrls.TRibbonSmallSplitButton.CursorInSplit 7159
RibbonActnCtrls.TRibbonSmallSplitButton.DrawBackground 7156
RibbonActnCtrls.TRibbonSmallSplitButton.FCursorInSplit 7154
RibbonActnCtrls.TRibbonSmallSplitButton.FSplitRect 7154
RibbonActnCtrls.TRibbonSmallSplitButton.GetArrowPosition 7157
RibbonActnCtrls.TRibbonSmallSplitButton.InSplit 7157
RibbonActnCtrls.TRibbonSmallSplitButton.KeyTipClick 7157
RibbonActnCtrls.TRibbonSmallSplitButton.MouseDown 7157
RibbonActnCtrls.TRibbonSmallSplitButton.MouseMove 7158
RibbonActnCtrls.TRibbonSmallSplitButton.UpdateSplitBounds 7158
RibbonActnCtrls.TRibbonSpinButton 7159
RibbonActnCtrls.TRibbonSpinButton.Align 7164
RibbonActnCtrls.TRibbonSpinButton.Anchors 7167
RibbonActnCtrls.TRibbonSpinButton.Constraints 7168
RibbonActnCtrls.TRibbonSpinButton.Control 7161
RibbonActnCtrls.TRibbonSpinButton.Create 7160
RibbonActnCtrls.TRibbonSpinButton.Ctl3D 7169
RibbonActnCtrls.TRibbonSpinButton.DownGlyph 7169
RibbonActnCtrls.TRibbonSpinButton.DownNumGlyphs 7169
RibbonActnCtrls.TRibbonSpinButton.DragCursor 7170
RibbonActnCtrls.TRibbonSpinButton.DragKind 7170
RibbonActnCtrls.TRibbonSpinButton.DragMode 7170
RibbonActnCtrls.TRibbonSpinButton.Enabled 7172
RibbonActnCtrls.TRibbonSpinButton.FocusControl 7172
RibbonActnCtrls.TRibbonSpinButton.KeyDown 7161
RibbonActnCtrls.TRibbonSpinButton.Loaded 7162
RibbonActnCtrls.TRibbonSpinButton.Notification 7162
RibbonActnCtrls.TRibbonSpinButton.OnDownClick 7183
RibbonActnCtrls.TRibbonSpinButton.OnDragDrop 7183
RibbonActnCtrls.TRibbonSpinButton.OnDragOver 7186
RibbonActnCtrls.TRibbonSpinButton.OnEndDock 7190
RibbonActnCtrls.TRibbonSpinButton.OnEndDrag 7190
RibbonActnCtrls.TRibbonSpinButton.OnEnter 7191
RibbonActnCtrls.TRibbonSpinButton.OnExit 7194
RibbonActnCtrls.TRibbonSpinButton.OnStartDock 7196
RibbonActnCtrls.TRibbonSpinButton.OnStartDrag 7197
RibbonActnCtrls.TRibbonSpinButton.OnUpClick 7198
RibbonActnCtrls.TRibbonSpinButton.ParentCtl3D 7173
RibbonActnCtrls.TRibbonSpinButton.ParentShowHint 7173
RibbonActnCtrls.TRibbonSpinButton.PopupMenu 7173
RibbonActnCtrls.TRibbonSpinButton.SetBounds 7163
RibbonActnCtrls.TRibbonSpinButton.ShowHint 7178
RibbonActnCtrls.TRibbonSpinButton.TabOrder 7180
RibbonActnCtrls.TRibbonSpinButton.TabStop 7181
RibbonActnCtrls.TRibbonSpinButton.UpGlyph 7181
RibbonActnCtrls.TRibbonSpinButton.UpNumGlyphs 7181
RibbonActnCtrls.TRibbonSpinButton.Visible 7181
RibbonActnCtrls.TRibbonSpinEdit 7198
RibbonActnCtrls.TRibbonSpinEdit.Anchors 7207
RibbonActnCtrls.TRibbonSpinEdit.AutoSelect 7207
RibbonActnCtrls.TRibbonSpinEdit.AutoSize 7208
RibbonActnCtrls.TRibbonSpinEdit.Button 7209
RibbonActnCtrls.TRibbonSpinEdit.CMMouseenter 7201
RibbonActnCtrls.TRibbonSpinEdit.CMMouseleave 7201
RibbonActnCtrls.TRibbonSpinEdit.Color 7209
RibbonActnCtrls.TRibbonSpinEdit.Constraints 7210
RibbonActnCtrls.TRibbonSpinEdit.Create 7200
RibbonActnCtrls.TRibbonSpinEdit.CreateParams 7201
RibbonActnCtrls.TRibbonSpinEdit.CreateWnd 7202

RibbonActnCtrls.TRibbonSpinEdit.Ctl3D 7210
RibbonActnCtrls.TRibbonSpinEdit.Destroy 7200
RibbonActnCtrls.TRibbonSpinEdit.DownClick 7202
RibbonActnCtrls.TRibbonSpinEdit.DragCursor 7211
RibbonActnCtrls.TRibbonSpinEdit.DragMode 7211
RibbonActnCtrls.TRibbonSpinEdit.EditorEnabled 7213
RibbonActnCtrls.TRibbonSpinEdit.Enabled 7213
RibbonActnCtrls.TRibbonSpinEdit.Font 7214
RibbonActnCtrls.TRibbonSpinEdit.GetChildren 7203
RibbonActnCtrls.TRibbonSpinEdit.Increment 7215
RibbonActnCtrls.TRibbonSpinEdit.IsValidChar 7203
RibbonActnCtrls.TRibbonSpinEdit.KeyDown 7203
RibbonActnCtrls.TRibbonSpinEdit.KeyPress 7204
RibbonActnCtrls.TRibbonSpinEdit.MaxLength 7215
RibbonActnCtrls.TRibbonSpinEdit.MaxValue 7215
RibbonActnCtrls.TRibbonSpinEdit.MinValue 7216
RibbonActnCtrls.TRibbonSpinEdit.OnChange 7228
RibbonActnCtrls.TRibbonSpinEdit.OnClick 7228
RibbonActnCtrls.TRibbonSpinEdit.OnDblClick 7230
RibbonActnCtrls.TRibbonSpinEdit.OnDragDrop 7230
RibbonActnCtrls.TRibbonSpinEdit.OnDragOver 7233
RibbonActnCtrls.TRibbonSpinEdit.OnEndDrag 7237
RibbonActnCtrls.TRibbonSpinEdit.OnEnter 7238
RibbonActnCtrls.TRibbonSpinEdit.OnExit 7240
RibbonActnCtrls.TRibbonSpinEdit.OnKeyDown 7243
RibbonActnCtrls.TRibbonSpinEdit.OnKeyPress 7246
RibbonActnCtrls.TRibbonSpinEdit.OnKeyUp 7248
RibbonActnCtrls.TRibbonSpinEdit.OnMouseDown 7249
RibbonActnCtrls.TRibbonSpinEdit.OnMouseMove 7252
RibbonActnCtrls.TRibbonSpinEdit.OnMouseUp 7254
RibbonActnCtrls.TRibbonSpinEdit.OnStartDrag 7256
RibbonActnCtrls.TRibbonSpinEdit.ParentColor 7216
RibbonActnCtrls.TRibbonSpinEdit.ParentCtl3D 7216
RibbonActnCtrls.TRibbonSpinEdit.ParentFont 7217
RibbonActnCtrls.TRibbonSpinEdit.ParentShowHint 7217
RibbonActnCtrls.TRibbonSpinEdit.PopupMenu 7218
RibbonActnCtrls.TRibbonSpinEdit.ReadOnly 7223
RibbonActnCtrls.TRibbonSpinEdit.SetEnabled 7204
RibbonActnCtrls.TRibbonSpinEdit.SetParent 7205
RibbonActnCtrls.TRibbonSpinEdit.ShowHint 7223
RibbonActnCtrls.TRibbonSpinEdit.TabOrder 7224
RibbonActnCtrls.TRibbonSpinEdit.TabStop 7225
RibbonActnCtrls.TRibbonSpinEdit.UpClick 7206
RibbonActnCtrls.TRibbonSpinEdit.Value 7226
RibbonActnCtrls.TRibbonSpinEdit.Visible 7226
RibbonActnCtrls.TRibbonSpinEdit.WMKillFocus 7206
RibbonActnCtrls.TRibbonSplitMenuItem 7257
RibbonActnCtrls.TRibbonSplitMenuItem.CalcBounds 7259
RibbonActnCtrls.TRibbonSplitMenuItem.Click 7260
RibbonActnCtrls.TRibbonSplitMenuItem.Create 7258
RibbonActnCtrls.TRibbonSplitMenuItem.DrawBackground 7260
RibbonActnCtrls.TRibbonSplitMenuItem.DrawSelectedBackground 7260
RibbonActnCtrls.TRibbonSplitMenuItem.FMouseInSplit 7259
RibbonActnCtrls.TRibbonSplitMenuItem.FSplitRect 7259
RibbonActnCtrls.TRibbonSplitMenuItem.IsInSplit 7261
RibbonActnCtrls.TRibbonSplitMenuItem.MouseInSplit 7262
RibbonActnCtrls.TRibbonSplitMenuItem.MouseMove 7261
RibbonActnCtrls.TRibbonSplitMenuItem.UpdateSplitBounds 7261
RibbonActnCtrls.TRibbonStyleMenuButton 7262
RibbonActnCtrls.TRibbonStyleMenuButton.DrawBackground 7263
RibbonActnCtrls.TRibbonStyleMenuButton.RibbonStyle 7263
RibbonActnCtrls.TRibbonTextControl 7263
RibbonActnCtrls.TRibbonTextControl.CalcBounds 7264
RibbonActnCtrls.TRibbonTextControl.DrawBackground 7265
RibbonActnCtrls.TRibbonTextControl.DrawText 7265
RibbonActnCtrls.TRibbonToolScrollButton 7265
RibbonActnCtrls.TRibbonToolScrollButton.DrawArrows 7266
RibbonActnCtrls.TRibbonToolScrollButton.DrawBackground 7266
RibbonActnCtrls.TSpinButtonStyle 7290
RibbonActnCtrls.TSpinButtonStyle enumeration 7290
RibbonActnCtrls.TTimeBtnState 7291
RibbonActnCtrls.TTimeBtnState type 7291
RibbonActnCtrls.TTimerSpeedButton 7267
RibbonActnCtrls.TTimerSpeedButton.ButtonStyle 7270
RibbonActnCtrls.TTimerSpeedButton.CMMouseenter 7268

RibbonActnCtrls.TTimerSpeedButton.CMMouseleave 7269
RibbonActnCtrls.TTimerSpeedButton.Create 7267
RibbonActnCtrls.TTimerSpeedButton.Destroy 7268
RibbonActnCtrls.TTimerSpeedButton.MouseDown 7269
RibbonActnCtrls.TTimerSpeedButton.MouseInControl 7271
RibbonActnCtrls.TTimerSpeedButton.MouseUp 7269
RibbonActnCtrls.TTimerSpeedButton.Paint 7270
RibbonActnCtrls.TTimerSpeedButton.TimeBtnState 7271
RibbonActnCtrls.UM_FADECONTROL 7292
RibbonActnCtrls.UM_FADECONTROL constant 7292
RibbonActnMenus 7293
RibbonActnMenus namespace 7293
Classes 7294
RibbonActnMenus.TOptionItem 7294
RibbonActnMenus.TOptionItem.Action 7296
RibbonActnMenus.TOptionItem.Assign 7295
RibbonActnMenus.TOptionItem.Caption 7297
RibbonActnMenus.TOptionItem.GetDisplayName 7296
RibbonActnMenus.TOptionItem.Hint 7297
RibbonActnMenus.TOptionItem.KeyTip 7297
RibbonActnMenus.TOptionItem.Tag 7297
RibbonActnMenus.TOptionItems 7298
RibbonActnMenus.TOptionItems.Add 7298
RibbonActnMenus.TOptionItems.ApplicationMenu 7299
RibbonActnMenus.TOptionItems.Insert 7299
RibbonActnMenus.TOptionItems.Items 7300
RibbonActnMenus.TRibbonActionPopupMenu 7300
RibbonActnMenus.TRibbonActionPopupMenu.CreateControl 7301
RibbonActnMenus.TRibbonActionPopupMenu.NCPaint 7301
RibbonActnMenus.TRibbonActionPopupMenu.SelectFirstItem 7302
RibbonActnMenus.TRibbonActionPopupMenu.TrackMenu 7302
RibbonActnMenus.TRibbonApplicationMenuBar 7302
RibbonActnMenus.TRibbonApplicationMenuBar.ActionManager 7310
RibbonActnMenus.TRibbonApplicationMenuBar.Align 7310
RibbonActnMenus.TRibbonApplicationMenuBar.CloseMenu 7305
RibbonActnMenus.TRibbonApplicationMenuBar.Create 7304
RibbonActnMenus.TRibbonApplicationMenuBar.CreateControl 7305
RibbonActnMenus.TRibbonApplicationMenuBar.CreateControls 7306
RibbonActnMenus.TRibbonApplicationMenuBar.Destroy 7304
RibbonActnMenus.TRibbonApplicationMenuBar.DragOver 7306
RibbonActnMenus.TRibbonApplicationMenuBar.DrawBackground 7306
RibbonActnMenus.TRibbonApplicationMenuBar.GetControlClass 7306
RibbonActnMenus.TRibbonApplicationMenuBar.GetRibbon 7307
RibbonActnMenus.TRibbonApplicationMenuBar.Height 7314
RibbonActnMenus.TRibbonApplicationMenuBar.HorzMargin 7315
RibbonActnMenus.TRibbonApplicationMenuBar.Left 7315
RibbonActnMenus.TRibbonApplicationMenuBar.NewPopup 7307
RibbonActnMenus.TRibbonApplicationMenuBar.Notification 7307
RibbonActnMenus.TRibbonApplicationMenuBar.OptionItems 7316
RibbonActnMenus.TRibbonApplicationMenuBar.Paint 7308
RibbonActnMenus.TRibbonApplicationMenuBar.RecentItems 7316
RibbonActnMenus.TRibbonApplicationMenuBar.RibbonStyle 7308
RibbonActnMenus.TRibbonApplicationMenuBar.SelectApplicationButton 7308
RibbonActnMenus.TRibbonApplicationMenuBar.SetBounds 7308
RibbonActnMenus.TRibbonApplicationMenuBar.SetRibbon 7309
RibbonActnMenus.TRibbonApplicationMenuBar.ShowKeyTips 7309
RibbonActnMenus.TRibbonApplicationMenuBar.Spacing 7316
RibbonActnMenus.TRibbonApplicationMenuBar.Top 7317
RibbonActnMenus.TRibbonApplicationMenuBar.VertMargin 7318
RibbonActnMenus.TRibbonApplicationMenuBar.VisibleMenu 7319
RibbonActnMenus.TRibbonApplicationMenuBar.VisibleMenuCount 7319
RibbonActnMenus.TRibbonApplicationMenuBar.Width 7319

RibbonActnMenus.TRibbonApplicationMenuBar.WMSysCommand 7309
RibbonActnMenus.TRibbonApplicationPopupMenu 7321
RibbonActnMenus.TRibbonApplicationPopupMenu.AlignControls 7323
RibbonActnMenus.TRibbonApplicationPopupMenu.CMTextchanged 7323
RibbonActnMenus.TRibbonApplicationPopupMenu.Create 7322
RibbonActnMenus.TRibbonApplicationPopupMenu.CreateControls 7323
RibbonActnMenus.TRibbonApplicationPopupMenu.CreateOptionsItems 7324
RibbonActnMenus.TRibbonApplicationPopupMenu.CreateRecentDocuments 7324
RibbonActnMenus.TRibbonApplicationPopupMenu.CreateWnd 7324
RibbonActnMenus.TRibbonApplicationPopupMenu.CustomAlignPosition 7325
RibbonActnMenus.TRibbonApplicationPopupMenu.Destroy 7322
RibbonActnMenus.TRibbonApplicationPopupMenu.DoDropActions 7325
RibbonActnMenus.TRibbonApplicationPopupMenu.DoDropCategory 7326
RibbonActnMenus.TRibbonApplicationPopupMenu.DoDropItem 7326
RibbonActnMenus.TRibbonApplicationPopupMenu.DrawBackground 7326
RibbonActnMenus.TRibbonApplicationPopupMenu.GetBorderHeight 7326
RibbonActnMenus.TRibbonApplicationPopupMenu.GetRibbon 7326
RibbonActnMenus.TRibbonApplicationPopupMenu.NCPaint 7327
RibbonActnMenus.TRibbonApplicationPopupMenu.PositionPopup 7327
RibbonActnMenus.TRibbonApplicationPopupMenu.RibbonStyle 7327
RibbonActnMenus.TRibbonApplicationPopupMenu.SetColorMap 7327
RibbonActnMenus.TRibbonApplicationPopupMenu.SetRibbon 7328
RibbonActnMenus.TRibbonQATCustomizeMenu 7328
RibbonActnMenus.TRibbonQATCustomizeMenu.DisplayShadow 7329
RibbonActnMenus.TRibbonQATCustomizeMenu.DoAddCustomizeItems 7329
RibbonActnMenus.TRibbonQATCustomizeMenu.GetControlClass 7329
RibbonActnMenus.TRibbonQATCustomizeMenu.HideShadow 7329
RibbonActnMenus.TRibbonQATCustomizeMenu.NCPaint 7330
RibbonActnMenus.TRibbonQATCustomizeMenu.PositionPopup 7330
RibbonActnMenus.TRibbonQATCustomizeMenu.VisibleChanging 7330
RibbonActnMenus.TRibbonRecentDocuments 7330
RibbonActnMenus.TRibbonRecentDocuments.Caption 7332
RibbonActnMenus.TRibbonRecentDocuments.Create 7331
RibbonActnMenus.TRibbonRecentDocuments.Paint 7332
RibbonActnMenus.TRibbonStyleMainPopupMenu 7335
RibbonActnMenus.TRibbonStyleMainPopupMenu.Create 7335
RibbonActnMenus.TRibbonStyleMainPopupMenu.NCPaint 7336
RibbonActnMenus.TRibbonStyleMainPopupMenu.PositionPopup 7336
RibbonActnMenus.TRibbonStylePopupMenu 7336
RibbonActnMenus.TRibbonStylePopupMenu.AlignControls 7338
RibbonActnMenus.TRibbonStylePopupMenu.CanAutoSize 7339
RibbonActnMenus.TRibbonStylePopupMenu.Create 7337
RibbonActnMenus.TRibbonStylePopupMenu.CreateControl 7339
RibbonActnMenus.TRibbonStylePopupMenu.Destroy 7338
RibbonActnMenus.TRibbonStylePopupMenu.GalleryResize 7340
RibbonActnMenus.TRibbonStylePopupMenu.WMExitsizemove 7339
RibbonActnMenus.TRibbonStylePopupMenu.WMNCHitTest 7340

RibbonConsts 7341
RibbonConsts namespace 7341
Constants 7342
RibbonConsts.SAddToQuickAccessToolbar 7342
RibbonConsts.SAddToQuickAccessToolbar constant 7342
RibbonConsts.SCommandIsDisabled 7343
RibbonConsts.SCommandIsDisabled constant 7343
RibbonConsts.SCustomizeQuickAccessToolbar 7343
RibbonConsts.SCustomizeQuickAccessToolbar constant 7343
RibbonConsts.SDefaultFooterText 7343
RibbonConsts.SDefaultFooterText constant 7343
RibbonConsts.SDuplicateRibbonComponentsNotAllowed 7344
RibbonConsts.SDuplicateRibbonComponentsNotAllowed constant 7344
RibbonConsts.SGenerateScreenTipsError 7344
RibbonConsts.SGenerateScreenTipsError constant 7344
RibbonConsts.SInvalidRibbonGroupParent 7344
RibbonConsts.SInvalidRibbonGroupParent constant 7344
RibbonConsts.SInvalidRibbonPageChild 7345
RibbonConsts.SInvalidRibbonPageChild constant 7345
RibbonConsts.SInvalidRibbonPageParent 7345
RibbonConsts.SInvalidRibbonPageParent constant 7345
RibbonConsts.SInvalidScreenTipItemAssign 7345
RibbonConsts.SInvalidScreenTipItemAssign constant 7345
RibbonConsts.SMinimizeTheRibbon 7346
RibbonConsts.SMinimizeTheRibbon constant 7346
RibbonConsts.SMoreCommands 7346
RibbonConsts.SMoreCommands constant 7346
RibbonConsts.SMoreControls 7346
RibbonConsts.SMoreControls constant 7346
RibbonConsts.SPageCaptionExists 7347
RibbonConsts.SPageCaptionExists constant 7347
RibbonConsts.SQATShowAboveTheRibbon 7347
RibbonConsts.SQATShowAboveTheRibbon constant 7347
RibbonConsts.SQATShowBelowTheRibbon 7347
RibbonConsts.SQATShowBelowTheRibbon constant 7347
RibbonConsts.SQuickAccessToolbarGlyphRequired 7348
RibbonConsts.SQuickAccessToolbarGlyphRequired constant 7348
RibbonConsts.SRecentDocuments 7348
RibbonConsts.SRecentDocuments constant 7348
RibbonConsts.SRemoveFromQuickAccessToolbar 7348
RibbonConsts.SRemoveFromQuickAccessToolbar constant 7348
RibbonConsts.SRibbonControlInvalidParent 7349
RibbonConsts.SRibbonControlInvalidParent constant 7349
RibbonConsts.SRibbonTabIndexInvalid 7349
RibbonConsts.SRibbonTabIndexInvalid constant 7349
RibbonConsts.SShowQuickAccessToolbarAbove 7349
RibbonConsts.SShowQuickAccessToolbarAbove constant 7349
RibbonConsts.SShowQuickAccessToolbarBelow 7350
RibbonConsts.SShowQuickAccessToolbarBelow constant 7350
RibbonConsts.SUseTheGlyphProperty 7350
RibbonConsts.SUseTheGlyphProperty constant 7350
RibbonGalleryBar 7351
RibbonGalleryBar namespace 7351
Classes 7352
RibbonGalleryBar.TCustomRibbonGalleryBar 7352
RibbonGalleryBar.TCustomRibbonGalleryBar.Align 7354
RibbonGalleryBar.TCustomRibbonGalleryBar.AlignControls 7353
RibbonGalleryBar.TCustomRibbonGalleryBar.Create 7353
RibbonGalleryBar.TCustomRibbonGalleryBar.CreateControl 7354
RibbonGalleryBar.TCustomRibbonGalleryBar.DoDropCategory 7354
RibbonGalleryBar.TCustomRibbonGalleryBar.ItemsPerRow 7358
RibbonGalleryBar.TCustomRibbonGalleryBar.ShowItemCaptions 7358
RibbonGalleryBar.TCustomRibbonGalleryPopupBar 7358
RibbonGalleryBar.TCustomRibbonGalleryPopupBar.AlignControls 7359
RibbonGalleryBar.TCustomRibbonGalleryPopupBar.CreateControls 7360
RibbonGalleryBar.TCustomRibbonGalleryPopupBar.DoGetControlClass 7360
RibbonGalleryBar.TCustomRibbonGalleryPopupBar.SetColorMap 7360
RibbonGalleryBar.TRibbonGalleryBar 7360

RibbonGalleryBar.TRibbonGalleryBar.Items 7361
RibbonGalleryBar.TRibbonGalleryBar.ItemsPerRow 7362
RibbonGalleryBar.TRibbonGalleryBar.ShowItemCaptions 7362
RibbonGalleryBar.TRibbonGalleryPopupBar 7362
RibbonGalleryBar.TRibbonPopupGalleryBar 7362
RibbonGalleryBar.TRibbonPopupGalleryBar.Assign 7364
RibbonGalleryBar.TRibbonPopupGalleryBar.CreateParams 7364
RibbonGalleryBar.TRibbonPopupGalleryBar.DisplayGallery 7365
RibbonStyleActnCtrls 7367
RibbonStyleActnCtrls namespace 7367
Classes 7369
Structs, Records, Enums 7383
Types 7406
RibbonStyleActnCtrls.TAlphaBitmap 7369
RibbonStyleActnCtrls.TAlphaBitmap.Create 7371
RibbonStyleActnCtrls.TCustomRibbonSkin 7372
RibbonStyleActnCtrls.TCustomRibbonSkin.Bitmap 7376
RibbonStyleActnCtrls.TCustomRibbonSkin.Create 7373
RibbonStyleActnCtrls.TCustomRibbonSkin.Destroy 7373
RibbonStyleActnCtrls.TCustomRibbonSkin.DoLoadByFile 7374
RibbonStyleActnCtrls.TCustomRibbonSkin.DoLoadByResource 7374
RibbonStyleActnCtrls.TCustomRibbonSkin.GetLoadType 7375
RibbonStyleActnCtrls.TCustomRibbonSkin.GetLoadTypeValue 7375
RibbonStyleActnCtrls.TCustomRibbonSkin.GetResourceHandle 7375
RibbonStyleActnCtrls.TCustomRibbonSkin.LoadSkin 7375
RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType 7373
RibbonStyleActnCtrls.TCustomRibbonSkin.TRibbonSkinLoadType enumeration 7373
RibbonStyleActnCtrls.TCustomRibbonSkinClass 7406
RibbonStyleActnCtrls.TCustomRibbonSkinClass type 7406
RibbonStyleActnCtrls.TRibbonElement 7383
RibbonStyleActnCtrls.TRibbonElement enumeration 7383
RibbonStyleActnCtrls.TRibbonStyleActionBars 7376
RibbonStyleActnCtrls.TRibbonStyleActionBars.Create 7377
RibbonStyleActnCtrls.TRibbonStyleActionBars.Destroy 7377
RibbonStyleActnCtrls.TRibbonStyleActionBars.DrawElement 7379
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetAddRemoveItemClass 7380
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetColorMapClass 7380
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetControlClass 7380
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetPopupClass 7380
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetScrollBtnClass 7381
RibbonStyleActnCtrls.TRibbonStyleActionBars.GetSkinClass 7381
RibbonStyleActnCtrls.TRibbonStyleActionBars.RibbonSkin 7381
RibbonStyleActnCtrls.TSkinApplicationMenu 7384
RibbonStyleActnCtrls.TSkinApplicationMenu enumeration 7384
RibbonStyleActnCtrls.TSkinButton 7386
RibbonStyleActnCtrls.TSkinButton enumeration 7386
RibbonStyleActnCtrls.TSkinCheckBox 7390
RibbonStyleActnCtrls.TSkinCheckBox enumeration 7390
RibbonStyleActnCtrls.TSkinComboBox 7391
RibbonStyleActnCtrls.TSkinComboBox enumeration 7391
RibbonStyleActnCtrls.TSkinForm 7391
RibbonStyleActnCtrls.TSkinForm enumeration 7391
RibbonStyleActnCtrls.TSkinGallery 7394
RibbonStyleActnCtrls.TSkinGallery enumeration 7394
RibbonStyleActnCtrls.TSkinMenu 7395
RibbonStyleActnCtrls.TSkinMenu enumeration 7395
RibbonStyleActnCtrls.TSkinQuickAccessToolbar 7397
RibbonStyleActnCtrls.TSkinQuickAccessToolbar enumeration 7397
RibbonStyleActnCtrls.TSkinRadioButton 7398
RibbonStyleActnCtrls.TSkinRadioButton enumeration 7398
RibbonStyleActnCtrls.TSkinRibbon 7399
RibbonStyleActnCtrls.TSkinRibbon enumeration 7399
RibbonStyleActnCtrls.TSkinRibbonGroup 7400
RibbonStyleActnCtrls.TSkinRibbonGroup enumeration 7400
RibbonStyleActnCtrls.TSkinScrollBar 7402

RibbonStyleActnCtrls.TSkinScrollBar enumeration 7402
RibbonStyleActnCtrls.TSkinSpinEdit 7403
RibbonStyleActnCtrls.TSkinSpinEdit enumeration 7403
RibbonStyleActnCtrls.TSkinStatusBar 7403
RibbonStyleActnCtrls.TSkinStatusBar enumeration 7403
RibbonStyleActnCtrls.TSkinTab 7404
RibbonStyleActnCtrls.TSkinTab enumeration 7404
Rio 7407
Rio namespace 7407
Classes 7408
Constants 7423
Interfaces 7420
Types 7422
Rio.IRIOAccess 7420
Rio.IRIOAccess.GetRIO 7421
Rio.IRIOAccess.RIO 7421
Rio.StubSize 7423
Rio.StubSize constant 7423
Rio.TAfterExecuteEvent 7422
Rio.TAfterExecuteEvent type 7422
Rio.TBeforeExecuteEvent 7422
Rio.TBeforeExecuteEvent type 7422
Rio.TRIO 7408
Rio.TRIO.AfterConstruction 7412
Rio.TRIO.BeforeDestruction 7412
Rio.TRIO.Converter 7415
Rio.TRIO.Create 7410
Rio.TRIO.Destroy 7410
Rio.TRIO.DoAfterExecute 7413
Rio.TRIO.DoBeforeExecute 7413
Rio.TRIO.FConverter 7411
Rio.TRIO.FIID 7411
Rio.TRIO.FWebNode 7411
Rio.TRIO.GetResponseStream 7414
Rio.TRIO.IntfMD 7411
Rio.TRIO.NewInstance 7414
Rio.TRIO.OnAfterExecute 7417
Rio.TRIO.OnBeforeExecute 7418
Rio.TRIO.OnGetAttachment 7419
Rio.TRIO.OnSendAttachment 7419
Rio.TRIO.QueryInterface 7415
Rio.TRIO.RefCount 7416
Rio.TRIO.SOAPHeaders 7416
Rio.TRIO.WebNode 7417
rmCaption enumeration member 6937
rmDown enumeration member 4771
rmFontSize enumeration member 6937
rmFullHeight enumeration member 6937
rmGroupCaptionHeight enumeration member 6937
rmGroupHeight enumeration member 6937
rmHeight enumeration member 6937
rmMinHeight enumeration member 6937
rmNearest enumeration member 4771
rmPageHeight enumeration member 6937
rmQATWidth enumeration member 6937
rmSystemButtonHeight enumeration member 6937
rmSystemButtonWidth enumeration member 6937
rmTabHeight enumeration member 6937
rmTabTop enumeration member 6937
rmTruncate enumeration member 4771
rmUp enumeration member 4771
RollbackAction enumeration member 4023
RollbackAdvise enumeration member 4024
RollbackGlobal enumeration member 4024
RollbackState enumeration member 4025
rrAbort enumeration member 6587
rrApply enumeration member 6587
rrIgnore enumeration member 6587
rrMerge enumeration member 6587
rrSkip enumeration member 6587
rsLine enumeration member 1544
rsNone enumeration member 1544
rsPattern enumeration member 1544
rsUpdate enumeration member 1544
rtBitmap enumeration member 4230
rtCursor enumeration member 4230
rtIcon enumeration member 4230
RTLConsts 7425

RTLConsts namespace 7425
Constants 7431
RTLConsts.hNoContext 7436
RTLConsts.hNoContext constant 7436
RTLConsts.hNoContextFound 7436
RTLConsts.hNoContextFound constant 7436
RTLConsts.hNoFilterViewer 7436
RTLConsts.hNoFilterViewer constant 7436
RTLConsts.hNoSystem 7437
RTLConsts.hNoSystem constant 7437
RTLConsts.hNoTableOfContents 7437
RTLConsts.hNoTableOfContents constant 7437
RTLConsts.hNothingFound 7437
RTLConsts.hNothingFound constant 7437
RTLConsts.hNoTopics 7438
RTLConsts.hNoTopics constant 7438
RTLConsts.SAcreFeetDescription 7438
RTLConsts.SAcreFeetDescription constant 7438
RTLConsts.SAcreInchesDescription 7438
RTLConsts.SAcreInchesDescription constant 7438
RTLConsts.SAcresDescription 7439
RTLConsts.SAcresDescription constant 7439
RTLConsts.SAncestorNotFound 7439
RTLConsts.SAncestorNotFound constant 7439
RTLConsts.SAngstromsDescription 7439
RTLConsts.SAngstromsDescription constant 7439
RTLConsts.SAnsiUTF8Expected 7440
RTLConsts.SAnsiUTF8Expected constant 7440
RTLConsts.SAreaDescription 7440
RTLConsts.SAreaDescription constant 7440
RTLConsts.SAresDescription 7440
RTLConsts.SAresDescription constant 7440
RTLConsts.sArgument_InvalidHighSurrogate 7441
RTLConsts.sArgument_InvalidHighSurrogate constant 7441
RTLConsts.sArgument_InvalidLowSurrogate 7441
RTLConsts.sArgument_InvalidLowSurrogate constant 7441
RTLConsts.sArgumentOutOfRange_Index 7441
RTLConsts.sArgumentOutOfRange_Index constant 7441
RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate 7442
RTLConsts.sArgumentOutOfRange_InvalidHighSurrogate constant 7442
RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate 7442
RTLConsts.sArgumentOutOfRange_InvalidLowSurrogate constant 7442
RTLConsts.sArgumentOutOfRange_InvalidUTF32 7443
RTLConsts.sArgumentOutOfRange_InvalidUTF32 constant 7443
RTLConsts.sArgumentOutOfRange_StringIndex 7443
RTLConsts.sArgumentOutOfRange_StringIndex constant 7443
RTLConsts.SAssignError 7443
RTLConsts.SAssignError constant 7443
RTLConsts.SAstronomicalUnitsDescription 7444
RTLConsts.SAstronomicalUnitsDescription constant 7444
RTLConsts.sAsyncSocketError 7444
RTLConsts.sAsyncSocketError constant 7444
RTLConsts.SBitsIndexError 7444
RTLConsts.SBitsIndexError constant 7444
RTLConsts.SBucketListLocked 7445
RTLConsts.SBucketListLocked constant 7445
RTLConsts.sCannotCallAcquireOnConditionVar 7445
RTLConsts.sCannotCallAcquireOnConditionVar constant 7445
RTLConsts.sCannotCreateSocket 7445
RTLConsts.sCannotCreateSocket constant 7445
RTLConsts.sCannotListenOnOpen 7446
RTLConsts.sCannotListenOnOpen constant 7446
RTLConsts.sCantChangeWhileActive 7446
RTLConsts.sCantChangeWhileActive constant 7446
RTLConsts.SCantWriteResourceStreamError 7446
RTLConsts.SCantWriteResourceStreamError constant 7446
RTLConsts.SCelsiusDescription 7447
RTLConsts.SCelsiusDescription constant 7447
RTLConsts.SCentaresDescription 7447
RTLConsts.SCentaresDescription constant 7447
RTLConsts.SCentigramsDescription 7447
RTLConsts.SCentigramsDescription constant 7447
RTLConsts.SCentiLitersDescription 7448
RTLConsts.SCentiLitersDescription constant 7448
RTLConsts.SCentimetersDescription 7448
RTLConsts.SCentimetersDescription constant 7448

RTLConsts.SCenturiesDescription 7448
RTLConsts.SCenturiesDescription constant 7448
RTLConsts.SChainsDescription 7449
RTLConsts.SChainsDescription constant 7449
RTLConsts.SCharExpected 7449
RTLConsts.SCharExpected constant 7449
RTLConsts.SCheckSynchronizeError 7449
RTLConsts.SCheckSynchronizeError constant 7449
RTLConsts.SClassNotFound 7450
RTLConsts.SClassNotFound constant 7450
RTLConsts.SCmplxCouldNotParseImaginary 7450
RTLConsts.SCmplxCouldNotParseImaginary constant 7450
RTLConsts.SCmplxCouldNotParsePlus 7450
RTLConsts.SCmplxCouldNotParsePlus constant 7450
RTLConsts.SCmplxCouldNotParseReal 7451
RTLConsts.SCmplxCouldNotParseReal constant 7451
RTLConsts.SCmplxCouldNotParseSymbol 7451
RTLConsts.SCmplxCouldNotParseSymbol constant 7451
RTLConsts.SCmplxErrorSuffix 7451
RTLConsts.SCmplxErrorSuffix constant 7451
RTLConsts.SCmplxUnexpectedChars 7452
RTLConsts.SCmplxUnexpectedChars constant 7452
RTLConsts.SCmplxUnexpectedEOS 7452
RTLConsts.SCmplxUnexpectedEOS constant 7452
RTLConsts.SComponentNameTooLong 7452
RTLConsts.SComponentNameTooLong constant 7452
RTLConsts.SConvDuplicateFamily 7453
RTLConsts.SConvDuplicateFamily constant 7453
RTLConsts.SConvDuplicateType 7453
RTLConsts.SConvDuplicateType constant 7453
RTLConsts.SConvFactorZero 7453
RTLConsts.SConvFactorZero constant 7453
RTLConsts.SConvIllegalFamily 7454
RTLConsts.SConvIllegalFamily constant 7454
RTLConsts.SConvIllegalType 7454
RTLConsts.SConvIllegalType constant 7454
RTLConsts.SConvIncompatibleTypes2 7454
RTLConsts.SConvIncompatibleTypes2 constant 7454
RTLConsts.SConvIncompatibleTypes3 7455
RTLConsts.SConvIncompatibleTypes3 constant 7455
RTLConsts.SConvIncompatibleTypes4 7455
RTLConsts.SConvIncompatibleTypes4 constant 7455
RTLConsts.SConvStrParseError 7455
RTLConsts.SConvStrParseError constant 7455
RTLConsts.SConvUnknownDescription 7456
RTLConsts.SConvUnknownDescription constant 7456
RTLConsts.SConvUnknownDescriptionWithPrefix 7456
RTLConsts.SConvUnknownDescriptionWithPrefix constant 7456
RTLConsts.SConvUnknownFamily 7456
RTLConsts.SConvUnknownFamily constant 7456
RTLConsts.SConvUnknownType 7457
RTLConsts.SConvUnknownType constant 7457
RTLConsts.SCordFeetDescription 7457
RTLConsts.SCordFeetDescription constant 7457
RTLConsts.SCordsDescription 7457
RTLConsts.SCordsDescription constant 7457
RTLConsts.SCubicCentimetersDescription 7458
RTLConsts.SCubicCentimetersDescription constant 7458
RTLConsts.SCubicDecametersDescription 7458
RTLConsts.SCubicDecametersDescription constant 7458
RTLConsts.SCubicDecimetersDescription 7458
RTLConsts.SCubicDecimetersDescription constant 7458
RTLConsts.SCubicFeetDescription 7459
RTLConsts.SCubicFeetDescription constant 7459
RTLConsts.SCubicHectometersDescription 7459
RTLConsts.SCubicHectometersDescription constant 7459
RTLConsts.SCubicInchesDescription 7459
RTLConsts.SCubicInchesDescription constant 7459
RTLConsts.SCubicKilometersDescription 7460
RTLConsts.SCubicKilometersDescription constant 7460
RTLConsts.SCubicMetersDescription 7460
RTLConsts.SCubicMetersDescription constant 7460
RTLConsts.SCubicMilesDescription 7460
RTLConsts.SCubicMilesDescription constant 7460
RTLConsts.SCubicMillimetersDescription 7461
RTLConsts.SCubicMillimetersDescription constant 7461
RTLConsts.SCubicYardsDescription 7461
RTLConsts.SCubicYardsDescription constant 7461

RTLConsts.SCubitsDescription 7461
RTLConsts.SCubitsDescription constant 7461
RTLConsts.SDateTimeDescription 7462
RTLConsts.SDateTimeDescription constant 7462
RTLConsts.SDaysDescription 7462
RTLConsts.SDaysDescription constant 7462
RTLConsts.SDecadesDescription 7462
RTLConsts.SDecadesDescription constant 7462
RTLConsts.SDecagramsDescription 7463
RTLConsts.SDecagramsDescription constant 7463
RTLConsts.SDecaLitersDescription 7463
RTLConsts.SDecaLitersDescription constant 7463
RTLConsts.SDecametersDescription 7463
RTLConsts.SDecametersDescription constant 7463
RTLConsts.SDecasteresDescription 7464
RTLConsts.SDecasteresDescription constant 7464
RTLConsts.SDecigramsDescription 7464
RTLConsts.SDecigramsDescription constant 7464
RTLConsts.SDeciLitersDescription 7464
RTLConsts.SDeciLitersDescription constant 7464
RTLConsts.SDecimetersDescription 7465
RTLConsts.SDecimetersDescription constant 7465
RTLConsts.SDecisteresDescription 7465
RTLConsts.SDecisteresDescription constant 7465
RTLConsts.SDelimiterQuoteCharError 7465
RTLConsts.SDelimiterQuoteCharError constant 7465
RTLConsts.SDistanceDescription 7466
RTLConsts.SDistanceDescription constant 7466
RTLConsts.SDramsDescription 7466
RTLConsts.SDramsDescription constant 7466
RTLConsts.SDryBucketsDescription 7466
RTLConsts.SDryBucketsDescription constant 7466
RTLConsts.SDryBushelsDescription 7467
RTLConsts.SDryBushelsDescription constant 7467
RTLConsts.SDryGallonsDescription 7467
RTLConsts.SDryGallonsDescription constant 7467
RTLConsts.SDryPecksDescription 7467
RTLConsts.SDryPecksDescription constant 7467
RTLConsts.SDryPintsDescription 7468
RTLConsts.SDryPintsDescription constant 7468
RTLConsts.SDryQuartsDescription 7468
RTLConsts.SDryQuartsDescription constant 7468
RTLConsts.SDuplicateClass 7468
RTLConsts.SDuplicateClass constant 7468
RTLConsts.SDuplicateItem 7469
RTLConsts.SDuplicateItem constant 7469
RTLConsts.SDuplicateName 7469
RTLConsts.SDuplicateName constant 7469
RTLConsts.SDuplicateString 7469
RTLConsts.SDuplicateString constant 7469
RTLConsts.SFahrenheitDescription 7470
RTLConsts.SFahrenheitDescription constant 7470
RTLConsts.SFailedToCallConstructor 7470
RTLConsts.SFailedToCallConstructor constant 7470
RTLConsts.SFathomsDescription 7470
RTLConsts.SFathomsDescription constant 7470
RTLConsts.SFCreateError 7471
RTLConsts.SFCreateError constant 7471
RTLConsts.SFCreateErrorEx 7471
RTLConsts.SFCreateErrorEx constant 7471
RTLConsts.SFeetDescription 7471
RTLConsts.SFeetDescription constant 7471
RTLConsts.SFixedColTooBig 7472
RTLConsts.SFixedColTooBig constant 7472
RTLConsts.SFixedRowTooBig 7472
RTLConsts.SFixedRowTooBig constant 7472
RTLConsts.SFluidCupsDescription 7472
RTLConsts.SFluidCupsDescription constant 7472
RTLConsts.SFluidGallonsDescription 7473
RTLConsts.SFluidGallonsDescription constant 7473
RTLConsts.SFluidGillsDescription 7473
RTLConsts.SFluidGillsDescription constant 7473
RTLConsts.SFluidOuncesDescription 7473
RTLConsts.SFluidOuncesDescription constant 7473
RTLConsts.SFluidPintsDescription 7474
RTLConsts.SFluidPintsDescription constant 7474
RTLConsts.SFluidQuartsDescription 7474
RTLConsts.SFluidQuartsDescription constant 7474

RTLConsts.SFluidTablespoonsDescription 7474
RTLConsts.SFluidTablespoonsDescription constant 7474
RTLConsts.SFluidTeaspoonsDescription 7475
RTLConsts.SFluidTeaspoonsDescription constant 7475
RTLConsts.SFOpenError 7475
RTLConsts.SFOpenError constant 7475
RTLConsts.SFOpenErrorEx 7475
RTLConsts.SFOpenErrorEx constant 7475
RTLConsts.SFortnightsDescription 7476
RTLConsts.SFortnightsDescription constant 7476
RTLConsts.SFurlongsDescription 7476
RTLConsts.SFurlongsDescription constant 7476
RTLConsts.SGigametersDescription 7476
RTLConsts.SGigametersDescription constant 7476
RTLConsts.SGrainsDescription 7477
RTLConsts.SGrainsDescription constant 7477
RTLConsts.SGramsDescription 7477
RTLConsts.SGramsDescription constant 7477
RTLConsts.SGridTooLarge 7477
RTLConsts.SGridTooLarge constant 7477
RTLConsts.SHandsDescription 7478
RTLConsts.SHandsDescription constant 7478
RTLConsts.SHectaresDescription 7478
RTLConsts.SHectaresDescription constant 7478
RTLConsts.SHectogramsDescription 7478
RTLConsts.SHectogramsDescription constant 7478
RTLConsts.SHectoLitersDescription 7479
RTLConsts.SHectoLitersDescription constant 7479
RTLConsts.SHectometersDescription 7479
RTLConsts.SHectometersDescription constant 7479
RTLConsts.SHoursDescription 7479
RTLConsts.SHoursDescription constant 7479
RTLConsts.SIdentifierExpected 7480
RTLConsts.SIdentifierExpected constant 7480
RTLConsts.SInchesDescription 7480
RTLConsts.SInchesDescription constant 7480
RTLConsts.SIndexOutOfRange 7480
RTLConsts.SIndexOutOfRange constant 7480
RTLConsts.SIniFileWriteError 7481
RTLConsts.SIniFileWriteError constant 7481
RTLConsts.SInvalidActionCreation 7481
RTLConsts.SInvalidActionCreation constant 7481
RTLConsts.SInvalidActionEnumeration 7481
RTLConsts.SInvalidActionEnumeration constant 7481
RTLConsts.SInvalidActionRegistration 7482
RTLConsts.SInvalidActionRegistration constant 7482
RTLConsts.SInvalidActionUnregistration 7482
RTLConsts.SInvalidActionUnregistration constant 7482
RTLConsts.SInvalidBinary 7482
RTLConsts.SInvalidBinary constant 7482
RTLConsts.SInvalidDate 7483
RTLConsts.SInvalidDate constant 7483
RTLConsts.SInvalidDateDay 7483
RTLConsts.SInvalidDateDay constant 7483
RTLConsts.SInvalidDateMonthWeek 7483
RTLConsts.SInvalidDateMonthWeek constant 7483
RTLConsts.SInvalidDateTime 7484
RTLConsts.SInvalidDateTime constant 7484
RTLConsts.SInvalidDateWeek 7484
RTLConsts.SInvalidDateWeek constant 7484
RTLConsts.SInvalidDayOfWeekInMonth 7484
RTLConsts.SInvalidDayOfWeekInMonth constant 7484
RTLConsts.SInvalidFileName 7485
RTLConsts.SInvalidFileName constant 7485
RTLConsts.SInvalidImage 7485
RTLConsts.SInvalidImage constant 7485
RTLConsts.SInvalidInteger 7485
RTLConsts.SInvalidInteger constant 7485
RTLConsts.SInvalidJulianDate 7486
RTLConsts.SInvalidJulianDate constant 7486
RTLConsts.SInvalidMask 7486
RTLConsts.SInvalidMask constant 7486
RTLConsts.SInvalidName 7486
RTLConsts.SInvalidName constant 7486
RTLConsts.SInvalidProperty 7487
RTLConsts.SInvalidProperty constant 7487
RTLConsts.SInvalidPropertyElement 7487
RTLConsts.SInvalidPropertyElement constant 7487

RTLConsts.SInvalidPropertyPath 7487
RTLConsts.SInvalidPropertyPath constant 7487
RTLConsts.SInvalidPropertyType 7488
RTLConsts.SInvalidPropertyType constant 7488
RTLConsts.SInvalidPropertyValue 7488
RTLConsts.SInvalidPropertyValue constant 7488
RTLConsts.SInvalidRegType 7488
RTLConsts.SInvalidRegType constant 7488
RTLConsts.SInvalidString 7489
RTLConsts.SInvalidString constant 7489
RTLConsts.SInvalidStringGridOp 7489
RTLConsts.SInvalidStringGridOp constant 7489
RTLConsts.SInvalidTime 7489
RTLConsts.SInvalidTime constant 7489
RTLConsts.SItemNotFound 7490
RTLConsts.SItemNotFound constant 7490
RTLConsts.SJulianDateDescription 7490
RTLConsts.SJulianDateDescription constant 7490
RTLConsts.SKelvinDescription 7490
RTLConsts.SKelvinDescription constant 7490
RTLConsts.SKilogramsDescription 7491
RTLConsts.SKilogramsDescription constant 7491
RTLConsts.SKiloLitersDescription 7491
RTLConsts.SKiloLitersDescription constant 7491
RTLConsts.SKilometersDescription 7491
RTLConsts.SKilometersDescription constant 7491
RTLConsts.SLightYearsDescription 7492
RTLConsts.SLightYearsDescription constant 7492
RTLConsts.SLineTooLong 7492
RTLConsts.SLineTooLong constant 7492
RTLConsts.SLinksDescription 7492
RTLConsts.SLinksDescription constant 7492
RTLConsts.SListCapacityError 7493
RTLConsts.SListCapacityError constant 7493
RTLConsts.SListCountError 7493
RTLConsts.SListCountError constant 7493
RTLConsts.SListIndexError 7493
RTLConsts.SListIndexError constant 7493
RTLConsts.SLitersDescription 7494
RTLConsts.SLitersDescription constant 7494
RTLConsts.SLongTonsDescription 7494
RTLConsts.SLongTonsDescription constant 7494
RTLConsts.SMaskEditErr 7494
RTLConsts.SMaskEditErr constant 7494
RTLConsts.SMaskErr 7495
RTLConsts.SMaskErr constant 7495
RTLConsts.SMassDescription 7495
RTLConsts.SMassDescription constant 7495
RTLConsts.SMegametersDescription 7495
RTLConsts.SMegametersDescription constant 7495
RTLConsts.SMemoryStreamError 7496
RTLConsts.SMemoryStreamError constant 7496
RTLConsts.SMetersDescription 7496
RTLConsts.SMetersDescription constant 7496
RTLConsts.SMetricTonsDescription 7496
RTLConsts.SMetricTonsDescription constant 7496
RTLConsts.SMicrogramsDescription 7497
RTLConsts.SMicrogramsDescription constant 7497
RTLConsts.SMicromicronsDescription 7497
RTLConsts.SMicromicronsDescription constant 7497
RTLConsts.SMicronsDescription 7497
RTLConsts.SMicronsDescription constant 7497
RTLConsts.SMilesDescription 7498
RTLConsts.SMilesDescription constant 7498
RTLConsts.SMillenniaDescription 7498
RTLConsts.SMillenniaDescription constant 7498
RTLConsts.SMilligramsDescription 7498
RTLConsts.SMilligramsDescription constant 7498
RTLConsts.SMilliLitersDescription 7499
RTLConsts.SMilliLitersDescription constant 7499
RTLConsts.SMillimetersDescription 7499
RTLConsts.SMillimetersDescription constant 7499
RTLConsts.SMillimicronsDescription 7499
RTLConsts.SMillimicronsDescription constant 7499
RTLConsts.SMilliSecondsDescription 7500
RTLConsts.SMilliSecondsDescription constant 7500
RTLConsts.SMinutesDescription 7500
RTLConsts.SMinutesDescription constant 7500

RTLConsts.SMissingDateTimeField 7500
RTLConsts.SMissingDateTimeField constant 7500
RTLConsts.SModifiedJulianDateDescription 7501
RTLConsts.SModifiedJulianDateDescription constant 7501
RTLConsts.SMonthsDescription 7501
RTLConsts.SMonthsDescription constant 7501
RTLConsts.SNanogramsDescription 7501
RTLConsts.SNanogramsDescription constant 7501
RTLConsts.SNauticalMilesDescription 7502
RTLConsts.SNauticalMilesDescription constant 7502
RTLConsts.sNoAddress 7502
RTLConsts.sNoAddress constant 7502
RTLConsts.SNoComSupport 7502
RTLConsts.SNoComSupport constant 7502
RTLConsts.sNoConstruct 7503
RTLConsts.sNoConstruct constant 7503
RTLConsts.SNotPrinting 7503
RTLConsts.SNotPrinting constant 7503
RTLConsts.SNumberExpected 7503
RTLConsts.SNumberExpected constant 7503
RTLConsts.SOuncesDescription 7504
RTLConsts.SOuncesDescription constant 7504
RTLConsts.SPacesDescription 7504
RTLConsts.SPacesDescription constant 7504
RTLConsts.SParsecsDescription 7504
RTLConsts.SParsecsDescription constant 7504
RTLConsts.SParseError 7505
RTLConsts.SParseError constant 7505
RTLConsts.SPicasDescription 7505
RTLConsts.SPicasDescription constant 7505
RTLConsts.SPointsDescription 7505
RTLConsts.SPointsDescription constant 7505
RTLConsts.SPoundsDescription 7506
RTLConsts.SPoundsDescription constant 7506
RTLConsts.SPrinting 7506
RTLConsts.SPrinting constant 7506
RTLConsts.SPropertyException 7506
RTLConsts.SPropertyException constant 7506
RTLConsts.SRankineDescription 7507
RTLConsts.SRankineDescription constant 7507
RTLConsts.SReadError 7507
RTLConsts.SReadError constant 7507
RTLConsts.SReadOnlyProperty 7507
RTLConsts.SReadOnlyProperty constant 7507
RTLConsts.SReaumurDescription 7508
RTLConsts.SReaumurDescription constant 7508
RTLConsts.SRegCreateFailed 7508
RTLConsts.SRegCreateFailed constant 7508
RTLConsts.SRegGetDataFailed 7508
RTLConsts.SRegGetDataFailed constant 7508
RTLConsts.SRegisterError 7509
RTLConsts.SRegisterError constant 7509
RTLConsts.SRegSetDataFailed 7509
RTLConsts.SRegSetDataFailed constant 7509
RTLConsts.SResNotFound 7509
RTLConsts.SResNotFound constant 7509
RTLConsts.SRodsDescription 7510
RTLConsts.SRodsDescription constant 7510
RTLConsts.SSecondsDescription 7510
RTLConsts.SSecondsDescription constant 7510
RTLConsts.SSeekNotImplemented 7510
RTLConsts.SSeekNotImplemented constant 7510
RTLConsts.sSocketAlreadyOpen 7511
RTLConsts.sSocketAlreadyOpen constant 7511
RTLConsts.sSocketIOError 7511
RTLConsts.sSocketIOError constant 7511
RTLConsts.sSocketMustBeBlocking 7511
RTLConsts.sSocketMustBeBlocking constant 7511
RTLConsts.sSocketRead 7512
RTLConsts.sSocketRead constant 7512
RTLConsts.sSocketWrite 7512
RTLConsts.sSocketWrite constant 7512
RTLConsts.SSortedListError 7512
RTLConsts.SSortedListError constant 7512
RTLConsts.SSquareCentimetersDescription 7513
RTLConsts.SSquareCentimetersDescription constant 7513
RTLConsts.SSquareDecametersDescription 7513
RTLConsts.SSquareDecametersDescription constant 7513

RTLConsts.SSquareDecimetersDescription 7513
RTLConsts.SSquareDecimetersDescription constant 7513
RTLConsts.SSquareFeetDescription 7514
RTLConsts.SSquareFeetDescription constant 7514
RTLConsts.SSquareHectometersDescription 7514
RTLConsts.SSquareHectometersDescription constant 7514
RTLConsts.SSquareInchesDescription 7514
RTLConsts.SSquareInchesDescription constant 7514
RTLConsts.SSquareKilometersDescription 7515
RTLConsts.SSquareKilometersDescription constant 7515
RTLConsts.SSquareMetersDescription 7515
RTLConsts.SSquareMetersDescription constant 7515
RTLConsts.SSquareMilesDescription 7515
RTLConsts.SSquareMilesDescription constant 7515
RTLConsts.SSquareMillimetersDescription 7516
RTLConsts.SSquareMillimetersDescription constant 7516
RTLConsts.SSquareRodsDescription 7516
RTLConsts.SSquareRodsDescription constant 7516
RTLConsts.SSquareYardsDescription 7516
RTLConsts.SSquareYardsDescription constant 7516
RTLConsts.SSteresDescription 7517
RTLConsts.SSteresDescription constant 7517
RTLConsts.SStonesDescription 7517
RTLConsts.SStonesDescription constant 7517
RTLConsts.SStreamSetSize 7517
RTLConsts.SStreamSetSize constant 7517
RTLConsts.SStringExpected 7518
RTLConsts.SStringExpected constant 7518
RTLConsts.SSymbolExpected 7518
RTLConsts.SSymbolExpected constant 7518
RTLConsts.STemperatureDescription 7518
RTLConsts.STemperatureDescription constant 7518
RTLConsts.SThreadCreateError 7519
RTLConsts.SThreadCreateError constant 7519
RTLConsts.SThreadError 7519
RTLConsts.SThreadError constant 7519
RTLConsts.SThreadExternalTerminate 7519
RTLConsts.SThreadExternalTerminate constant 7519
RTLConsts.SThreadExternalWait 7520
RTLConsts.SThreadExternalWait constant 7520
RTLConsts.STimeDescription 7520
RTLConsts.STimeDescription constant 7520
RTLConsts.STimeEncodeError 7520
RTLConsts.STimeEncodeError constant 7520
RTLConsts.STonsDescription 7521
RTLConsts.STonsDescription constant 7521
RTLConsts.STooManyDeleted 7521
RTLConsts.STooManyDeleted constant 7521
RTLConsts.SUKBucketsDescription 7521
RTLConsts.SUKBucketsDescription constant 7521
RTLConsts.SUKBushelsDescription 7522
RTLConsts.SUKBushelsDescription constant 7522
RTLConsts.SUKGallonsDescription 7522
RTLConsts.SUKGallonsDescription constant 7522
RTLConsts.SUKGillsDescription 7522
RTLConsts.SUKGillsDescription constant 7522
RTLConsts.SUKOuncesDescription 7523
RTLConsts.SUKOuncesDescription constant 7523
RTLConsts.SUKPecksDescription 7523
RTLConsts.SUKPecksDescription constant 7523
RTLConsts.SUKPintsDescription 7523
RTLConsts.SUKPintsDescription constant 7523
RTLConsts.SUKPottlesDescription 7524
RTLConsts.SUKPottlesDescription constant 7524
RTLConsts.SUKQuartsDescription 7524
RTLConsts.SUKQuartsDescription constant 7524
RTLConsts.SUnknownGroup 7524
RTLConsts.SUnknownGroup constant 7524
RTLConsts.SUnknownProperty 7525
RTLConsts.SUnknownProperty constant 7525
RTLConsts.SVolumeDescription 7525
RTLConsts.SVolumeDescription constant 7525
RTLConsts.SWeeksDescription 7525
RTLConsts.SWeeksDescription constant 7525
RTLConsts.sWindowsSocketError 7526
RTLConsts.sWindowsSocketError constant 7526
RTLConsts.SWriteError 7526
RTLConsts.SWriteError constant 7526

RTLConsts.SYardsDescription 7526
RTLConsts.SYardsDescription constant 7526
RTLConsts.SYearsDescription 7527
RTLConsts.SYearsDescription constant 7527

S

saCommit enumeration member 3765
saCommitRetaining enumeration member 3765
saIgnore enumeration member 2592
saMaximize enumeration member 2592
samBorder enumeration member 7384
samButton enumeration member 7384
samButtonHover enumeration member 7384
samButtonPressed enumeration member 7384
samButtonShadow enumeration member 7384
saMinimize enumeration member 2592
samMenuDisabled enumeration member 7384
samMenuHover enumeration member 7384
samMenuNormal enumeration member 7384
samMenuPressed enumeration member 7384
samMenuSplitHover enumeration member 7384
samMenuSplitNormal enumeration member 7384
samMenuSplitPressed enumeration member 7384
samMenuSubMenu enumeration member 7384
saNone enumeration member 3765
saRestore enumeration member 2592
saRollback enumeration member 3765
saRollbackRetaining enumeration member 3765
sbArrow enumeration member 7386
sbArrowDisabled enumeration member 7386
sbArrowPressed enumeration member 7386
sbDisabled enumeration member 7398
sbDisabledSelected enumeration member 7398
sbDisabledSeparator enumeration member 7386
sbGroup enumeration member 7386
sbGroupEnd enumeration member 7386
sbGroupEndHover enumeration member 7386
sbGroupEndHoverSelected enumeration member 7386
sbGroupEndInactive enumeration member 7386
sbGroupEndPressed enumeration member 7386
sbGroupEndSelected enumeration member 7386
sbGroupEndSplitHover enumeration member 7386
sbGroupEndSplitHoverSplit enumeration member 7386
sbGroupEndSplitInactive enumeration member 7386
sbGroupEndSplitInactiveSplit enumeration member 7386
sbGroupEndSplitPressed enumeration member 7386
sbGroupEndSplitPressedSplit enumeration member 7386
sbGroupMiddle enumeration member 7386
sbGroupMiddleHover enumeration member 7386
sbGroupMiddleHoverSelected enumeration member 7386
sbGroupMiddleInactive enumeration member 7386
sbGroupMiddlePressed enumeration member 7386
sbGroupMiddleSelected enumeration member 7386
sbGroupMiddleSplitHover enumeration member 7386
sbGroupMiddleSplitHoverSplit enumeration member 7386
sbGroupMiddleSplitInactive enumeration member 7386
sbGroupMiddleSplitInactiveSplit enumeration member 7386
sbGroupMiddleSplitPressed enumeration member 7386
sbGroupMiddleSplitPressedSplit enumeration member 7386
sbGroupSingle enumeration member 7386
sbGroupSingleHover enumeration member 7386
sbGroupSingleHoverSelected enumeration member 7386
sbGroupSingleInactive enumeration member 7386
sbGroupSinglePressed enumeration member 7386
sbGroupSingleSelected enumeration member 7386
sbGroupSingleSplitHover enumeration member 7386
sbGroupSingleSplitHoverSplit enumeration member 7386
sbGroupSingleSplitInactive enumeration member 7386
sbGroupSingleSplitInactiveSplit enumeration member 7386
sbGroupSingleSplitPressed enumeration member 7386
sbGroupSingleSplitPressedSplit enumeration member 7386
sbGroupSplit enumeration member 7386
sbGroupStart enumeration member 7386
sbGroupStartHover enumeration member 7386
sbGroupStartHoverSelected enumeration member 7386
sbGroupStartInactive enumeration member 7386
sbGroupStartInactiveSplit enumeration member 7386
sbGroupStartPressed enumeration member 7386

sbGroupStartSelected enumeration member 7386
sbGroupStartSplitHover enumeration member 7386
sbGroupStartSplitHoverSplit enumeration member 7386
sbGroupStartSplitInactive enumeration member 7386
sbGroupStartSplitInactiveSplit enumeration member 7386
sbGroupStartSplitPressed enumeration member 7386
sbGroupStartSplitPressedSplit enumeration member 7386
sbHorizontal enumeration member 2591
sbHover enumeration member 7398
sbHoverSelected enumeration member 7398
sbLarge enumeration member 7386
sbLargeArrowDown enumeration member 7386
sbLargeArrowDownDisabled enumeration member 7386
sbLargeDisabled enumeration member 7386
sbLargeHover enumeration member 7386
sbLargeHoverSelected enumeration member 7386
sbLargePressed enumeration member 7386
sbLargeSelected enumeration member 7386
sbLargeSplit enumeration member 7386
sbLargeSplitActive enumeration member 7386
sbLargeSplitDisabled enumeration member 7386
sbLargeSplitHover enumeration member 7386
sbLargeSplitHoverSelected enumeration member 7386
sbLargeSplitHoverSplit enumeration member 7386
sbLargeSplitPressed enumeration member 7386
sbLargeSplitPressedSplit enumeration member 7386
sbLargeSplitSelected enumeration member 7386
sbNormal enumeration member 7398
sbPressed enumeration member 7398
sbsDown enumeration member 7290
sbSelected enumeration member 7398
sbSeparator enumeration member 7386
sbSmall enumeration member 7386
sbSmallDisabled enumeration member 7386
sbSmallHover enumeration member 7386
sbSmallHoverSelected enumeration member 7386
sbSmallInactive enumeration member 7386
sbSmallPressed enumeration member 7386
sbSmallSelected enumeration member 7386
sbSmallSplit enumeration member 7386
sbSmallSplitDisabled enumeration member 7386
sbSmallSplitHover enumeration member 7386
sbSmallSplitHoverSelected enumeration member 7386
sbSmallSplitHoverSplit enumeration member 7386
sbSmallSplitInactive enumeration member 7386
sbSmallSplitInactiveSplit enumeration member 7386
sbSmallSplitPressed enumeration member 7386
sbSmallSplitPressedSplit enumeration member 7386
sbSmallSplitSelected enumeration member 7386
sbsUp enumeration member 7290
sbVertical enumeration member 2591
scButton enumeration member 7391
scButtonHover enumeration member 7391
scButtonIcon enumeration member 7391
scButtonInactive enumeration member 7391
scButtonPressed enumeration member 7391
scDisabled enumeration member 7390
scDisabledSelected enumeration member 7390
scHide enumeration member 157
scHover enumeration member 7390
scHoverSelected enumeration member 7390
scMaximize enumeration member 157
scMinimize enumeration member 157
scNormal enumeration member 7390
scPressed enumeration member 7390
scRestore enumeration member 157
scSelected enumeration member 7390
scShow enumeration member 157
scShowDefault enumeration member 157
scShowMaximized enumeration member 157
scShowMinimized enumeration member 157
scShowMinNoActive enumeration member 157
scShowNA enumeration member 157
scShowNoActivate enumeration member 157
scShowNormal enumeration member 157
sdAllowCreate enumeration member 1885
sdDown enumeration member 2990
sdLeft enumeration member 2990

sdNewFolder enumeration member 1884
sdNewUI enumeration member 1884
sdPerformCreate enumeration member 1885
sdPrompt enumeration member 1885
sdRight enumeration member 2990
sdShowEdit enumeration member 1884
sdShowFiles enumeration member 1884
sdShowShares enumeration member 1884
sdUp enumeration member 2990
sdValidateDir enumeration member 1884
sfBottomFrame enumeration member 7391
sfBottomFrameDisabled enumeration member 7391
sfButton enumeration member 7391
sfButtonHover enumeration member 7391
sfButtonPressed enumeration member 7391
sfClose enumeration member 7391
sfCloseDisabled enumeration member 7391
sfCloseHover enumeration member 7391
sfClosePressed enumeration member 7391
sfHelp enumeration member 7391
sfHelpDisabled enumeration member 7391
sfHelpHover enumeration member 7391
sfHelpPressed enumeration member 7391
sfLeftFrame enumeration member 7391
sfLeftFrameDisabled enumeration member 7391
sfLeftFrameTop enumeration member 7391
sfLeftFrameTopDisabled enumeration member 7391
sfMaximize enumeration member 7391
sfMaximizeDisabled enumeration member 7391
sfMaximizeHover enumeration member 7391
sfMaximizePressed enumeration member 7391
sfMinimize enumeration member 7391
sfMinimizeDisabled enumeration member 7391
sfMinimizeHover enumeration member 7391
sfMinimizePressed enumeration member 7391
sfNone enumeration member 7391
sfRestore enumeration member 7391
sfRestoreDisabled enumeration member 7391
sfRestoreHover enumeration member 7391
sfRestorePressed enumeration member 7391
sfRightFrame enumeration member 7391
sfRightFrameDisabled enumeration member 7391
sfRightFrameTop enumeration member 7391
sfRightFrameTopDisabled enumeration member 7391
sgButton enumeration member 7394
sgDownButton enumeration member 7394
sgDownButtonDisabled enumeration member 7394
sgDownButtonHover enumeration member 7394
sgDownButtonPressed enumeration member 7394
sgMoreButton enumeration member 7394
sgMoreButtonHover enumeration member 7394
sgMoreButtonHoverDisabled enumeration member 7394
sgUpButton enumeration member 7394
sgUpButtonDisabled enumeration member 7394
sgUpButtonHover enumeration member 7394
sgUpButtonPressed enumeration member 7394
ShowIBMessage enumeration member 3555
ShowSQLCode enumeration member 3555
ShowSQLMessage enumeration member 3555
sltID enumeration member 1546
sltURL enumeration member 1546
smArrowLeftDisabledIcon enumeration member 7395
smArrowLeftIcon enumeration member 7395
smArrowRightDisabledIcon enumeration member 7395
smArrowRightIcon enumeration member 7395
smAutomation enumeration member 5962
smAutoSize enumeration member 6058
smBackground enumeration member 7395
smBanner enumeration member 7395
smCenter enumeration member 6058
smCheck enumeration member 7395
smCheckDisabled enumeration member 7395
smCheckDisabledIcon enumeration member 7395
smCheckIcon enumeration member 7395
smClip enumeration member 6058
smDisabled enumeration member 7395
smHover enumeration member 7395
smNormal enumeration member 7395

smOptionButton enumeration member 7395
smOptionButtonDisabled enumeration member 7395
smOptionButtonHover enumeration member 7395
smRadio enumeration member 7395
smRadioDisabled enumeration member 7395
smRegServer enumeration member 5962
smResize enumeration member 7395
smResizeBoth enumeration member 7395
smResizeVertical enumeration member 7395
smRichContentDisabled enumeration member 7395
smRichContentHover enumeration member 7395
smScale enumeration member 6058
smScrollButton enumeration member 7395
smScrollButtonDownIcon enumeration member 7395
smScrollButtonHover enumeration member 7395
smScrollButtonUpIcon enumeration member 7395
smSelected enumeration member 7395
smSeparator enumeration member 7395
smStandalone enumeration member 5962
smStretch enumeration member 6058
smUnregServer enumeration member 5962
soCacheMimeResponse enumeration member 6162
soCustomFaultAtDetailsNode enumeration member 6162
soDocument enumeration member 6162
soDontClearOutBoundHeaders enumeration member 6162
soDontSendEmptyNodes enumeration member 6162
soLiteralParams enumeration member 6162
soReturnSuccessForFault enumeration member 6162
soRootRefNodesToBody enumeration member 6162
soSendMultiRefArray enumeration member 6162
soSendMultiRefObj enumeration member 6162
soSendUntyped enumeration member 6162
soTryAllSchema enumeration member 6162
soUTF8EncodeXML enumeration member 6162
soUTF8InHeader enumeration member 6162
soXXXXHdr enumeration member 6162
SPX enumeration member 4020
sqBackground enumeration member 7397
sqButton enumeration member 7397
sqButtonCustomizeIcon enumeration member 7397
sqButtonHover enumeration member 7397
sqButtonMoreIcon enumeration member 7397
sqButtonPressed enumeration member 7397
sqEnd enumeration member 7397
sqEndRTL enumeration member 7397
sqGroup enumeration member 7397
sqGroupHover enumeration member 7397
sqGroupPressed enumeration member 7397
SQLCommit enumeration member 4098
SQLDDL enumeration member 4098
SQLDelete enumeration member 4098
SQLException class 4530
- about SQLException class 4530
- ErrorCode 4531
- SQLException 4531
- SQLException constructor 4531
- SQLException Members 4530
- SQLException properties 4531
- SQLState 4532

SQLExecProcedure enumeration member 4098
SQLGetSegment enumeration member 4098
SQLInsert enumeration member 4098
SQLPutSegment enumeration member 4098
SQLRollback enumeration member 4098
SQLSelect enumeration member 4098
SQLSelectForUpdate enumeration member 4098
SQLSetGenerator enumeration member 4098
SQLStartTransaction enumeration member 4098
SQLUnknown enumeration member 4098
SQLUpdate enumeration member 4098
sqMiddle enumeration member 7397
sqStart enumeration member 7397
sqStartRTL enumeration member 7397
srAbort enumeration member 3872
srBackground enumeration member 7399
srBackgroundDisabled enumeration member 7399
srBody enumeration member 7399
srContinue enumeration member 3872

srgBackground enumeration member 7400
srgBackgroundDisabled enumeration member 7400
srgBackgroundHover enumeration member 7400
srgCaption enumeration member 7400
srgCaptionDisabled enumeration member 7400
srgCaptionHover enumeration member 7400
srgDialogButton enumeration member 7400
srgDialogButtonHover enumeration member 7400
srgDialogButtonIcon enumeration member 7400
srgDialogButtonIconHover enumeration member 7400
srgDialogButtonIconPressed enumeration member 7400
srgDialogButtonPressed enumeration member 7400
srgMinimizeButton enumeration member 7400
srgMinimizeButtonHover enumeration member 7400
srgMinimizeButtonIcon enumeration member 7400
srgMinimizeButtonPressed enumeration member 7400
srgMinimizeButtonPressedIcon enumeration member 7400
srHeader enumeration member 7399
srHelp enumeration member 7399
srPage enumeration member 7399
ssAbsolute enumeration member 1545
ssAuto enumeration member 1545
ssbBar enumeration member 7402
ssbBarHover enumeration member 7402
ssbBarIcon enumeration member 7402
ssbBarPressed enumeration member 7402
ssbButton enumeration member 7402
ssbButtonActive enumeration member 7402
ssbButtonDisabled enumeration member 7402
ssbButtonDownIcon enumeration member 7402
ssbButtonHover enumeration member 7402
ssbButtonPressed enumeration member 7402
ssbButtonUpIcon enumeration member 7402
ssbrBackground enumeration member 7403
ssbrControl enumeration member 7403
ssbrGripper enumeration member 7403
ssbrNotification enumeration member 7403
ssButton enumeration member 7403
ssButtonDisabled enumeration member 7403
ssButtonDown enumeration member 7403
ssButtonDownIcon enumeration member 7403
ssButtonHover enumeration member 7403
ssButtonInactive enumeration member 7403
ssButtonPressed enumeration member 7403
ssButtonUp enumeration member 7403
ssButtonUpIcon enumeration member 7403
ssFlat enumeration member 2592
ssHotTrack enumeration member 2592
ssPercent enumeration member 1545
ssRegular enumeration member 2592
stBackground enumeration member 7404
stBlue enumeration member 2989
stBoth enumeration member 4586
stCircle enumeration member 1544
stCombo enumeration member 2989
stData enumeration member 4586
stEllipse enumeration member 1544
stGreen enumeration member 2989
stHover enumeration member 7404
stHoverSelected enumeration member 7404
stHoverUnselected enumeration member 7404
stHue enumeration member 2989
stKeyTipsActive enumeration member 7404
stLuminance enumeration member 2989
stNone enumeration member 4586
stNormal enumeration member 7404
stRectangle enumeration member 1544
stRed enumeration member 2989
stRoundRect enumeration member 1544
stRoundSquare enumeration member 1544
stSaturation enumeration member 2989
stScrollGroupLeft enumeration member 7404
stScrollGroupLeftHover enumeration member 7404
stScrollGroupRight enumeration member 7404
stScrollGroupRightHover enumeration member 7404
stScrollLeft enumeration member 7404
stScrollLeftHover enumeration member 7404
stScrollLeftPressed enumeration member 7404

stScrollRight enumeration member 7404
stScrollRightHover enumeration member 7404
stScrollRightPressed enumeration member 7404
stSeparator enumeration member 7404
stSquare enumeration member 1544
stText enumeration member 4586
SweepDB enumeration member 4026
SystemRelations enumeration member 4023

T

TAbstractContentParser class 3255
about TAbstractContentParser class 3255
CanParse 3257
Create 3256
GetContentFields 3257
GetFiles 3258
TAbstractContentParser Members 3256
TAbstractContentParser methods 3256
TAbstractContentParser properties 3258
WebRequest 3258
TAbstractScriptEnginesList class 3455
about TAbstractScriptEnginesList class 3455
Create 3456
FindScriptProducerClass 3456
TAbstractScriptEnginesList Members 3456
TAbstractScriptEnginesList methods 3456
TAbstractScriptError class 3457
about TAbstractScriptError class 3457
CharPos 3459
Description 3459
FileName 3460
GetCharPos 3458
GetDescription 3458
GetFileName 3458
GetLine 3458
GetSourceLine 3458
Line 3460
SetDescription 3459
SourceLine 3460
TAbstractScriptError Members 3457
TAbstractScriptError methods 3457
TAbstractScriptError properties 3459
TAbstractScriptErrors class 3460
about TAbstractScriptErrors class 3460
Add 3461
Count 3462
Errors 3462
GetError 3461
GetErrorCount 3462
TAbstractScriptErrors Members 3461
TAbstractScriptErrors methods 3461
TAbstractScriptErrors properties 3462
TAbstractScriptProducer class 3463
about TAbstractScriptProducer class 3463
Create 3463
TAbstractScriptProducer Members 3463
TAbstractWebModuleFactory class 3259
about TAbstractWebModuleFactory class 3259
CacheMode 3264
ComponentClass 3264
CreateMode 3265
GetCacheMode 3260
GetComponentClass 3261
GetCreateMode 3261
GetIsAppModule 3261
GetModule 3262
GetModuleName 3262
GetWebPageInfo 3263
IsAppModule 3265
ModuleName 3265
PreventDestruction 3263
TAbstractWebModuleFactory Members 3259
TAbstractWebModuleFactory methods 3260
TAbstractWebModuleFactory properties 3263
WebPageInfo 3266
TAbstractWebModuleList class 3266
about TAbstractWebModuleList class 3266
AddModuleClass 3268

AddModuleName 3269
Factory 3272
FactoryCount 3272
FindModuleClass 3269
FindModuleName 3270
GetFactory 3270
GetFactoryCount 3270
GetItem 3271
GetItemCount 3271
GetOnModuleAdded 3271
ItemCount 3273
Items 3273
OnModuleAdded 3274
SetOnModuleAdded 3271
TAbstractWebModuleList events 3274
TAbstractWebModuleList Members 3267
TAbstractWebModuleList methods 3268
TAbstractWebModuleList properties 3272
TAbstractWebPageInfo class 3275
about TAbstractWebPageInfo class 3275
Factory 3281
GetFactory 3277
GetIsLoginRequired 3277
GetIsPublished 3277
GetPageDescription 3278
GetPageFile 3278
GetPageHREF 3278
GetPageName 3279
GetPageTitle 3279
GetViewAccess 3279
IsLoginRequired 3281
IsPublished 3282
PageDescription 3282
PageFile 3283
PageHREF 3283
PageName 3283
PageTitle 3284
SetFactory 3280
TAbstractWebPageInfo Members 3276
TAbstractWebPageInfo methods 3276
TAbstractWebPageInfo properties 3280
ViewAccess 3284
TAbstractWebPageModuleFactory class 3285
about TAbstractWebPageModuleFactory class 3285
Create 3286
Destroy 3286
GetWebPageInfo 3287
TAbstractWebPageModuleFactory Members 3285
TAbstractWebPageModuleFactory methods 3287
TAbstractWebRequestFile class 3288
about TAbstractWebRequestFile class 3288
ContentType 3291
FieldName 3291
FileName 3291
GetContentType 3289
GetFieldName 3289
GetFileName 3290
GetStream 3290
Stream 3292
TAbstractWebRequestFile Members 3288
TAbstractWebRequestFile methods 3289
TAbstractWebRequestFile properties 3291
TAbstractWebRequestFiles class 3292
about TAbstractWebRequestFiles class 3292
Count 3294
GetCount 3293
GetItem 3294
Items 3295
TAbstractWebRequestFiles Members 3293
TAbstractWebRequestFiles methods 3293
TAbstractWebRequestFiles properties 3294
TAbstractWebSession class 3295
about TAbstractWebSession class 3295
GetSessionID 3297
GetTimeoutMinutes 3297
GetValue 3298
SessionID 3300
SetTimeoutMinutes 3298

SetValue 3299
TAbstractWebSession Members 3296
TAbstractWebSession methods 3297
TAbstractWebSession properties 3300
Terminate 3299
TimeoutMinutes 3300
UpdateResponse 3299
Values 3301
TACommit enumeration member 3766
TACommitRetaining enumeration member 3766
TAlphaBitmap class 7369
about TAlphaBitmap class 7369
Create 7371
TAlphaBitmap Members 7369
TApplication class 1924
about TApplication class 1924
ActionUpdateDelay 1968
ActivateHint 1930
Active 1968
ActiveFormHandle 1968
AddPopupForm 1931
AllowTesting 1969
AutoDragDocking 1969
BiDiKeyboard 1969
BiDiMode 1970
BringToFront 1931
CancelHint 1931
ControlDestroyed 1931
Create 1928
CreateForm 1932
CreateHandle 1934
CurrentHelpFile 1970
DefaultFont 1971
Destroy 1928
DialogHandle 1971
DoApplicationIdle 1935
EnumAllWindowsOnActivateHint 1972
ExecuteAction 1935
ExeName 1972
Handle 1974
HandleException 1935
HandleMessage 1937
HelpCommand 1941
HelpContext 1941
HelpFile 1974
HelpJump 1942
HelpKeyword 1942
HelpShowTableOfContents 1943
HelpSystem 1976
HideHint 1943
Hint 1976
HintColor 1979
HintHidePause 1980
HintMouseMessage 1944
HintPause 1980
HintShortCuts 1980
HintShortPause 1981
HookMainWindow 1944
HookSynchronizeWakeup 1944
Icon 1981
Idle 1945
Initialize 1945
IsDlgMsg 1947
IsHintMsg 1947
IsKeyMsg 1948
IsMDIMsg 1948
IsPreProcessMessage 1948
IsRightToLeft 1948
IsShortCut 1949
MainForm 1983
MainFormHandle 1983
MainFormOnTaskBar 1983
MessageBox 1949
Minimize 1951
ModalFinished 1952
ModalLevel 1985
ModalPopupMode 1985
ModalStarted 1952

NonBiDiKeyboard 1985
NormalizeAllTopMosts 1952
NormalizeTopMosts 1953
OnActionExecute 1990
OnActionUpdate 1991
OnActivate 1992
OnDeactivate 1992
OnException 1993
OnGetActiveFormHandle 1995
OnGetMainFormHandle 1995
OnHelp 1995
OnHint 1997
OnIdle 1998
OnMessage 2003
OnMinimize 2004
OnModalBegin 2006
OnModalEnd 2006
OnRestore 2006
OnSettingChange 2007
OnShortCut 2007
OnShowHint 2008
PopupControlWnd 1986
ProcessMessages 1954
RemovePopupForm 1956
Restore 1957
RestoreTopMosts 1958
Run 1959
ShowException 1961
ShowHint 1986
ShowMainForm 1987
TApplication events 1990
TApplication Members 1925
TApplication methods 1929
TApplication properties 1967
Terminate 1962
Terminated 1988
Title 1988
UnhookMainWindow 1964
UnhookSynchronizeWakeup 1964
UpdateAction 1965
UpdateFormatSettings 1989
UpdateMetricSettings 1989
UseRightToLeftAlignment 1966
UseRightToLeftReading 1966
UseRightToLeftScrollBar 1966
TApplicationMenu class 6716
about TApplicationMenu class 6716
Assign 6718
Caption 6719
CommandType 6720
Create 6717
Destroy 6717
GetActionClientItem 6718
Icon 6720
IconSize 6720
Initialize 6719
KeyTip 6720
Menu 6721
ScreenTip 6721
Show 6719
TApplicationMenu Members 6716
TApplicationMenu methods 6717
TApplicationMenu properties 6719
TApplyUpdatesButton class 5250
about TApplyUpdatesButton class 5250
AddElements 5252
Caption 5253
Create 5251
Custom 5254
GetSubComponents 5252
ImplAddElements 5252
ImplContent 5252
Style 5254
StyleRule 5254
TApplyUpdatesButton Members 5251
TApplyUpdatesButton methods 5252
TApplyUpdatesButton properties 5253
XMLBroker 5255

XMLUseParent 5255
TARollback enumeration member 3766
TARollbackRetaining enumeration member 3766
TAutoDispatch class 5942
about TAutoDispatch class 5942
AddRef 5944
AutoObject 5946
Create 5943
GetAutoObject 5944
GetIDsOfNames 5944
GetTypeInfo 5944
GetTypeInfoCount 5944
Invoke 5945
QueryInterface 5945
Release 5945
TAutoDispatch Members 5943
TAutoDispatch methods 5943
TAutoDispatch properties 5945
TAutomation class 5946
about TAutomation class 5946
AutoObjectCount 5948
Create 5947
Destroy 5947
IsInprocServer 5948
OnLastRelease 5949
RegisterClass 5947
StartMode 5948
TAutomation events 5948
TAutomation Members 5946
TAutomation methods 5947
TAutomation properties 5948
UpdateRegistry 5947
TAutoObject class 5949
about TAutoObject class 5949
AddRef 5951
AutoDispatch 5952
Create 5950
CreateAutoDispatch 5951
Destroy 5950
GetExceptionInfo 5951
OleObject 5952
RefCount 5952
Release 5952
TAutoObject Members 5949
TAutoObject methods 5951
TAutoObject properties 5952
TBaseInetXPageProducer class 5701
about TBaseInetXPageProducer class 5701
AddIncludes 5705
AddScriptComponents 5705
CanAddClass 5705
ContentFromStream 5706
Create 5703
CreatePageElements 5706
CreateScriptManager 5707
DeclareFunctions 5707
DeclareIncludes 5707
DeclareRowSets 5707
DeclareScript 5708
DeclareStyles 5708
DeclareWarnings 5708
DeclareXMLDocuments 5708
DeclareXMLIslands 5708
Destroy 5704
DoAfterGetContent 5709
DoAfterGetXMLData 5709
DoBeforeGetContent 5709
DoBeforeGetXMLData 5709
DoTagEvent 5710
EnableXMLIslands 5715
FindXMLBroker 5710
FindXMLBrokerName 5710
GeneratePageElements 5711
GetContentOptions 5711
GetDefaultTemplate 5711
GetRequiredTags 5711
GetScriptManager 5712
GetUseXMLIslands 5712

GetXMLData 5712
GetXMLDelta 5713
GetXMLErrors 5713
ImplCanAddClass 5713
IncludePathURL 5715
OnAfterGetContent 5718
OnAfterGetXMLData 5718
OnBeforeGetContent 5719
OnBeforeGetXMLData 5719
PageElements 5716
ScriptManager 5716
SetIncludePathURL 5713
SetStyles 5714
SetStylesFile 5714
SetWebContentOptions 5714
Styles 5716
StylesFile 5717
TBaseInetXPageProducer events 5717
TBaseInetXPageProducer Members 5702
TBaseInetXPageProducer methods 5704
TBaseInetXPageProducer properties 5715
TBasePageItemsProducer class 6390
about TBasePageItemsProducer class 6390
Create 6391
Destroy 6391
GetChildren 6392
GetComponentList 6392
GetDefaultComponentList 6392
SetChildOrder 6393
TBasePageItemsProducer Members 6391
TBasePageItemsProducer methods 6392
TBasePageItemsProducer properties 6393
WebPageItems 6393
TBasePageProducer class 3464
about TBasePageProducer class 3464
Content 3466
ContentFromStream 3467
ContentFromString 3467
Create 3466
DoTagEvent 3468
GetProducerTemplateFileName 3468
GetProducerTemplateStream 3468
GetScriptEngine 3469
GetTagID 3469
GetTemplateFileName 3469
GetTemplateStream 3469
HandleScriptTag 3469
HandleTag 3470
ImplHandleTag 3470
OnHTMLTag 3472
ScriptEngine 3471
ServerScriptFromStream 3470
StripParamQuotes 3471
TBasePageProducer events 3472
TBasePageProducer Members 3465
TBasePageProducer methods 3466
TBasePageProducer properties 3471
UseScriptEngine 3470
WebModuleContext 3471
TBaseProvider class 6445
about TBaseProvider class 6445
AfterUpdateRecord 6458
BeforeUpdateRecord 6458
CheckResolver 6449
Create 6447
CreateDataPacket 6449
CreateResolver 6450
Destroy 6448
DoAfterUpdateRecord 6450
DoBeforeCommit 6450
DoBeforeUpdateRecord 6451
DoOnGetData 6451
DoOnUpdateData 6451
FetchDetails 6451
FreeResolver 6452
InternalApplyUpdates 6452
InternalGetRecords 6453
InternalRowRequest 6454

LocateRecord 6454
OnDataRequest 6458
OnGetData 6459
OnUpdateData 6459
OnUpdateError 6460
Options 6455
PacketDataSet 6456
Resolver 6456
TBaseProvider events 6457
TBaseProvider Members 6446
TBaseProvider methods 6448
TBaseProvider properties 6455
UpdateMode 6457
UpdateRecord 6455
TBevel class 163
about TBevel class 163
Align 165
Anchors 169
Constraints 169
Create 164
Paint 165
ParentShowHint 170
Shape 170
ShowHint 171
Style 172
TBevel Members 164
TBevel methods 164
TBevel properties 165
Visible 173
tbHorizontal enumeration member 2593
TBitmap class 2628
about TBitmap class 2628
AlphaFormat 2651
Assign 2633
Canvas 2651
Changed 2635
Create 2631
Destroy 2632
Dormant 2635
Draw 2636
DrawTransparent 2637
FreeImage 2637
GetEmpty 2638
GetHeight 2638
GetPalette 2639
GetSupportsPartialTransparency 2639
GetWidth 2639
Handle 2652
HandleAllocated 2639
HandleNeeded 2640
HandleType 2653
IgnorePalette 2654
LoadFromClipboardFormat 2640
LoadFromResourceID 2642
LoadFromResourceName 2642
LoadFromStream 2643
Mask 2644
MaskHandle 2654
MaskHandleNeeded 2644
Monochrome 2654
PaletteNeeded 2645
PixelFormat 2656
ReadData 2645
ReleaseHandle 2645
ReleaseMaskHandle 2646
ReleasePalette 2647
SaveToClipboardFormat 2647
SaveToStream 2649
ScanLine 2658
SetHeight 2649
SetPalette 2649
SetSize 2650
SetWidth 2650
TBitmap Members 2629
TBitmap methods 2632
TBitmap properties 2650
TransparentColor 2660
TransparentMode 2661

WriteData 2650
TBitmapImage class 2662
about TBitmapImage class 2662
Destroy 2663
FreeHandle 2663
TBitmapImage Members 2662
TBitmapImage methods 2663
tbLeftButton enumeration member 4968
TBoundLabel class 174
about TBoundLabel class 174
AdjustBounds 176
BiDiMode 178
Caption 178
Color 181
Create 176
DragCursor 181
DragKind 182
DragMode 182
Font 184
Height 185
Layout 186
Left 186
OnClick 201
OnContextPopup 203
OnDblClick 203
OnDragDrop 204
OnDragOver 207
OnEndDock 210
OnEndDrag 210
OnMouseActivate 211
OnMouseDown 212
OnMouseMove 214
OnMouseUp 217
OnStartDock 219
OnStartDrag 220
ParentBiDiMode 187
ParentColor 188
ParentFont 188
ParentShowHint 189
PopupMenu 189
ShowAccelChar 194
ShowHint 195
TBoundLabel events 200
TBoundLabel Members 175
TBoundLabel methods 176
TBoundLabel properties 177
Top 197
Transparent 198
Width 198
WordWrap 200
tbRightButton enumeration member 4968
TBrush class 2664
about TBrush class 2664
Assign 2666
Bitmap 2668
Color 2669
Create 2665
Destroy 2665
GetBitmap 2666
GetColor 2666
GetHandle 2667
GetStyle 2667
Handle 2670
SetBitmap 2667
SetColor 2667
SetHandle 2668
SetStyle 2668
Style 2670
TBrush Members 2664
TBrush methods 2666
TBrush properties 2668
TBrushRecall class 2671
about TBrushRecall class 2671
Create 2672
TBrushRecall Members 2672
TButtonedEdit class 220
about TButtonedEdit class 220
Align 227

Alignment 230
Anchors 231
AutoSelect 231
AutoSize 232
BevelEdges 233
BevelInner 233
BevelKind 234
BevelOuter 234
BevelWidth 234
BiDiMode 235
BorderStyle 235
CharCase 236
Color 236
Constraints 237
Ctl3D 237
DoubleBuffered 238
DragCursor 240
DragKind 240
DragMode 240
Enabled 242
Font 242
HideSelection 244
Images 244
ImeMode 247
ImeName 248
LeftButton 248
MaxLength 252
NumbersOnly 252
OEMConvert 252
OnChange 270
OnClick 271
OnContextPopup 273
OnDblClick 273
OnDragDrop 274
OnDragOver 277
OnEndDock 280
OnEndDrag 280
OnEnter 281
OnExit 284
OnKeyDown 287
OnKeyPress 290
OnKeyUp 291
OnLeftButtonClick 293
OnMouseActivate 296
OnMouseDown 297
OnMouseEnter 299
OnMouseLeave 300
OnMouseMove 300
OnMouseUp 302
OnRightButtonClick 304
OnStartDock 307
OnStartDrag 308
ParentBiDiMode 253
ParentColor 253
ParentCtl3D 254
ParentDoubleBuffered 254
ParentFont 254
ParentShowHint 255
PasswordChar 255
PopupMenu 256
ReadOnly 261
RightButton 261
ShowHint 264
TabOrder 266
TabStop 266
TButtonedEdit events 270
TButtonedEdit Members 224
TButtonedEdit properties 226
Text 267
TextHint 268
Visible 268
tbVertical enumeration member 2593
TCanvas class 2673
about TCanvas class 2673
Arc 2679
Brush 2716
BrushCopy 2680
CanvasOrientation 2718

Changed 2681
Changing 2682
Chord 2682
ClipRect 2718
CopyMode 2719
CopyRect 2683
Create 2675
CreateHandle 2685
Destroy 2678
Draw 2685
Draw method 2685
DrawFocusRect 2687
Ellipse 2688
Ellipse method 2688
FillRect 2689
FloodFill 2690
Font 2721
FrameRect 2692
Handle 2722
HandleAllocated 2693
LineTo 2693
Lock 2694
LockCount 2725
MoveTo 2695
OnChange 2729
OnChanging 2730
Pen 2725
PenPos 2727
Pie 2695
Pixels 2727
PolyBezier 2696
PolyBezierTo 2697
Polygon 2697
Polyline 2698
Rectangle 2700
Rectangle method 2700
Refresh 2701
RequiredState 2702
RoundRect 2702
RoundRect method 2702
StretchDraw 2704
TCanvas events 2729
TCanvas Members 2673
TCanvas methods 2678
TCanvas properties 2716
TextExtent 2705
TextFlags 2728
TextHeight 2706
TextOut 2707
TextRect 2713
TextRect method 2713
TextWidth 2714
TryLock 2715
Unlock 2716
TCaptionAttributes class 5256
about TCaptionAttributes class 5256
Align 5257
Create 5257
Custom 5258
Parent 5258
Style 5258
StyleRule 5259
TCaptionAttributes Members 5256
TCaptionAttributes properties 5257
VAlign 5259
TCategoryPanel class 309
about TCategoryPanel class 309
BiDiMode 313
Caption 314
Collapsed 316
CollapsedHotImageIndex 316
CollapsedImageIndex 317
CollapsedPressedImageIndex 317
Color 318
Ctl3D 318
DockSite 319
DoubleBuffered 320
DragCursor 322

DragKind 322
DragMode 322
Enabled 324
ExpandedHotImageIndex 324
ExpandedImageIndex 325
ExpandedPressedImageIndex 325
Font 326
FullRepaint 327
Height 327
Left 328
Locked 329
OnClick 344
OnCollapse 346
OnContextPopup 346
OnDblClick 347
OnDockDrop 347
OnDockOver 349
OnDragDrop 352
OnDragOver 355
OnEndDock 358
OnEndDrag 358
OnEnter 359
OnExit 362
OnExpand 364
OnGetSiteInfo 365
OnMouseActivate 367
OnMouseDown 368
OnMouseEnter 370
OnMouseLeave 370
OnMouseMove 371
OnMouseUp 373
OnStartDock 375
OnStartDrag 376
OnUnDock 376
Padding 329
ParentBackground 330
ParentBiDiMode 330
ParentColor 330
ParentCtl3D 331
ParentDoubleBuffered 331
ParentFont 332
ParentShowHint 332
PopupMenu 333
ShowHint 338
TabOrder 339
TabStop 340
TCategoryPanel events 344
TCategoryPanel Members 309
TCategoryPanel properties 312
UseDockManager 340
Visible 340
Width 342
TCategoryPanelGroup class 379
about TCategoryPanelGroup class 379
Align 383
Anchors 387
BevelEdges 387
BevelInner 388
BevelKind 388
BevelOuter 389
BevelWidth 389
BiDiMode 389
ChevronAlignment 390
ChevronColor 390
ChevronHotColor 391
Color 391
Constraints 392
Ctl3D 393
DockSite 393
DoubleBuffered 394
DragCursor 396
DragKind 396
DragMode 397
Enabled 398
Font 399
GradientBaseColor 400
GradientColor 400
GradientDirection 400

HeaderAlignment 401
HeaderFont 401
HeaderHeight 401
HeaderImage 402
HeaderStyle 402
Height 402
Images 403
OnCanResize 418
OnClick 419
OnConstrainedResize 421
OnContextPopup 421
OnDblClick 422
OnDockDrop 422
OnDockOver 424
OnDragDrop 427
OnDragOver 430
OnEndDock 433
OnEndDrag 433
OnEnter 434
OnExit 437
OnGetSiteInfo 439
OnMouseActivate 442
OnMouseDown 443
OnMouseEnter 445
OnMouseLeave 445
OnMouseMove 445
OnMouseUp 448
OnMouseWheel 450
OnMouseWheelDown 450
OnMouseWheelUp 451
OnResize 451
OnStartDock 452
OnStartDrag 452
OnUnDock 453
Padding 404
ParentBackground 404
ParentBiDiMode 404
ParentColor 405
ParentCtl3D 405
ParentDoubleBuffered 406
ParentFont 406
ParentShowHint 406
PopupMenu 407
ShowHint 412
TabOrder 414
TabStop 414
TCategoryPanelGroup events 418
TCategoryPanelGroup Members 379
TCategoryPanelGroup properties 382
Visible 415
Width 416
TCategoryPanelSurface class 456
about TCategoryPanelSurface class 456
Create 456
CreateParams 457
TCategoryPanelSurface Members 456
TCategoryPanelSurface methods 457
TCellCollection class 458
about TCellCollection class 458
GetAttr 459
GetAttrCount 459
GetItem 460
GetItemAttr 460
Items 461
Owner 460
SetItem 461
TCellCollection Members 458
TCellCollection methods 458
TCellCollection properties 461
Update 461
TCellItem class 462
about TCellItem class 462
AssignTo 463
AutoAdded 464
Create 462
SetSizeStyle 464
SetValue 464
Size 465

SizeStyle 465
TCellItem Members 462
TCellItem methods 463
TCellItem properties 464
Value 465
TChangeLink class 4190
about TChangeLink class 4190
Change 4191
Destroy 4191
OnChange 4192
Sender 4192
TChangeLink events 4192
TChangeLink Members 4190
TChangeLink methods 4191
TChangeLink properties 4192
tcLowerCase enumeration member 1886
TColorBox class 466
about TColorBox class 466
Align 469
Anchors 472
AutoComplete 473
AutoDropDown 474
BevelEdges 474
BevelInner 475
BevelKind 475
BevelOuter 476
BiDiMode 476
Color 477
Constraints 477
Ctl3D 478
DefaultColorColor 479
DoubleBuffered 479
DropDownCount 481
Enabled 481
Font 482
ItemHeight 483
NoneColorColor 483
OnChange 497
OnClick 499
OnCloseUp 501
OnContextPopup 501
OnDblClick 502
OnDragDrop 502
OnDragOver 505
OnDropDown 508
OnEndDock 510
OnEndDrag 510
OnEnter 511
OnExit 514
OnGetColors 516
OnKeyDown 517
OnKeyPress 520
OnKeyUp 521
OnMouseEnter 523
OnMouseLeave 523
OnSelect 524
OnStartDock 524
OnStartDrag 525
ParentBiDiMode 484
ParentColor 484
ParentCtl3D 485
ParentDoubleBuffered 485
ParentFont 485
ParentShowHint 486
PopupMenu 486
Selected 491
ShowHint 492
Style 493
TabOrder 493
TabStop 494
TColorBox events 496
TColorBox Members 466
TColorBox properties 468
Visible 494
TColorListBox class 525
about TColorListBox class 525
Align 528
Anchors 532

AutoComplete 532
BevelEdges 533
BevelInner 533
BevelKind 533
BevelOuter 534
BiDiMode 534
Color 535
Constraints 535
Ctl3D 536
DefaultColorColor 536
DoubleBuffered 536
Enabled 538
Font 539
ItemHeight 540
NoneColorColor 540
OnClick 555
OnContextPopup 557
OnDblClick 558
OnDragDrop 558
OnDragOver 561
OnEndDock 564
OnEndDrag 565
OnEnter 566
OnExit 568
OnGetColors 571
OnKeyDown 571
OnKeyPress 575
OnKeyUp 576
OnMouseActivate 577
OnMouseDown 578
OnMouseEnter 581
OnMouseLeave 581
OnMouseMove 581
OnMouseUp 584
OnStartDock 586
OnStartDrag 586
ParentBiDiMode 540
ParentColor 541
ParentCtl3D 541
ParentDoubleBuffered 542
ParentFont 542
ParentShowHint 542
PopupMenu 543
Selected 548
ShowHint 550
Style 551
TabOrder 552
TabStop 553
TColorListBox events 554
TColorListBox Members 526
TColorListBox properties 527
Visible 553
TColumnCollection class 587
about TColumnCollection class 587
Add 589
Create 588
GetItemAttr 589
Notify 589
TColumnCollection Members 588
TColumnCollection methods 588
TColumnItem class 590
about TColumnItem class 590
TCOMConnection class 4784
about TCOMConnection class 4784
DoConnect 4786
SetConnected 4786
TCOMConnection Members 4785
TCOMConnection methods 4785
TConfigFileData class 3939
about TConfigFileData class 3939
ConfigFileKey 3941
ConfigFileValue 3941
Create 3940
Destroy 3940
TConfigFileData fields 3940
TConfigFileData Members 3940
TConfigParams class 3941
about TConfigParams class 3941

BaseLocation 3942
ConfigFileData 3943
ConfigFileParams 3943
Create 3942
Destroy 3942
LockFileLocation 3943
MessageFileLocation 3943
SecurityDatabaseLocation 3944
TConfigParams fields 3942
TConfigParams Members 3941
TContentParser class 3301
about TContentParser class 3301
CanParse 3303
Destroy 3302
GetContentFields 3303
GetFiles 3304
TContentParser Members 3302
TContentParser methods 3303
TControlBar class 590
about TControlBar class 590
Align 594
Anchors 597
AutoDock 598
AutoDrag 598
AutoSize 599
BevelEdges 599
BevelInner 600
BevelKind 600
BevelOuter 601
BevelWidth 601
BorderWidth 601
Canvas 602
Color 604
Constraints 604
CornerEdge 605
DockSite 605
DragCursor 606
DragKind 607
DragMode 607
DrawingStyle 609
Enabled 609
GradientDirection 609
GradientEndColor 609
GradientStartColor 610
OnAlignInsertBefore 623
OnAlignPosition 624
OnBandDrag 624
OnBandInfo 625
OnBandMove 625
OnBandPaint 626
OnBeginBandMove 626
OnCanResize 626
OnClick 627
OnConstrainedResize 628
OnContextPopup 629
OnDblClick 630
OnDockDrop 630
OnDockOver 632
OnDragDrop 634
OnDragOver 637
OnEndBandMove 640
OnEndDock 640
OnEndDrag 641
OnEnter 642
OnExit 644
OnGetSiteInfo 647
OnMouseActivate 649
OnMouseDown 650
OnMouseEnter 652
OnMouseLeave 653
OnMouseMove 653
OnMouseUp 655
OnPaint 657
OnResize 658
OnStartDock 659
OnStartDrag 659
OnUnDock 660
ParentBackground 610

ParentColor 610
ParentCtl3D 611
ParentFont 611
ParentShowHint 612
Picture 612
PopupMenu 612
RowSize 617
RowSnap 618
ShowHint 618
TabOrder 620
TabStop 620
TControlBar events 622
TControlBar Members 591
TControlBar properties 593
Visible 621
TControlCollection class 663
about TControlCollection class 663
Add 664
AddControl 665
ControlItems 667
Controls 668
Create 664
GetControl 665
GetControlItem 665
GetItem 665
IndexOf 666
Items 668
Owner 666
RemoveControl 666
SetControl 666
SetItem 666
TControlCollection Members 663
TControlCollection methods 664
TControlCollection properties 667
Update 667
TControlItem class 668
about TControlItem class 668
AssignTo 670
Column 672
ColumnSpan 672
Control 672
Create 669
Destroy 669
GridPanel 672
InternalSetLocation 671
Pushed 672
Row 673
RowSpan 673
SetLocation 671
TControlItem Members 669
TControlItem methods 670
TControlItem properties 671
TControlScrollBar class 2009
about TControlScrollBar class 2009
Assign 2011
ButtonSize 2012
ChangeBiDiPosition 2011
Color 2013
Increment 2013
IsScrollBarVisible 2011
Kind 2015
Margin 2015
ParentColor 2016
Position 2017
Range 2018
ScrollPos 2020
Size 2021
Smooth 2021
Style 2022
TControlScrollBar Members 2010
TControlScrollBar methods 2010
TControlScrollBar properties 2012
ThumbSize 2022
Tracking 2022
Visible 2022
TCookie class 3304
about TCookie class 3304
AssignTo 3306

Create 3305
Domain 3307
Expires 3307
GetHeaderValue 3306
HeaderValue 3307
Name 3308
Path 3308
Secure 3309
TCookie Members 3305
TCookie methods 3306
TCookie properties 3306
Value 3309
TCookieCollection class 3309
about TCookieCollection class 3309
Add 3311
Create 3310
GetCookie 3311
Items 3312
SetCookie 3311
TCookieCollection Members 3310
TCookieCollection methods 3311
TCookieCollection properties 3312
WebResponse 3312
TCP enumeration member 4020
tcUpperCase enumeration member 1886
TCustomActionControlBar class 6721
about TCustomActionControlBar class 6721
CMControlChange 6722
Create 6722
CreateControl 6722
CreateControls 6723
CreatingControls 6723
GetControlClass 6723
TCustomActionControlBar Members 6721
TCustomActionControlBar methods 6722
TCustomActionControlBar properties 6723
TCustomActiveForm class 2024
about TCustomActiveForm class 2024
ActiveControl 2031
Anchors 2032
AutoScroll 2033
AutoSize 2033
AxBorderStyle 2033
BorderWidth 2034
Caption 2034
Color 2037
Constraints 2038
Create 2029
CreateParams 2029
Font 2038
Height 2039
HorzScrollBar 2040
KeyPreview 2041
OldCreateOrder 2045
OnActivate 2058
OnClick 2064
OnContextPopup 2066
OnCreate 2066
OnDblClick 2068
OnDeactivate 2068
OnDestroy 2069
OnDragDrop 2070
OnDragOver 2073
OnKeyDown 2076
OnKeyPress 2080
OnKeyUp 2081
OnMouseActivate 2083
OnMouseDown 2084
OnMouseEnter 2086
OnMouseLeave 2086
OnMouseMove 2086
OnMouseUp 2089
OnPaint 2091
Padding 2045
PixelsPerInch 2046
PopupMenu 2046
PrintScale 2051
Scaled 2052

ShowHint 2052
TCustomActiveForm events 2057
TCustomActiveForm Members 2024
TCustomActiveForm methods 2029
TCustomActiveForm properties 2030
VertScrollBar 2054
Visible 2054
WantChildKey 2030
Width 2055
TCustomButtonedEdit class 673
about TCustomButtonedEdit class 673
Create 674
DefaultHandler 675
Destroy 675
DoSetTextHint 676
GetEditButtonClass 676
Images 682
LeftButton 685
Notification 676
OnLeftButtonClick 692
OnRightButtonClick 695
RightButton 688
TCustomButtonedEdit events 691
TCustomButtonedEdit Members 674
TCustomButtonedEdit methods 675
TCustomButtonedEdit properties 682
UpdateEditMargins 677
WndProc 677
TCustomCategoryPanel class 698
about TCustomCategoryPanel class 698
CalcRectBounds 703
Collapse 703
Collapsed 714
CollapsedHotImageIndex 714
CollapsedImageIndex 715
CollapsedPressedImageIndex 715
Create 700
DefineProperties 703
Destroy 700
DrawCollapsedPanel 704
DrawHeader 704
DrawHeader method 704
DrawHeaderBackground 705
DrawHeaderCaption 705
DrawHeaderChevron 705
Expand 705
ExpandedHotImageIndex 716
ExpandedImageIndex 716
ExpandedPressedImageIndex 716
FPanelSurface 702
GetChildren 706
GetCollapsedHeight 706
GetTabOrderList 706
HeaderHeight 717
HeaderState 717
Loaded 707
OnCollapse 718
OnExpand 718
PanelGroup 717
ReadState 707
ResizeHeader 708
SetBounds 708
SetParent 708
TCustomCategoryPanel enumerations 701
TCustomCategoryPanel events 718
TCustomCategoryPanel fields 702
TCustomCategoryPanel Members 698
TCustomCategoryPanel methods 702
TCustomCategoryPanel properties 714
UpdateControlOriginalParentSize 709
UpdateHeader 709
WndProc 709
TCustomCategoryPanelGroup class 718
about TCustomCategoryPanelGroup class 718
ChevronAlignment 730
ChevronColor 730
ChevronHotColor 731
CollapseAll 722

Color 731
Create 720
CreatePanel 722
CreateWnd 723
Destroy 721
ExpandAll 723
GetCategoryPanelClass 723
GradientBaseColor 732
GradientColor 732
GradientDirection 733
HeaderAlignment 733
HeaderFont 733
HeaderHeight 734
HeaderImage 734
HeaderStyle 734
Images 735
Loaded 724
Notification 724
Panels 735
ReorderPanels 725
TCustomCategoryPanelGroup enumerations 721
TCustomCategoryPanelGroup Members 719
TCustomCategoryPanelGroup methods 722
TCustomCategoryPanelGroup properties 729
WndProc 725
TCustomColorBox class 735
about TCustomColorBox class 735
CloseUp 737
ColorNames 741
Colors 742
Create 737
CreateWnd 738
DefaultColorColor 742
DrawItem 738
KeyDown 738
KeyPress 739
Loaded 739
NoneColorColor 743
OnGetColors 744
PickCustomColor 740
PopulateList 740
Select 740
Selected 743
SetStyle 741
Style 744
TCustomColorBox events 744
TCustomColorBox Members 736
TCustomColorBox methods 737
TCustomColorBox properties 741
TCustomColorListBox class 745
about TCustomColorListBox class 745
ColorNames 750
Colors 750
Create 746
CreateWnd 746
DefaultColorColor 750
DrawItem 747
KeyDown 747
KeyPress 748
Loaded 748
NoneColorColor 750
OnGetColors 753
PickCustomColor 749
PopulateList 749
Selected 751
SetStyle 749
Style 753
TCustomColorListBox events 753
TCustomColorListBox Members 745
TCustomColorListBox methods 746
TCustomColorListBox properties 749
TCustomContentProducer class 3313
about TCustomContentProducer class 3313
Content 3314
ContentFromStream 3315
ContentFromString 3315
Dispatcher 3317
DispatcherComponent 3318

GetAppDispatcher 3316
Notification 3316
ProduceContent 3316
ProduceContentFromStream 3317
ProduceContentFromString 3317
SetAppDispatcher 3317
TCustomContentProducer Members 3314
TCustomContentProducer methods 3314
TCustomContentProducer properties 3317
TCustomControlBar class 754
about TCustomControlBar class 754
AlignControls 757
AutoDock 766
AutoDrag 766
AutoSize 767
BevelKind 767
CanAutoSize 757
CornerEdge 768
Create 755
CreateParams 758
Destroy 756
DoBandMove 758
DoBandPaint 758
DoBeginBandMove 759
DockOver 759
DockSite 768
DoEndBandMove 759
DoPaletteChange 760
DragControl 760
DrawingStyle 769
FlipChildren 761
GetControlInfo 761
GetPalette 762
GetSiteInfo 762
GradientDirection 769
GradientEndColor 769
GradientStartColor 769
HitTest 763
MouseDown 763
MouseMove 763
MouseUp 764
OnBandDrag 771
OnBandInfo 771
OnBandMove 772
OnBandPaint 772
OnBeginBandMove 773
OnEndBandMove 773
OnPaint 773
Paint 764
PaintControlFrame 765
Picture 770
RowSize 770
RowSnap 770
StickControls 765
TCustomControlBar events 771
TCustomControlBar Members 754
TCustomControlBar methods 756
TCustomControlBar properties 765
TCustomDataForm class 5260
about TCustomDataForm class 5260
GetHTMLFormTag 5262
ImplCanAddClass 5262
ImplContent 5262
TCustomDataForm Members 5260
TCustomDataForm methods 5261
TCustomDataGrid class 5263
about TCustomDataGrid class 5263
Create 5266
Destroy 5266
DisplayRows 5269
FormatTable 5266
HeadingAttributes 5269
ImplAddElements 5267
ImplCanAddClass 5267
ImplContent 5267
ImplIsMultipleRecordView 5268
RowAttributes 5270
SetHeadingAttributes 5268

SetRowAttributes 5268
SetTableAttributes 5269
TableAttributes 5270
TCustomDataGrid Members 5264
TCustomDataGrid methods 5266
TCustomDataGrid properties 5269
TCustomDataNavigator class 5271
about TCustomDataNavigator class 5271
Custom 5275
GetDefaultButtons 5273
ImplCanAddClass 5273
ImplContent 5273
ImplGetSubComponents 5274
Style 5275
StyleRule 5276
TCustomDataNavigator Members 5272
TCustomDataNavigator methods 5273
TCustomDataNavigator properties 5275
VisibleButtons 5274
TCustomDockForm class 2092
about TCustomDockForm class 2092
AutoScroll 2099
BorderStyle 2100
Create 2097
DoAddDockClient 2097
DoRemoveDockClient 2098
FormStyle 2100
GetSiteInfo 2098
Loaded 2099
PixelsPerInch 2103
TCustomDockForm Members 2092
TCustomDockForm methods 2097
TCustomDockForm properties 2099
TCustomDrawGrid class 3000
about TCustomDrawGrid class 3000
Canvas 3007
CellRect 3003
Col 3009
ColumnMoved 3004
ColWidths 3011
DrawCell 3004
EditorMode 3011
GetEditMask 3004
GetEditText 3005
GridHeight 3012
GridWidth 3012
LeftCol 3012
MouseToCell 3005
OnColumnMoved 3019
OnDrawCell 3020
OnGetEditMask 3021
OnGetEditText 3022
OnRowMoved 3023
OnSelectCell 3025
OnSetEditText 3026
OnTopLeftChanged 3027
Row 3013
RowHeights 3014
RowMoved 3005
SelectCell 3006
Selection 3015
SetEditText 3006
TabStops 3017
TCustomDrawGrid events 3019
TCustomDrawGrid Members 3000
TCustomDrawGrid methods 3003
TCustomDrawGrid properties 3007
TopLeftChanged 3007
TopRow 3017
TCustomFieldGroup class 5276
about TCustomFieldGroup class 5276
Create 5279
Custom 5281
Destroy 5279
GetLayoutAttributes 5280
ImplContent 5280
Style 5281
StyleRule 5282

TCustomFieldGroup Members 5277
TCustomFieldGroup methods 5280
TCustomFieldGroup properties 5281
TCustomFileRun class 17
about TCustomFileRun class 17
Browse 22
BrowseDlg 23
Create 19
Directory 23
ExecuteTarget 20
FileName 23
HandlesTarget 20
HInst 24
Notification 21
Operation 25
Parameters 25
ParentControl 26
SetupBrowseDialog 21
ShowCmd 26
TCustomFileRun Members 18
TCustomFileRun methods 19
TCustomFileRun properties 22
UpdateTarget 21
TCustomFlowPanel class 774
about TCustomFlowPanel class 774
AlignControls 776
AutoWrap 778
Create 775
Destroy 775
FlowStyle 778
GetChildren 776
GetControlIndex 777
SetControlIndex 777
TCustomFlowPanel Members 774
TCustomFlowPanel methods 776
TCustomFlowPanel properties 777
TCustomForm class 2103
about TCustomForm class 2103
Action 2144
Activate 2111
Active 2145
ActiveChanged 2112
ActiveControl 2145
ActiveMDIChild 2146
ActiveOleControl 2146
AfterConstruction 2112
AlignControls 2113
AlphaBlend 2147
AlphaBlendValue 2147
AutoScroll 2148
BeforeDestruction 2113
BeginAutoDrag 2113
BorderIcons 2148
BorderStyle 2149
Canvas 2150
Caption 2151
ChangeScale 2114
ClientHandle 2154
ClientHeight 2154
ClientWidth 2155
Close 2114
CloseModal 2115
CloseQuery 2115
Color 2157
Create 2108
CreateNew 2108
CreateParams 2116
CreateWindowHandle 2116
CreateWnd 2117
Ctl3D 2158
Deactivate 2117
DefaultHandler 2118
DefaultMonitor 2158
DefineProperties 2118
DefocusControl 2118
Designer 2159
Destroy 2109
DestroyHandle 2119

DestroyWindowHandle 2119
Dock 2120
DoClose 2120
DoCreate 2120
DoDestroy 2121
DoDock 2121
DoHide 2122
DoShow 2122
DropTarget 2159
FFormState 2110
FocusControl 2122
Font 2160
FormState 2161
FormStyle 2162
get_ActiveMDIChild 2123
get_MDIChildCount 2123
get_MDIChildren 2123
GetBorderIconStyles 2123
GetBorderStyles 2124
GetChildren 2124
GetClientRect 2124
GetFloating 2125
GetFormImage 2125
GetOwnerWindow 2127
GlassFrame 2164
HandleCreateException 2127
Height 2164
HelpFile 2165
Hide 2127
HorzScrollBar 2166
Icon 2167
InitializeNewForm 2128
IsShortCut 2128
KeyPreview 2167
Left 2171
Loaded 2129
MakeFullyVisible 2129
MDIChildCount 2172
MDIChildren 2173
Menu 2174
ModalResult 2176
Monitor 2177
MouseWheelHandler 2130
Notification 2130
ObjectMenuItem 2177
OldCreateOrder 2178
OleFormObject 2178
OnActivate 2196
OnCanResize 2202
OnClick 2203
OnClose 2205
OnCloseQuery 2206
OnCreate 2207
OnDblClick 2209
OnDeactivate 2209
OnDestroy 2209
OnDragDrop 2211
OnDragOver 2214
OnHelp 2217
OnHide 2217
OnKeyDown 2219
OnKeyPress 2223
OnKeyUp 2224
OnMouseActivate 2226
OnMouseDown 2227
OnMouseMove 2229
OnMouseUp 2231
OnPaint 2233
OnResize 2235
OnShortCut 2235
OnShow 2236
Paint 2131
PaintWindow 2131
PaletteChanged 2132
ParentFont 2179
PixelsPerInch 2179
PopupChildren 2179
PopupMenu 2180

PopupMode 2185
PopupParent 2185
Position 2185
Print 2132
PrintScale 2186
QueryInterface 2132
ReadState 2133
RecreateAsPopup 2133
Release 2133
RequestAlign 2134
Resizing 2134
Scaled 2187
ScreenSnap 2187
SendCancelMode 2134
set_PopupParent 2135
SetChildOrder 2135
SetFocus 2135
SetFocusedControl 2136
SetParent 2136
SetParentBiDiMode 2136
Show 2137
ShowModal 2139
SnapBuffer 2188
TCustomForm events 2195
TCustomForm fields 2109
TCustomForm Members 2104
TCustomForm methods 2110
TCustomForm properties 2142
TileMode 2188
Top 2189
TransparentColor 2190
TransparentColorValue 2191
UpdateActions 2140
UpdateWindowState 2141
ValidateRename 2141
VertScrollBar 2191
Visible 2192
VisibleChanging 2141
WantChildKey 2142
Width 2193
WindowMenu 2194
WindowState 2195
WndProc 2142
TCustomFrame class 2237
about TCustomFrame class 2237
Create 2238
CreateParams 2239
GetChildren 2239
PaintWindow 2240
SetParent 2240
TCustomFrame Members 2238
TCustomFrame methods 2239
TCustomGrid class 3027
about TCustomGrid class 3027
AdjustSize 3034
BeginColumnDrag 3034
BeginRowDrag 3034
BorderStyle 3060
BoxRect 3035
CalcDrawInfo 3035
CalcFixedInfo 3036
CalcSizingState 3036
CanEditAcceptKey 3036
CanEditModify 3037
CanEditShow 3037
CanGridAcceptKey 3037
CellRect 3038
ChangeGridOrientation 3038
CheckColumnDrag 3039
CheckRowDrag 3039
Col 3060
ColCount 3062
Color 3063
ColumnMoved 3039
ColWidths 3063
ColWidthsChanged 3040
Create 3030
CreateEditor 3040

CreateParams 3041
DefaultColWidth 3064
DefaultDrawing 3066
DefaultRowHeight 3066
DefineProperties 3041
DeleteColumn 3041
DeleteRow 3042
DesignOptionsBoost 3031
Destroy 3031
DoExit 3042
DoMouseWheelDown 3042
DoMouseWheelUp 3043
DrawCell 3043
EditorMode 3066
EndColumnDrag 3044
EndRowDrag 3044
FGridState 3031
FixedColor 3067
FixedCols 3067
FixedRows 3068
FocusCell 3044
FSaveCellExtents 3032
GetEditLimit 3045
GetEditMask 3045
GetEditStyle 3046
GetEditText 3046
GetGridHeight 3046
GetGridWidth 3047
GridHeight 3068
GridLineWidth 3069
GridWidth 3070
HideEdit 3047
HideEditor 3048
HitTest 3071
InplaceEditor 3071
InvalidateCell 3048
InvalidateCol 3048
InvalidateEditor 3049
InvalidateGrid 3049
InvalidateRow 3050
KeyDown 3050
KeyPress 3050
LeftCol 3072
MouseCoord 3051
MouseDown 3051
MouseMove 3052
MouseUp 3052
MoveColRow 3053
MoveColumn 3053
MoveRow 3053
Options 3072
Paint 3054
ParentColor 3073
Row 3074
RowCount 3075
RowHeights 3076
RowHeightsChanged 3054
RowMoved 3055
ScrollBars 3077
ScrollData 3055
SelectCell 3055
Selection 3077
SetEditText 3056
ShowEditor 3056
ShowEditorChar 3057
SizeChanged 3057
Sizing 3057
TabStop 3079
TabStops 3079
TCustomGrid fields 3031
TCustomGrid Members 3028
TCustomGrid methods 3032
TCustomGrid properties 3059
TimedScroll 3058
TopLeftChanged 3058
TopRow 3080
UpdateDesigner 3059
VirtualView 3032

VisibleColCount 3082
VisibleRowCount 3082
TCustomGridPanel class 781
about TCustomGridPanel class 781
AlignControls 784
AutoAddColumn 785
AutoAddRow 785
CellCount 788
CellIndexToCell 785
CellRect 788
CellSize 789
CellToCellIndex 785
ColumnCollection 789
ColumnSpanIndex 789
ControlCollection 789
Create 783
Destroy 783
ExpandStyle 790
GetChildren 785
IsColumnEmpty 786
IsRowEmpty 786
Loaded 786
Paint 787
RemoveEmptyAutoAddColumns 787
RemoveEmptyAutoAddRows 787
RowCollection 790
RowSpanIndex 790
TCustomGridPanel Members 782
TCustomGridPanel methods 784
TCustomGridPanel properties 788
UpdateControlOriginalParentSize 788
TCustomImageList class 4193
about TCustomImageList class 4193
Add 4197
AddIcon 4197
AddImage 4198
AddImages 4198
AddMasked 4199
AllocBy 4217
Assign 4199
AssignTo 4200
BkColor 4218
BlendColor 4218
Change 4200
Clear 4200
ColorDepth 4218
Count 4219
Create 4195
CreateSize 4195
DefineProperties 4201
Delete 4201
Destroy 4196
DoDraw 4201
Draw 4202
Draw method 4202
DrawingStyle 4223
DrawOverlay 4205
DrawOverlay method 4205
FileLoad 4206
GetBitmap 4206
GetHotSpot 4207
GetIcon 4207
GetIcon method 4207
GetImageBitmap 4208
GetImages 4208
GetInstRes 4209
GetInstRes method 4209
GetMaskBitmap 4210
GetResource 4210
Handle 4224
HandleAllocated 4211
HandleNeeded 4211
Height 4224
ImageType 4224
Initialize 4211
Insert 4212
InsertIcon 4212
InsertMasked 4212

Masked 4225
Move 4213
OnChange 4226
Overlay 4213
ReadData 4213
RegisterChanges 4214
Replace 4214
ReplaceIcon 4214
ReplaceMasked 4215
ResInstLoad 4215
ResourceLoad 4216
SetSize 4216
ShareImages 4225
TCustomImageList events 4226
TCustomImageList Members 4193
TCustomImageList methods 4196
TCustomImageList properties 4217
UnRegisterChanges 4216
Width 4225
WriteData 4217
TCustomInetXPageProducer class 5720
about TCustomInetXPageProducer class 5720
Content 5723
Create 5722
Destroy 5723
GetTemplateFileName 5725
GetTemplateStream 5725
HTMLDoc 5726
HTMLFile 5726
TCustomInetXPageProducer Members 5720
TCustomInetXPageProducer methods 5723
TCustomInetXPageProducer properties 5725
WebContent 5725
TCustomIniFile class 4258
about TCustomIniFile class 4258
Create 4262
DeleteKey 4263
EraseSection 4264
FileName 4291
ReadBinaryStream 4264
ReadBool 4265
ReadDate 4268
ReadDateTime 4268
ReadFloat 4269
ReadInteger 4269
ReadSection 4272
ReadSections 4273
ReadSections method 4273
ReadSectionValues 4274
ReadString 4274
ReadTime 4277
SectionExists 4278
TCustomIniFile Members 4262
TCustomIniFile methods 4263
TCustomIniFile properties 4290
UpdateFile 4278
ValueExists 4278
WriteBinaryStream 4279
WriteBool 4279
WriteDate 4282
WriteDateTime 4283
WriteFloat 4283
WriteInteger 4284
WriteString 4287
WriteTime 4290
TCustomLabeledEdit class 790
about TCustomLabeledEdit class 790
CMBidimodechanged 792
CMEnabledchanged 793
CMVisiblechanged 793
Create 791
EditLabel 796
LabelPosition 796
LabelSpacing 796
Notification 793
SetBounds 794
SetName 794
SetParent 795

SetupInternalLabel 795
TCustomLabeledEdit Members 791
TCustomLabeledEdit methods 792
TCustomLabeledEdit properties 795
TCustomLayoutGroup class 5282
about TCustomLayoutGroup class 5282
Create 5284
Custom 5286
Destroy 5284
DisplayColumns 5286
GetLayoutAttributes 5284
ImplCanAddClass 5285
ImplContent 5285
Style 5287
StyleRule 5287
TCustomLayoutGroup Members 5283
TCustomLayoutGroup methods 5284
TCustomLayoutGroup properties 5286
TCustomLinkLabel class 797
about TCustomLinkLabel class 797
Alignment 800
AutoSize 800
Create 798
CreateParams 799
CreateWnd 799
OnLinkClick 802
SetAutoSize 800
TCustomLinkLabel events 801
TCustomLinkLabel Members 797
TCustomLinkLabel methods 798
TCustomLinkLabel nested types 801
TCustomLinkLabel properties 800
TLinkAlignment 801
UseThemes 800
UseVisualStyle 801
TCustomListAction class 4535
about TCustomListAction class 4535
Active 4540
Count 4540
Create 4536
ExecuteTarget 4537
GetCount 4537
GetString 4538
HandlesTarget 4538
Images 4540
ItemIndex 4541
Loaded 4538
Loading 4541
OnGetItemCount 4542
OnItemSelected 4542
SetString 4539
Strings 4541
TCustomListAction events 4542
TCustomListAction Members 4536
TCustomListAction methods 4537
TCustomListAction properties 4539
TCustomMaskEdit class 4592
about TCustomMaskEdit class 4592
CheckCursor 4595
Clear 4595
Create 4594
EditCanModify 4596
EditMask 4604
EditText 4606
GetFirstEditChar 4596
GetLastEditChar 4597
GetMaxChars 4597
GetNextEditChar 4598
GetPriorEditChar 4598
GetSel 4599
GetTextLen 4599
IsMasked 4606
KeyDown 4599
KeyPress 4600
KeyUp 4600
MaskState 4607
MaxLength 4607
ReformatText 4601

Reset 4601
SetCursor 4602
SetSel 4602
TCustomMaskEdit Members 4593
TCustomMaskEdit methods 4594
TCustomMaskEdit properties 4604
Text 4608
Validate 4603
ValidateEdit 4603
ValidateError 4603
TCustomObjectBroker class 4787
about TCustomObjectBroker class 4787
GetComputerForGUID 4788
GetComputerForProgID 4789
GetPortForComputer 4789
SetConnectStatus 4790
TCustomObjectBroker Members 4788
TCustomObjectBroker methods 4788
TCustomOutline class 6258
about TCustomOutline class 6258
Add 6264
AddChild 6265
AddChildObject 6266
AddObject 6266
BadIndex 6267
BeginUpdate 6268
Canvas 6284
Clear 6268
Click 6268
Collapse 6269
Create 6262
DblClick 6269
DefineProperties 6269
Delete 6270
DeleteNode 6270
Destroy 6263
DrawCell 6271
EndUpdate 6271
Expand 6272
FullCollapse 6272
FullExpand 6272
GetDataItem 6273
GetItem 6273
GetNodeDisplayWidth 6274
GetTextItem 6274
GetVisibleNode 6274
Insert 6275
InsertObject 6275
InvalidateCell 6276
ItemCount 6286
ItemHeight 6286
Items 6287
ItemSeparator 6288
KeyDown 6277
KeyPress 6277
Lines 6288
Loaded 6278
LoadFromFile 6278
LoadFromFile method 6278
LoadFromStream 6279
LoadFromStream method 6279
MouseDown 6280
Move 6280
OnCollapse 6295
OnDrawItem 6295
OnExpand 6296
Options 6288
OutlineStyle 6289
PictureClosed 6290
PictureLeaf 6291
PictureMinus 6291
PictureOpen 6291
PicturePlus 6292
Row 6292
SaveToFile 6281
SaveToFile method 6280
SaveToStream 6281
SaveToStream method 6281

SelectedItem 6294
SetDisplayWidth 6282
SetGoodIndex 6282
SetLevel 6283
SetUpdateState 6283
Style 6294
TCustomOutline events 6294
TCustomOutline Members 6259
TCustomOutline methods 6263
TCustomOutline properties 6284
TCustomPacketWriter class 6460
about TCustomPacketWriter class 6460
AddAttribute 6463
Check 6463
Create 6461
Destroy 6461
DSWriter 6464
FBuffer 6462
SetDSWriter 6463
TCustomPacketWriter fields 6462
TCustomPacketWriter Members 6461
TCustomPacketWriter methods 6462
TCustomPacketWriter properties 6463
TCustomPageProducer class 3473
about TCustomPageProducer class 3473
Create 3474
Destroy 3475
GetTemplateFileName 3475
GetTemplateStream 3476
HandleTag 3476
HTMLDoc 3476
HTMLFile 3477
TCustomPageProducer Members 3473
TCustomPageProducer methods 3475
TCustomPageProducer properties 3476
TCustomPanel class 802
about TCustomPanel class 802
AdjustClientRect 804
Alignment 807
BevelInner 807
BevelOuter 808
BevelWidth 809
BorderStyle 810
BorderWidth 811
CanAutoSize 804
Color 812
Create 803
CreateParams 805
FullRepaint 813
GetControlsAlignment 805
Locked 813
Paint 806
ParentBackground 813
ParentColor 814
SetParentBackground 806
ShowCaption 814
TCustomPanel Members 803
TCustomPanel methods 804
TCustomPanel properties 806
VerticalAlignment 814
TCustomProvider class 6464
about TCustomProvider class 6464
ActiveUpdateException 6482
AfterApplyUpdates 6483
AfterExecute 6484
AfterGetParams 6485
AfterGetRecords 6485
AfterRowRequest 6486
ApplyUpdates 6468
ApplyUpdates method 6468
BeforeApplyUpdates 6487
BeforeExecute 6487
BeforeGetParams 6488
BeforeGetRecords 6488
BeforeRowRequest 6490
Create 6466
Data 6482
DataRequest 6468

Destroy 6467
DoAfterApplyUpdates 6469
DoAfterExecute 6469
DoAfterGetParams 6470
DoAfterGetRecords 6470
DoAfterRowRequest 6471
DoApplyUpdates 6471
DoBeforeApplyUpdates 6471
DoBeforeExecute 6472
DoBeforeGetParams 6472
DoBeforeGetRecords 6473
DoBeforeRowRequest 6474
DoGetRecords 6474
Execute 6474
Exported 6482
GetParams 6475
GetRecords 6476
GetRecords method 6476
InternalApplyUpdates 6477
InternalExecute 6478
InternalGetParams 6478
InternalGetRecords 6479
InternalRowRequest 6480
OnDataRequest 6490
OnValidate 6491
OwnerData 6483
RowRequest 6480
SetActiveUpdateException 6481
TCustomProvider events 6483
TCustomProvider Members 6465
TCustomProvider methods 6467
TCustomProvider properties 6481
TCustomQueryButtons class 5288
about TCustomQueryButtons class 5288
Custom 5291
GetDefaultButtons 5289
ImplAddElements 5290
ImplCanAddClass 5290
ImplContent 5290
Style 5292
StyleRule 5292
TCustomQueryButtons Members 5288
TCustomQueryButtons methods 5289
TCustomQueryButtons properties 5291
VisibleButtons 5291
TCustomQueryForm class 5293
about TCustomQueryForm class 5293
Action 5298
AssignStrings 5295
AssignStringsCallback 5295
Create 5295
GetHTMLFormTag 5296
ImplAddElements 5296
ImplCanAddClass 5296
ImplContent 5297
Method 5298
MethodString 5297
TCustomQueryForm Members 5294
TCustomQueryForm methods 5295
TCustomQueryForm properties 5297
TCustomRadioGroup class 815
about TCustomRadioGroup class 815
Buttons 819
CanModify 817
Columns 820
Create 816
Destroy 816
FlipChildren 817
GetChildren 818
ItemIndex 820
Items 821
Loaded 818
ReadState 819
TCustomRadioGroup Members 815
TCustomRadioGroup methods 817
TCustomRadioGroup properties 819
WordWrap 822
TCustomResolver class 6491

about TCustomResolver class 6491
ApplyUpdates 6494
BeginUpdate 6494
Create 6493
Destroy 6493
DoDelete 6495
DoInsert 6495
DoUpdate 6496
EndUpdate 6496
FreeTreeData 6497
HandleUpdateError 6497
InitializeConflictBuffer 6497
InitKeyFields 6498
InitTreeData 6498
InternalBeforeResolve 6498
InternalUpdateRecord 6499
LogUpdateError 6499
LogUpdateRecord 6500
Provider 6501
RowRequest 6500
TCustomResolver Members 6492
TCustomResolver methods 6493
TCustomResolver properties 6500
UpdateTree 6501
TCustomRibbon class 6724
about TCustomRibbon class 6724
ActionManager 6759
Active 6759
ActivePage 6760
AddRecentItem 6735
AddTab 6736
AllowKeyboardNavigation 6760
ApplicationMenu 6760
BeginUpdate 6736
CheckRibbonSize 6736
ClearRecentItems 6736
CMBidimodechanged 6736
CMDesignHitTest 6737
CMFontchanged 6737
CMMouseenter 6737
CMMouseleave 6737
CMTextchanged 6738
CMThemeChanged 6738
ColorMap 6761
ContextMenuPopupHandler 6738
Create 6728
CreateContextMenu 6738
cRibbonFirstTabOffSet 6730
cRibbonGroupCaptionHeight 6730
cRibbonGroupHeight 6730
cRibbonHeight 6730
cRibbonHideHeight 6730
cRibbonHideWidth 6731
cRibbonMinimizedHeight 6731
cRibbonMinimumCaptionWidth 6731
cRibbonPageHeight 6731
cRibbonQATHeight 6732
cRibbonQuickAccessToolbarLeft 6732
cRibbonQuickAccessToolbarTop 6732
cRibbonTabHeight 6732
cRibbonTabScrollButtonWidth 6732
cRibbonTabSpacing 6733
cRibbonUnthemedCaptionHeight 6733
CustomFrameSupported 6738
CustomFrameWndProc 6739
DblClick 6739
DefineProperties 6739
DeleteTab 6740
Destroy 6728
DisplayKeyTips 6740
DoAddToQuickAccessToolbar 6740
DoChangeQuickAccessToolbarPosition 6740
DoContextPopup 6741
DocumentName 6761
DoCustomizeQuickAccessToolbar 6741
DoGroupStyleChanged 6741
DoHelpButtonClick 6741
DoMinimizeRibbon 6742

DoMouseWheelDown 6742
DoMouseWheelEx 6742
DoMouseWheelUp 6743
DoRecentItemClick 6743
DoScrollTabsLeft 6743
DoScrollTabsRight 6743
DoTabChange 6744
DrawCaption 6744
DrawQuickAccessToolbar 6744
DrawSystemButtons 6744
DrawTab 6745
EndUpdate 6745
GetContextMenuOptions 6745
GetHelpButtonRect 6745
GetIcon 6745
GetRibbonHeight 6746
GetRibbonMetric 6746
GetTabAt 6746
GetTabRect 6746
GetTabWidth 6747
HasApplicationMenu 6747
HasQuickAccessToolbar 6747
HideRibbon 6761
HideTabs 6762
HotTabIndex 6762
Idle 6747
InvalidateHeaderArea 6747
InvalidateScrollButtons 6748
InvalidateSystemMenuArea 6748
InvalidateTabArea 6748
KeyDown 6749
KeyTipsVisible 6762
Loaded 6749
Menu 6762
Minimized 6763
MouseDown 6750
MouseInSystemMenu 6750
MouseMove 6750
MouseUp 6751
MoveGroup 6751
Notification 6752
OnHelpButtonClick 6766
OnLeftTabScrollButtonClick 6766
OnRecentItemClick 6766
OnRightTabScrollButtonClick 6767
OnTabChange 6767
Paint 6752
QuickAccessToolbar 6763
RebuildRibbonMetrics 6753
RemoveAllTabs 6753
RemoveRecentItem 6753
RemoveTab 6753
Resize 6754
ScaleFactor 6763
ScreenTips 6763
SetBounds 6754
SetParent 6754
ShowHelpButton 6764
Style 6764
TabHandledKeyTips 6755
TabHideKeyTips 6755
TabIndex 6764
Tabs 6764
TabShowKeyTips 6756
TCustomRibbon constants 6729
TCustomRibbon events 6766
TCustomRibbon Members 6725
TCustomRibbon methods 6733
TCustomRibbon nested types 6765
TCustomRibbon properties 6758
TTabChangeEvent 6765
UM_CHECKSIZE 6733
UM_DISPLAYKEYTIPS 6733
UMCheckResize 6756
UMDisplayKeyTips 6756
UpdateQATPosition 6757
UpdateSystemMenuInfo 6757
UseCustomFrame 6765

ValidTab 6757
WMEraseBkgnd 6757
WMGetDlgCode 6758
WMKillFocus 6758
WMSettingChange 6758
WMWindowPosChanged 6758
TCustomRibbonColorMap class 6767
about TCustomRibbonColorMap class 6767
ActiveTabFontColor 6768
BevelMinimizedInner 6768
BevelMinimizedOuter 6769
CaptionFontColor 6769
ControlColor 6769
ControlFrameColor 6769
ControlInactiveColor 6769
DocumentFontColor 6770
GroupFontColor 6770
InactiveTabFontColor 6770
QuickAccessToolbarMoreBorderColor 6770
QuickAccessToolbarMoreColor 6771
TCustomRibbonColorMap Members 6767
TCustomRibbonColorMap properties 6768
TCustomRibbonComboBox class 6947
about TCustomRibbonComboBox class 6947
BeginAutoDrag 6949
CMMouseenter 6949
CMMouseleave 6950
Color 6955
Create 6948
Destroy 6948
DragDrop 6950
DragOver 6950
Items 6956
KeyDown 6951
KeyPress 6951
MouseInControl 6956
MouseMove 6952
Notification 6952
SetBounds 6953
SetParent 6953
TCustomRibbonComboBox Members 6947
TCustomRibbonComboBox methods 6948
TCustomRibbonComboBox properties 6954
WMKillFocus 6954
WMSetFocus 6954
TCustomRibbonGalleryBar class 7352
about TCustomRibbonGalleryBar class 7352
Align 7354
AlignControls 7353
Create 7353
CreateControl 7354
DoDropCategory 7354
ItemsPerRow 7358
ShowItemCaptions 7358
TCustomRibbonGalleryBar Members 7352
TCustomRibbonGalleryBar methods 7353
TCustomRibbonGalleryBar properties 7354
TCustomRibbonGalleryPopupBar class 7358
about TCustomRibbonGalleryPopupBar class 7358
AlignControls 7359
CreateControls 7360
DoGetControlClass 7360
SetColorMap 7360
TCustomRibbonGalleryPopupBar Members 7359
TCustomRibbonGalleryPopupBar methods 7359
TCustomRibbonGroup class 6771
about TCustomRibbonGroup class 6771
AlignControls 6775
AlignGroupControls 6775
Assign 6775
AutoSizingChanged 6776
BarHandleKeyTips 6776
BarHideKeyTips 6776
BarShowKeyTips 6777
CanCollapse 6786
Clone 6777
CMFontchanged 6777
CMItemDropPoint 6777

CMMouseenter 6778
CMMouseleave 6778
CMTextchanged 6778
CollapsedControl 6786
Create 6773
CreateControl 6778
CreateControls 6779
Destroy 6773
DialogAction 6786
DockDrop 6779
DoDropItem 6779
DoEndDrag 6779
DragDrop 6779
DrawShowDialogButton 6780
GetCaptionHeight 6780
GetGroupHeight 6780
GetKeyTip 6780
GroupAlign 6787
GroupIndex 6787
GroupVariant 6787
GroupWidth 6788
HandleKeyTip 6781
HasGallery 6788
HideControls 6781
HideKeyTip 6781
Items 6788
KeyTip 6788
KeyTipClick 6782
MouseDown 6782
MouseMove 6782
MouseUp 6783
Notification 6783
Page 6789
Paint 6783
RequiredWidth 6789
Resize 6783
Rows 6789
SetAutoSizing 6783
SetBounds 6784
SetKeyTip 6784
SetParent 6784
SetupDropDownBtn 6785
ShowControls 6785
ShowKeyTip 6785
StyleChanged 6785
TCustomRibbonGroup enumerations 6773
TCustomRibbonGroup Members 6771
TCustomRibbonGroup methods 6774
TCustomRibbonGroup nested types 6790
TCustomRibbonGroup properties 6786
TRowPosition 6790
TCustomRibbonPage class 6790
about TCustomRibbonPage class 6790
AddGroup 6794
AlignGroups 6794
CMControlListChanging 6795
CMTextchanged 6795
CollapseGroup 6795
CollapseGroupCount 6803
Create 6792
CreateScrollButton 6795
cRibbonGroupLeftOffset 6793
cRibbonGroupSpacing 6793
Destroy 6792
DoScrollButtonClick 6795
FindGroup 6796
FindGroupAt 6796
GetGroupPos 6796
GetNextVisibleGroup 6796
GetPageHeight 6797
GetPageTop 6797
GroupCount 6803
GroupOffset 6803
Groups 6803
HandleKeyTips 6797
HideKeyTips 6797
Index 6803
Invalidate 6797

KeyTip 6804
Loaded 6798
Notification 6798
Paint 6799
RequestAlign 6799
Resize 6800
Ribbon 6804
RibbonStyle 6800
SetBounds 6800
SetParent 6801
ShowKeyTips 6802
TCustomRibbonPage constants 6793
TCustomRibbonPage Members 6790
TCustomRibbonPage methods 6793
TCustomRibbonPage properties 6802
UpdateActionManager 6802
UpdateScrollButtons 6802
TCustomRibbonQuickAccessPopupToolbar class 6804
about TCustomRibbonQuickAccessPopupToolbar class 6804
Create 6805
CreateControl 6806
CreateWnd 6806
DrawBackground 6806
GetControlClass 6806
NCPaint 6807
TCustomRibbonQuickAccessPopupToolbar Members 6804
TCustomRibbonQuickAccessPopupToolbar methods 6805
ValidOrientations 6807
TCustomRibbonQuickAccessToolbar class 6807
about TCustomRibbonQuickAccessToolbar class 6807
BarHandleKeyTips 6810
BarHideKeyTips 6810
BarShowKeyTips 6811
CMFontchanged 6811
Create 6809
CreateControl 6811
CreateControls 6811
Customize 6812
DoContextPopup 6812
DoDropActions 6812
DoDropCategory 6813
DoDropItem 6813
DoMoreCommands 6813
DoScrollBtnClick 6813
DoUpdateQATWidth 6813
DragDrop 6814
DrawBackground 6814
GetBarHeight 6814
GetBarWidth 6815
GetScrollBtnClass 6815
Items 6818
Notification 6815
OnMoreCommands 6818
RequestAlign 6815
Ribbon 6818
ScrollMode 6818
SetParent 6816
SetupDropDownBtn 6816
ShowBorder 6819
ShowCustomizeMenu 6817
ShowMoreControls 6817
TCustomRibbonQuickAccessToolbar Members 6808
TCustomRibbonQuickAccessToolbar methods 6809
TCustomRibbonQuickAccessToolbar nested types 6819
TCustomRibbonQuickAccessToolbar properties 6817
TMoreCommandsEvent 6819
UpdateQATWidth 6817
TCustomRibbonSkin class 7372
about TCustomRibbonSkin class 7372
Bitmap 7376
Create 7373
Destroy 7373
DoLoadByFile 7374
DoLoadByResource 7374
GetLoadType 7375
GetLoadTypeValue 7375
GetResourceHandle 7375
LoadSkin 7375

TCustomRibbonSkin enumerations 7373
TCustomRibbonSkin Members 7372
TCustomRibbonSkin methods 7374
TCustomRibbonSkin properties 7375
TCustomStaticListAction class 4543
about TCustomStaticListAction class 4543
Clients 4547
Count 4547
Create 4544
Destroy 4544
GetCount 4545
GetItem 4545
GetItemClass 4545
GetString 4546
Items 4547
OnGetItem 4547
SetString 4546
TCustomStaticListAction events 4547
TCustomStaticListAction Members 4543
TCustomStaticListAction methods 4545
TCustomStaticListAction properties 4546
TCustomTabData class 6819
about TCustomTabData class 6819
Add 6821
AvailableSpace 6822
CalculateTabWidths 6821
Clear 6821
Create 6820
Destroy 6821
FirstVisibleTabIndex 6822
FullWidth 6822
ItemCount 6823
LastVisibleTabIndex 6823
LeftTabScrollButtonRect 6823
NewWidth 6823
Owner 6824
RightTabScrollButtonRect 6824
ScrollButtons 6824
SeparatorOpacity 6824
TabSeparatorState 6824
TabUpdateRect 6825
TCustomTabData Members 6820
TCustomTabData methods 6821
TCustomTabData properties 6822
TCustomTrayIcon class 822
about TCustomTrayIcon class 822
Animate 830
AnimateInterval 830
BalloonFlags 831
BalloonHint 831
BalloonTimeout 831
BalloonTitle 831
Create 824
Data 832
Destroy 824
DoOnAnimate 825
GetAnimateInterval 825
GetBalloonTimeout 825
Hint 832
Icon 832
IconIndex 832
Icons 833
Notification 826
OnAnimate 834
OnBalloonClick 834
OnClick 834
OnDblClick 834
OnMouseDown 835
OnMouseMove 835
OnMouseUp 835
PopupMenu 833
Refresh 826
Refresh method 826
SetAnimate 827
SetAnimateInterval 827
SetBalloonHint 827
SetBalloonTimeout 827
SetBalloonTitle 828

SetDefaultIcon 828
SetHint 828
SetIcon 828
SetIconIndex 828
SetIconList 829
SetVisible 829
ShowBalloonHint 829
TCustomTrayIcon events 833
TCustomTrayIcon Members 823
TCustomTrayIcon methods 824
TCustomTrayIcon properties 830
Visible 833
WindowProc 829
TCustomVirtualListAction class 4548
about TCustomVirtualListAction class 4548
Count 4550
GetItem 4549
GetString 4549
OnGetItem 4550
TCustomVirtualListAction events 4550
TCustomVirtualListAction Members 4548
TCustomVirtualListAction methods 4549
TCustomVirtualListAction properties 4550
TCustomWebDispatcher class 3318
about TCustomWebDispatcher class 3318
Access_Request 3322
Access_Response 3322
Action 3327
ActionByName 3322
Actions 3327
AfterDispatch 3329
BeforeDispatch 3329
Create 3320
Destroy 3321
DispatchAction 3323
DoAfterDispatch 3324
DoBeforeDispatch 3324
FinishContext 3325
HandleException 3325
HandleRequest 3325
InitContext 3326
InitModule 3326
Notification 3326
OnException 3330
Request 3327
Response 3328
TCustomWebDispatcher events 3328
TCustomWebDispatcher Members 3319
TCustomWebDispatcher methods 3321
TCustomWebDispatcher properties 3326
TDatabaseInfo class 3944
about TDatabaseInfo class 3944
Create 3945
DbName 3945
Destroy 3945
NoOfAttachments 3945
NoOfDatabases 3946
TDatabaseInfo fields 3945
TDatabaseInfo Members 3944
TDataContext class 4354
about TDataContext class 4354
AddDynArrayToClear 4358
AddObjectToDestroy 4359
AddStrToClear 4359
AddUStrToClear 4360
AddVariantToClear 4360
AddWStrToClear 4361
AllocData 4361
Create 4355
Data 4356
DataOffset 4356
DataP 4357
Destroy 4355
DynArrayToClear 4357
FObjsToDestroy 4357
GetDataPointer 4362
RemoveObjectToDestroy 4362
SetDataPointer 4362

StrToClear 4357
TDataContext fields 4356
TDataContext Members 4355
TDataContext methods 4358
UStrToClear 4357
VarToClear 4358
WStrToClear 4358
TDataForm class 5298
about TDataForm class 5298
Custom 5300
Style 5301
StyleRule 5301
TDataForm Members 5299
TDataForm properties 5300
TDataGrid class 5302
about TDataGrid class 5302
DisplayRows 5305
HeadingAttributes 5305
RowAttributes 5306
TableAttributes 5306
TDataGrid Members 5303
TDataGrid properties 5305
TDataNavigator class 5307
about TDataNavigator class 5307
Custom 5309
Style 5309
StyleRule 5310
TDataNavigator Members 5307
TDataNavigator properties 5309
XMLComponent 5310
TDataPacketWriter class 6501
about TDataPacketWriter class 6501
AddColumn 6504
AddConstraints 6504
AddDataSetAttributes 6504
AddExtraFieldProps 6504
AddFieldLinks 6505
AddIndexDefs 6505
Constraints 6509
Destroy 6503
FreeInfoRecords 6505
GetDataPacket 6505
GetFieldIdx 6506
InitPutProcs 6506
OnGetParams 6510
Options 6509
PacketOptions 6510
PutADTField 6506
PutArrayField 6506
PutBlobField 6506
PutCalcField 6507
PutDataSetField 6507
PutField 6507
PutStringField 6507
PutVarBytesField 6508
PutWideStringField 6508
RefreshPutProcs 6508
Reset 6508
TDataPacketWriter events 6510
TDataPacketWriter Members 6502
TDataPacketWriter methods 6503
TDataPacketWriter properties 6509
WriteDataSet 6508
WriteMetaData 6509
TDataSetButton class 5311
about TDataSetButton class 5311
AddElements 5314
Caption 5316
Custom 5316
DefaultCaption 5313
GetCaption 5314
GetMethodName 5314
GetSubComponents 5314
ImplAddElements 5315
ImplContent 5315
Style 5316
StyleRule 5317
TDataSetButton fields 5313

TDataSetButton Members 5312
TDataSetButton methods 5313
TDataSetButton properties 5315
XMLComponent 5317
XMLMethodName 5313
XMLUseParent 5318
TDataSetProvider class 6510
about TDataSetProvider class 6510
ActiveUpdateException 6524
AfterApplyUpdates 6529
AfterExecute 6530
AfterGetParams 6531
AfterGetRecords 6531
AfterRowRequest 6532
AfterUpdateRecord 6533
BeforeApplyUpdates 6533
BeforeExecute 6534
BeforeGetParams 6534
BeforeGetRecords 6535
BeforeRowRequest 6536
BeforeUpdateRecord 6537
CheckDataSet 6514
Constraints 6525
Create 6513
CreateDataPacket 6515
CreateResolver 6515
DataSet 6525
Destroy 6513
DoBeforeExecute 6516
DoBeforeGetRecords 6516
DoGetProviderAttributes 6517
DoGetTableName 6518
Exported 6526
FetchDetails 6518
FindRecord 6518
GetDataSetFromDelta 6519
InternalApplyUpdates 6519
InternalExecute 6520
InternalGetParams 6520
InternalGetRecords 6521
InternalRowRequest 6522
LocateRecord 6522
Notification 6522
OnDataRequest 6537
OnGetData 6537
OnGetDataSetProperties 6538
OnGetTableName 6539
OnUpdateData 6539
OnUpdateError 6539
OnValidate 6540
Options 6526
OwnerData 6527
Params 6527
Reset 6523
ResolveToDataSet 6528
SetCommandText 6523
SetParams 6523
TDataSetProvider events 6529
TDataSetProvider Members 6511
TDataSetProvider methods 6514
TDataSetProvider properties 6524
TransactionStarted 6528
UpdateMode 6528
UpdateRecord 6523
TDataSetResolver class 6540
about TDataSetResolver class 6540
BeginUpdate 6542
Create 6541
DoDelete 6543
DoInsert 6543
DoUpdate 6543
EndUpdate 6544
InitializeConflictBuffer 6544
InternalBeforeResolve 6545
Provider 6545
TDataSetResolver Members 6541
TDataSetResolver methods 6542
TDataSetResolver properties 6545

TDCOMConnection class 4790
about TDCOMConnection class 4790
ComputerName 4793
Create 4792
DoConnect 4793
ObjectBroker 4794
TDCOMConnection Members 4791
TDCOMConnection methods 4792
TDCOMConnection properties 4793
TDefaultWebAppServices class 3330
about TDefaultWebAppServices class 3330
FindRequestHandler 3331
FinishContext 3332
GetExceptionHandler 3332
GetRequestHandler 3332
HandleRequest 3332
InitContext 3333
InvokeDispatcher 3334
Request 3334
RequestHandler 3334
Response 3334
TDefaultWebAppServices Members 3331
TDefaultWebAppServices methods 3331
TDefaultWebAppServices properties 3334
WebModules 3335
TDeleteButton class 5318
about TDeleteButton class 5318
Create 5320
TDeleteButton Members 5319
TDirectoryListBox class 1589
about TDirectoryListBox class 1589
Align 1602
Anchors 1605
AutoComplete 1606
BevelEdges 1606
BevelInner 1606
BevelKind 1607
BevelOuter 1607
BuildList 1595
CaseSensitive 1608
Change 1595
ClosedBMP 1593
Color 1608
Columns 1609
Constraints 1610
Create 1592
CreateWnd 1595
Ctl3D 1610
CurrentBMP 1593
DblClick 1596
Destroy 1593
Directory 1611
DirLabel 1612
DisplayCase 1596
DoubleBuffered 1613
DragCursor 1615
DragMode 1615
DrawItem 1597
Drive 1617
Enabled 1618
FDirectory 1594
FileCompareText 1597
FileList 1618
FOnChange 1594
Font 1621
GetItemPath 1597
ImeMode 1622
ImeName 1622
IntegralHeight 1623
ItemHeight 1623
KeyPress 1598
Notification 1598
OnChange 1636
OnClick 1638
OnContextPopup 1639
OnDblClick 1640
OnDragDrop 1640
OnDragOver 1643

OnEndDrag 1647
OnEnter 1648
OnExit 1650
OnKeyDown 1653
OnKeyPress 1657
OnKeyUp 1658
OnMouseActivate 1659
OnMouseDown 1660
OnMouseEnter 1663
OnMouseLeave 1663
OnMouseMove 1663
OnMouseUp 1665
OnStartDrag 1668
OpenCurrent 1598
OpenedBMP 1594
ParentColor 1624
ParentCtl3D 1624
ParentDoubleBuffered 1625
ParentFont 1625
ParentShowHint 1625
PopupMenu 1626
PreserveCase 1631
ReadBitmaps 1599
ReadDirectoryNames 1599
ShowHint 1631
TabOrder 1633
TabStop 1633
TDirectoryListBox events 1635
TDirectoryListBox fields 1593
TDirectoryListBox Members 1590
TDirectoryListBox methods 1594
TDirectoryListBox properties 1601
Update 1600
Visible 1634
TDispatchAppServer class 4794
about TDispatchAppServer class 4794
Create 4795
SafeCallException 4795
TDispatchAppServer Members 4795
TDispatchAppServer methods 4795
TDispatchConnection class 4796
about TDispatchConnection class 4796
AfterConnect 4808
AfterDisconnect 4808
BeforeConnect 4809
BeforeDisconnect 4809
Connected 4802
Create 4797
DoDisconnect 4798
GetConnected 4798
GetServer 4799
GetServerCLSID 4799
GetServerList 4800
LoginPrompt 4802
Notification 4800
ObjectBroker 4806
OnGetUsername 4810
OnLogin 4810
ServerGUID 4807
ServerName 4807
SetConnected 4800
TDispatchConnection events 4808
TDispatchConnection Members 4797
TDispatchConnection methods 4798
TDispatchConnection properties 4801
TDrawGrid class 3083
about TDrawGrid class 3083
Align 3088
Anchors 3092
BevelEdges 3092
BevelInner 3093
BevelKind 3093
BevelOuter 3094
BevelWidth 3094
BiDiMode 3095
BorderStyle 3095
ColCount 3095
Color 3096

Constraints 3097
Ctl3D 3098
DefaultColWidth 3098
DefaultDrawing 3100
DefaultRowHeight 3100
DoubleBuffered 3101
DragCursor 3102
DragKind 3103
DragMode 3103
Enabled 3105
FixedColor 3105
FixedCols 3106
FixedRows 3106
Font 3107
GridLineWidth 3108
OnClick 3125
OnColumnMoved 3127
OnContextPopup 3127
OnDblClick 3128
OnDragDrop 3128
OnDragOver 3131
OnDrawCell 3134
OnEndDock 3135
OnEndDrag 3135
OnEnter 3136
OnExit 3139
OnGetEditMask 3142
OnGetEditText 3142
OnKeyDown 3142
OnKeyPress 3146
OnKeyUp 3147
OnMouseActivate 3149
OnMouseDown 3150
OnMouseEnter 3152
OnMouseLeave 3152
OnMouseMove 3153
OnMouseUp 3155
OnMouseWheelDown 3157
OnMouseWheelUp 3157
OnRowMoved 3158
OnSelectCell 3158
OnSetEditText 3159
OnStartDock 3159
OnStartDrag 3160
OnTopLeftChanged 3160
Options 3109
ParentBiDiMode 3111
ParentColor 3111
ParentCtl3D 3112
ParentDoubleBuffered 3112
ParentFont 3112
ParentShowHint 3113
PopupMenu 3113
RowCount 3118
ScrollBars 3119
ShowHint 3120
TabOrder 3121
TDrawGrid events 3124
TDrawGrid Members 3083
TDrawGrid properties 3087
Visible 3122
VisibleColCount 3123
VisibleRowCount 3124
TDriveComboBox class 1668
about TDriveComboBox class 1668
Align 1676
Anchors 1680
AutoComplete 1680
AutoDropDown 1681
BevelEdges 1682
BevelInner 1682
BevelKind 1683
BevelOuter 1683
BuildList 1673
CDROMBMP 1672
Click 1673
Color 1684
Constraints 1684

Create 1671
CreateWnd 1674
Ctl3D 1685
Destroy 1671
DirList 1685
DoubleBuffered 1687
DragCursor 1688
DragMode 1689
DrawItem 1674
Drive 1690
Enabled 1692
FixedBMP 1672
FloppyBMP 1672
Font 1692
ImeMode 1693
ImeName 1694
NetworkBMP 1672
Notification 1675
OnChange 1708
OnClick 1710
OnContextPopup 1712
OnDblClick 1712
OnDragDrop 1713
OnDragOver 1716
OnDropDown 1719
OnEndDrag 1720
OnEnter 1721
OnExit 1724
OnKeyDown 1727
OnKeyPress 1730
OnKeyUp 1731
OnMouseEnter 1733
OnMouseLeave 1733
OnStartDrag 1734
ParentColor 1694
ParentCtl3D 1695
ParentDoubleBuffered 1695
ParentFont 1695
ParentShowHint 1696
PopupMenu 1696
RAMBMP 1673
ShowHint 1701
TabOrder 1703
TabStop 1704
TDriveComboBox events 1707
TDriveComboBox fields 1672
TDriveComboBox Members 1669
TDriveComboBox methods 1673
TDriveComboBox properties 1675
Text 1704
TextCase 1705
Visible 1706
TEditButton class 835
about TEditButton class 835
Create 839
Destroy 840
DisabledImageIndex 842
DropDownMenu 846
EditControl 846
Enabled 846
GetOwner 841
Glyph 849
HotImageIndex 850
ImageIndex 853
Images 856
Position 856
PressedImageIndex 856
TEditButton enumerations 840
TEditButton Members 839
TEditButton methods 841
TEditButton properties 842
UpdateBounds 842
Visible 857
TEnumPropDesc class 6067
about TEnumPropDesc class 6067
Create 6068
Destroy 6069
GetStrings 6069

StringToValue 6070
TEnumPropDesc Members 6068
TEnumPropDesc methods 6069
ValueToString 6070
TEventDispatch class 6070
_AddRef 6072
_Release 6072
about TEventDispatch class 6070
Control 6073
Create 6071
GetIDsOfNames 6072
GetTypeInfo 6072
GetTypeInfoCount 6073
Invoke 6073
QueryInterface 6073
TEventDispatch Members 6071
TEventDispatch methods 6071
TEventDispatch properties 6073
tfBlob enumeration member 3555
tfBottom enumeration member 2873
tfBytes enumeration member 5825
tfCalcRect enumeration member 2873
tfCenter enumeration member 2873
tfConnect enumeration member 3555
tfEditControl enumeration member 2873
tfEndEllipsis enumeration member 2873
tfError enumeration member 3555
tfExpandTabs enumeration member 2873
tfExternalLeading enumeration member 2873
tfFrames enumeration member 5825
tfHMS enumeration member 5825
TFieldGroup class 5320
about TFieldGroup class 5320
Custom 5324
ImplAddElements 5323
ImplCanAddClass 5324
Style 5325
StyleRule 5325
TFieldGroup Members 5321
TFieldGroup methods 5323
TFieldGroup properties 5324
TFieldRadioGroup class 5326
about TFieldRadioGroup class 5326
Caption 5329
CaptionAttributes 5329
CaptionPosition 5330
Custom 5330
DataSet 5330
DisplayColumns 5331
DisplayWidth 5331
FieldName 5332
Items 5332
ItemsField 5333
ReadOnly 5333
Style 5333
StyleRule 5334
TabIndex 5334
TFieldRadioGroup Members 5326
TFieldRadioGroup properties 5328
Values 5334
ValuesField 5335
TFieldSelectOptions class 5335
about TFieldSelectOptions class 5335
Caption 5339
CaptionAttributes 5339
CaptionPosition 5339
Custom 5340
DataSet 5340
DisplayRows 5341
FieldName 5341
Items 5342
ItemsField 5342
Style 5342
StyleRule 5343
TabIndex 5343
TFieldSelectOptions Members 5336
TFieldSelectOptions properties 5338
Values 5343

ValuesField 5344
TFieldStatus class 5344
about TFieldStatus class 5344
Caption 5347
CaptionAttributes 5347
CaptionPosition 5347
Custom 5348
Style 5348
StyleRule 5349
TFieldStatus Members 5345
TFieldStatus properties 5346
TFieldText class 5349
about TFieldText class 5349
Caption 5352
CaptionAttributes 5352
CaptionPosition 5352
Custom 5353
DisplayWidth 5353
FieldName 5354
ReadOnly 5354
Style 5354
StyleRule 5355
TabIndex 5355
TFieldText Members 5350
TFieldText properties 5351
TFieldTextArea class 5355
about TFieldTextArea class 5355
Caption 5358
CaptionAttributes 5359
CaptionPosition 5359
Custom 5359
DisplayRows 5360
DisplayWidth 5360
FieldName 5360
ReadOnly 5361
Style 5361
StyleRule 5362
TabIndex 5362
TFieldTextArea Members 5356
TFieldTextArea properties 5358
Wrap 5362
TFileListBox class 1734
about TFileListBox class 1734
Align 1748
Anchors 1751
ApplyFilePath 1742
AutoComplete 1752
BevelEdges 1752
BevelInner 1753
BevelKind 1753
BevelOuter 1754
Change 1743
Click 1744
Color 1754
Constraints 1755
Create 1738
CreateWnd 1744
Ctl3D 1755
Destroy 1738
DirBMP 1739
Directory 1756
DoubleBuffered 1757
DragCursor 1759
DragMode 1759
DrawItem 1745
Drive 1761
Enabled 1762
ExeBMP 1739
ExtendedSelect 1762
FDirectory 1739
FDirList 1740
FFileEdit 1740
FFileType 1740
FFilterCombo 1740
FileEdit 1763
FileName 1764
FileType 1764
FLastSel 1741

FMask 1741
FOnChange 1741
Font 1765
FShowGlyphs 1741
GetFilePath 1745
ImeMode 1766
ImeName 1767
IntegralHeight 1767
ItemHeight 1768
Mask 1768
MultiSelect 1769
Notification 1745
OnChange 1782
OnClick 1782
OnContextPopup 1784
OnDblClick 1785
OnDragDrop 1785
OnDragOver 1788
OnEndDrag 1791
OnEnter 1792
OnExit 1795
OnKeyDown 1797
OnKeyPress 1801
OnKeyUp 1802
OnMouseActivate 1804
OnMouseDown 1805
OnMouseEnter 1807
OnMouseLeave 1807
OnMouseMove 1808
OnMouseUp 1810
OnStartDrag 1812
ParentColor 1770
ParentCtl3D 1770
ParentDoubleBuffered 1771
ParentFont 1771
ParentShowHint 1771
PopupMenu 1772
ReadBitmaps 1746
ReadFileNames 1746
ShowGlyphs 1777
ShowHint 1777
TabOrder 1779
TabStop 1779
TFileListBox events 1781
TFileListBox fields 1739
TFileListBox Members 1736
TFileListBox methods 1742
TFileListBox properties 1747
UnknownBMP 1741
Update 1746
Visible 1780
TFileRun class 27
about TFileRun class 27
Browse 29
BrowseDlg 29
Caption 30
Directory 30
Enabled 31
FileName 31
HelpContext 31
Hint 32
ImageIndex 32
OnHint 36
OnUpdate 36
Operation 33
Parameters 33
ParentControl 34
SecondaryShortCuts 34
ShortCut 34
ShowCmd 35
TFileRun events 36
TFileRun Members 28
TFileRun properties 29
Visible 35
TFilterComboBox class 1813
about TFilterComboBox class 1813
Align 1819
Anchors 1822

AutoComplete 1823
AutoDropDown 1824
BevelEdges 1825
BevelInner 1825
BevelKind 1825
BevelOuter 1826
BuildList 1816
Change 1816
Click 1816
Color 1826
Constraints 1827
Create 1815
CreateWnd 1817
Ctl3D 1828
Destroy 1815
DoubleBuffered 1828
DragCursor 1830
DragMode 1830
Enabled 1832
FileList 1832
Filter 1833
Font 1835
ImeMode 1836
ImeName 1836
Mask 1837
Notification 1817
OnChange 1851
OnClick 1853
OnContextPopup 1855
OnDblClick 1856
OnDragDrop 1856
OnDragOver 1859
OnDropDown 1862
OnEndDrag 1864
OnEnter 1865
OnExit 1867
OnKeyDown 1870
OnKeyPress 1874
OnKeyUp 1875
OnMouseEnter 1876
OnMouseLeave 1877
OnStartDrag 1877
ParentColor 1838
ParentCtl3D 1839
ParentDoubleBuffered 1839
ParentFont 1839
ParentShowHint 1840
PopupMenu 1840
ShowHint 1845
TabOrder 1847
TabStop 1847
Text 1848
TFilterComboBox events 1850
TFilterComboBox Members 1813
TFilterComboBox methods 1816
TFilterComboBox properties 1818
Visible 1849
TFirstButton class 5363
about TFirstButton class 5363
Create 5365
TFirstButton Members 5363
tfLeft enumeration member 2873
TFlowPanel class 857
about TFlowPanel class 857
Align 863
Alignment 867
Anchors 867
AutoSize 868
AutoWrap 868
BevelEdges 868
BevelInner 869
BevelKind 870
BevelOuter 870
BevelWidth 871
BiDiMode 872
BorderStyle 873
BorderWidth 873
Caption 874

Color 877
Constraints 878
Ctl3D 878
DockSite 879
DoubleBuffered 880
DragCursor 882
DragKind 882
DragMode 882
Enabled 884
FlowStyle 884
Font 888
FullRepaint 889
Locked 889
OnAlignInsertBefore 903
OnAlignPosition 904
OnCanResize 905
OnClick 905
OnConstrainedResize 907
OnContextPopup 907
OnDblClick 908
OnDockDrop 908
OnDockOver 910
OnDragDrop 913
OnDragOver 916
OnEndDock 919
OnEndDrag 919
OnEnter 920
OnExit 923
OnGetSiteInfo 925
OnMouseActivate 928
OnMouseDown 929
OnMouseEnter 931
OnMouseLeave 931
OnMouseMove 931
OnMouseUp 934
OnResize 936
OnStartDock 937
OnStartDrag 937
OnUnDock 938
Padding 889
ParentBackground 890
ParentBiDiMode 890
ParentColor 890
ParentCtl3D 891
ParentDoubleBuffered 891
ParentFont 891
ParentShowHint 892
PopupMenu 893
ShowCaption 898
ShowHint 898
TabOrder 899
TabStop 900
TFlowPanel events 903
TFlowPanel Members 860
TFlowPanel properties 862
UseDockManager 900
VerticalAlignment 901
Visible 901
tfMilliseconds enumeration member 5825
tfMisc enumeration member 3555
tfModifyString enumeration member 2873
tfMSF enumeration member 5825
tfNoClip enumeration member 2873
tfNoPrefix enumeration member 2873
TFont class 2730
about TFont class 2730
Assign 2733
Changed 2734
Charset 2739
Color 2739
Create 2732
Destroy 2732
FontAdapter 2741
GetCharset 2735
GetHandle 2735
GetHeight 2735
GetName 2735
GetOrientation 2736

GetPitch 2736
GetSize 2736
GetStyle 2736
Handle 2741
Height 2742
Name 2743
Orientation 2744
Pitch 2744
PixelsPerInch 2745
SetCharset 2737
SetColor 2737
SetHandle 2737
SetHeight 2737
SetName 2737
SetOrientation 2738
SetPitch 2738
SetSize 2738
SetStyle 2738
Size 2745
Style 2746
TFont Members 2731
TFont methods 2732
TFont properties 2738
TFontRecall class 2747
about TFontRecall class 2747
Create 2747
TFontRecall Members 2747
TForm class 2241
about TForm class 2241
Action 2252
ActiveControl 2253
ActiveMDIChild 2254
Align 2254
AlphaBlend 2257
AlphaBlendValue 2258
Anchors 2258
ArrangeIcons 2247
AutoScroll 2259
AutoSize 2259
BiDiMode 2260
BorderIcons 2260
BorderStyle 2261
BorderWidth 2262
Caption 2262
Cascade 2248
ClientHandle 2264
ClientHeight 2265
ClientWidth 2266
Color 2268
Constraints 2269
Ctl3D 2269
DefaultMonitor 2270
DockManager 2270
DockSite 2272
DoubleBuffered 2273
DragKind 2275
DragMode 2275
Enabled 2277
Font 2277
FormStyle 2279
GlassFrame 2281
Height 2281
HelpFile 2282
HorzScrollBar 2283
Icon 2284
KeyPreview 2284
MDIChildCount 2288
MDIChildren 2289
Menu 2290
Next 2249
ObjectMenuItem 2292
OldCreateOrder 2292
OnActivate 2312
OnAlignInsertBefore 2318
OnAlignPosition 2318
OnCanResize 2319
OnClick 2319
OnClose 2321

OnCloseQuery 2323
OnConstrainedResize 2324
OnContextPopup 2324
OnCreate 2325
OnDblClick 2327
OnDeactivate 2327
OnDestroy 2327
OnDockDrop 2329
OnDockOver 2331
OnDragDrop 2333
OnDragOver 2336
OnEndDock 2339
OnGetSiteInfo 2339
OnHelp 2341
OnHide 2342
OnKeyDown 2344
OnKeyPress 2347
OnKeyUp 2348
OnMouseActivate 2350
OnMouseDown 2351
OnMouseEnter 2353
OnMouseLeave 2354
OnMouseMove 2354
OnMouseUp 2356
OnMouseWheel 2359
OnMouseWheelDown 2359
OnMouseWheelUp 2359
OnPaint 2360
OnResize 2361
OnShortCut 2362
OnShow 2362
OnStartDock 2363
OnUnDock 2364
Padding 2292
ParentBiDiMode 2293
ParentFont 2293
PixelsPerInch 2294
PopupMenu 2294
PopupMode 2299
PopupParent 2300
Position 2300
Previous 2249
PrintScale 2301
Scaled 2302
ScreenSnap 2302
ShowHint 2302
SnapBuffer 2304
TForm events 2311
TForm Members 2242
TForm methods 2246
TForm properties 2250
Tile 2249
TileMode 2304
TransparentColor 2305
TransparentColorValue 2306
UseDockManager 2306
VertScrollBar 2306
Visible 2307
Width 2308
WindowMenu 2310
WindowState 2310
TFormLayout class 5365
about TFormLayout class 5365
BreakButtons 5372
ColumnCount 5372
Create 5366
EndButtons 5367
EndFields 5367
EndLayout 5367
EndRow 5368
EndTable 5368
ImplLayoutButton 5368
ImplLayoutField 5369
ImplLayoutLabelAndField 5369
ImplLayoutTable 5370
NextColumn 5370
StartButtons 5370
StartFields 5371

StartRow 5371
StartTable 5371
TableHeader 5372
TFormLayout Members 5366
TFormLayout methods 5367
TFormLayout properties 5371
tfPathEllipsis enumeration member 2873
tfQExecute enumeration member 3555
tfQFetch enumeration member 3555
tfQPrepare enumeration member 3555
TFrame class 2366
about TFrame class 2366
Align 2370
Anchors 2373
AutoScroll 2374
AutoSize 2374
BiDiMode 2375
Color 2375
Constraints 2376
Ctl3D 2376
DockSite 2377
DoubleBuffered 2378
DragCursor 2380
DragKind 2380
DragMode 2380
Enabled 2382
Font 2382
OnAlignInsertBefore 2397
OnAlignPosition 2397
OnCanResize 2398
OnClick 2398
OnConstrainedResize 2400
OnContextPopup 2401
OnDblClick 2401
OnDockDrop 2402
OnDockOver 2404
OnDragDrop 2406
OnDragOver 2409
OnEndDock 2412
OnEndDrag 2412
OnEnter 2413
OnExit 2416
OnGetSiteInfo 2419
OnMouseActivate 2421
OnMouseDown 2422
OnMouseEnter 2424
OnMouseLeave 2424
OnMouseMove 2424
OnMouseUp 2427
OnMouseWheel 2429
OnMouseWheelDown 2429
OnMouseWheelUp 2430
OnResize 2430
OnStartDock 2431
OnStartDrag 2431
OnUnDock 2432
Padding 2384
ParentBackground 2384
ParentBiDiMode 2384
ParentColor 2385
ParentCtl3D 2385
ParentDoubleBuffered 2385
ParentFont 2386
ParentShowHint 2386
PopupMenu 2387
ShowHint 2392
TabOrder 2393
TabStop 2394
TFrame events 2396
TFrame Members 2367
TFrame properties 2369
Visible 2394
tfRight enumeration member 2873
tfRtlReading enumeration member 2873
tfSamples enumeration member 5825
tfService enumeration member 3555
tfSingleLine enumeration member 2873
tfSMPTE24 enumeration member 5825

tfSMPTE25 enumeration member 5825
tfSMPTE30 enumeration member 5825
tfSMPTE30Drop enumeration member 5825
tfStmt enumeration member 3555
tfTMSF enumeration member 5825
tfTop enumeration member 2873
tfTransact enumeration member 3555
tfVerticalCenter enumeration member 2873
tfWordBreak enumeration member 2873
tgCustom enumeration member 3522
tgEmbed enumeration member 3522
tgImage enumeration member 3522
tgImageMap enumeration member 3522
TGlassFrame class 2435
about TGlassFrame class 2435
Assign 2436
Bottom 2438
Change 2437
Create 2436
Enabled 2438
FrameExtended 2437
IntersectsControl 2438
Left 2438
OnChange 2439
Right 2439
SheetOfGlass 2439
TGlassFrame events 2439
TGlassFrame Members 2435
TGlassFrame methods 2436
TGlassFrame properties 2438
Top 2439
tgLink enumeration member 3522
tgObject enumeration member 3522
TGraphic class 2748
about TGraphic class 2748
Changed 2751
Create 2750
DefineProperties 2751
Draw 2752
DrawTransparent 2752
Empty 2760
Equals 2752, 2753
Equals method 2752
GetEmpty 2753
GetHeight 2753
GetPalette 2754
GetSupportsPartialTransparency 2754
GetTransparent 2754
GetWidth 2754
Height 2760
LoadFromClipboardFormat 2754
LoadFromFile 2755
LoadFromStream 2756
Modified 2761
OnChange 2763
OnProgress 2764
Palette 2761
PaletteModified 2761
Progress 2756
ReadData 2757
SaveToClipboardFormat 2757
SaveToFile 2757
SaveToStream 2758
SetHeight 2758
SetPalette 2759
SetSize 2759
SetTransparent 2759
SetWidth 2759
SupportsPartialTransparency 2762
TGraphic events 2763
TGraphic Members 2749
TGraphic methods 2750
TGraphic properties 2760
Transparent 2762
Width 2763
WriteData 2760
TGraphicsObject class 2764
about TGraphicsObject class 2764

Changed 2765
HandleAllocated 2766
Lock 2766
OnChange 2768
OwnerCriticalSection 2767
TGraphicsObject events 2768
TGraphicsObject Members 2765
TGraphicsObject methods 2765
TGraphicsObject properties 2767
Unlock 2767
TGridAttributes class 5373
about TGridAttributes class 5373
Align 5375
AssignTo 5374
BgColor 5376
Border 5376
CellPadding 5377
CellSpacing 5377
Create 5374
Custom 5377
Parent 5378
Style 5378
StyleRule 5379
TGridAttributes Members 5373
TGridAttributes methods 5374
TGridAttributes properties 5375
TGridPanel class 941
about TGridPanel class 941
Align 945
Alignment 948
Anchors 949
BevelEdges 949
BevelInner 950
BevelKind 951
BevelOuter 951
BevelWidth 952
BiDiMode 953
BorderStyle 954
BorderWidth 954
Caption 955
Color 958
ColumnCollection 959
Constraints 959
ControlCollection 959
Ctl3D 960
DockManager 960
DockSite 962
DoubleBuffered 963
DragCursor 965
DragKind 965
DragMode 966
Enabled 967
ExpandStyle 968
Font 968
FullRepaint 969
Locked 969
OnAlignInsertBefore 984
OnAlignPosition 984
OnCanResize 985
OnClick 986
OnConstrainedResize 987
OnContextPopup 988
OnDblClick 989
OnDockDrop 989
OnDockOver 991
OnDragDrop 993
OnDragOver 996
OnEndDock 999
OnEndDrag 1000
OnEnter 1001
OnExit 1003
OnGetSiteInfo 1006
OnMouseActivate 1008
OnMouseDown 1009
OnMouseEnter 1011
OnMouseLeave 1011
OnMouseMove 1012
OnMouseUp 1014

OnResize 1016
OnStartDock 1017
OnStartDrag 1018
OnUnDock 1018
Padding 970
ParentBackground 970
ParentBiDiMode 970
ParentColor 971
ParentCtl3D 971
ParentDoubleBuffered 972
ParentFont 972
ParentShowHint 972
PopupMenu 973
RowCollection 978
ShowCaption 978
ShowHint 978
TabOrder 980
TabStop 980
TGridPanel events 983
TGridPanel Members 941
TGridPanel properties 944
UseDockManager 981
VerticalAlignment 981
Visible 982
TGridRowAttributes class 5379
about TGridRowAttributes class 5379
Align 5382
AssignTo 5381
BgColor 5382
Create 5380
Custom 5382
Parent 5383
Style 5383
StyleRule 5384
TGridRowAttributes Members 5379
TGridRowAttributes methods 5380
TGridRowAttributes properties 5381
VAlign 5384
tgTable enumeration member 3522
THashedStringList class 4291
about THashedStringList class 4291
Changed 4292
Destroy 4292
IndexOf 4293
IndexOfName 4293
THashedStringList Members 4292
THashedStringList methods 4292
THeader class 1021
about THeader class 1021
Align 1027
AllowResize 1030
Anchors 1030
BorderStyle 1031
Constraints 1031
Create 1022
CreateParams 1023
Destroy 1023
DoubleBuffered 1032
Enabled 1034
Font 1034
MouseDown 1024
MouseMove 1024
MouseUp 1025
OnContextPopup 1046
OnSized 1047
OnSizing 1047
Paint 1025
ParentDoubleBuffered 1035
ParentFont 1036
ParentShowHint 1036
PopupMenu 1037
Sections 1042
SectionWidth 1042
ShowHint 1042
Sized 1026
Sizing 1026
TabOrder 1044
TabStop 1044

THeader events 1046
THeader Members 1021
THeader methods 1023
THeader properties 1026
Visible 1045
THeaderList class 4363
about THeaderList class 4363
Add 4365
Add method 4365
Clear 4366
Count 4368
Create 4364
Destroy 4364
Extract 4366
GetCount 4366
GetHeader 4367
GetOwnsObjects 4367
Headers 4368
IndexOf 4367
OwnsObjects 4369
SetOwnsObjects 4367
THeaderList Members 4363
THeaderList methods 4365
THeaderList properties 4367
THelpButton class 6825
about THelpButton class 6825
DrawImage 6826
THelpButton Members 6825
THelpButton methods 6826
THTMLTableAttributes class 3478
about THTMLTableAttributes class 3478
Align 3481
AssignTo 3480
BgColor 3482
Border 3482
CellPadding 3483
CellSpacing 3483
Create 3479
RestoreDefaults 3480
THTMLTableAttributes Members 3478
THTMLTableAttributes methods 3480
THTMLTableAttributes properties 3481
Width 3483
THTMLTableCellAttributes class 3484
about THTMLTableCellAttributes class 3484
THTMLTableElementAttributes class 3484
about THTMLTableElementAttributes class 3484
Align 3487
AssignTo 3486
BgColor 3487
RestoreDefaults 3486
THTMLTableElementAttributes Members 3485
THTMLTableElementAttributes methods 3486
THTMLTableElementAttributes properties 3487
VAlign 3488
THTMLTableHeaderAttributes class 3488
about THTMLTableHeaderAttributes class 3488
AssignTo 3490
Caption 3491
RestoreDefaults 3490
THTMLTableHeaderAttributes Members 3489
THTMLTableHeaderAttributes methods 3489
THTMLTableHeaderAttributes properties 3491
THTMLTableRowAttributes class 3491
about THTMLTableRowAttributes class 3491
THTMLTagAttributes class 3492
about THTMLTagAttributes class 3492
Changed 3494
Create 3493
Custom 3494
OnChange 3495
Producer 3495
RestoreDefaults 3494
THTMLTagAttributes events 3495
THTMLTagAttributes Members 3492
THTMLTagAttributes methods 3493
THTMLTagAttributes properties 3494
TIBBackupRestoreService class 3946

about TIBBackupRestoreService class 3946
TIBBackupRestoreService Members 3946
TIBBackupRestoreService properties 3947
Verbose 3948
TIBBackupService class 3948
about TIBBackupService class 3948
BackupFile 3951
BlockingFactor 3951
Create 3950
DatabaseName 3951
Destroy 3950
Options 3951
SetServiceStartOptions 3950
TIBBackupService Members 3948
TIBBackupService methods 3950
TIBBackupService properties 3950
TIBBase class 3705
about TIBBase class 3705
AfterDatabaseDisconnect 3715
AfterTransactionEnd 3715
BeforeDatabaseDisconnect 3715
BeforeTransactionEnd 3715
CheckDatabase 3710
CheckTransaction 3710
Create 3706
Database 3713
DBHandle 3713
Destroy 3707
DoAfterDatabaseDisconnect 3711
DoAfterTransactionEnd 3711
DoBeforeDatabaseDisconnect 3711
DoBeforeTransactionEnd 3711
DoDatabaseFree 3711
DoTransactionFree 3712
FAfterDatabaseDisconnect 3707
FAfterTransactionEnd 3707
FBeforeDatabaseDisconnect 3708
FBeforeTransactionEnd 3708
FDatabase 3708
FIndexInDatabase 3708
FIndexInTransaction 3709
FOnDatabaseFree 3709
FOnTransactionFree 3709
FOwner 3709
FTransaction 3709
GetDBHandle 3712
GetTRHandle 3712
OnDatabaseFree 3716
OnTransactionFree 3716
Owner 3714
SetDatabase 3712
SetTransaction 3713
TIBBase events 3714
TIBBase fields 3707
TIBBase Members 3705
TIBBase methods 3710
TIBBase properties 3713
Transaction 3714
TRHandle 3714
TIBBatch class 4036
about TIBBatch class 4036
Columns 4038
FColumns 4037
FFilename 4037
Filename 4039
FParams 4038
Params 4039
ReadyFile 4038
TIBBatch fields 4037
TIBBatch Members 4036
TIBBatch methods 4038
TIBBatch properties 4038
TIBBatchInput class 4039
about TIBBatchInput class 4039
ReadParameters 4040
TIBBatchInput Members 4040
TIBBatchInput methods 4040
TIBBatchMove class 3566

AbortOnKeyViol 3570
AbortOnProblem 3570
about TIBBatchMove class 3566
ChangedCount 3571
ChangedTableName 3571
CommitCount 3571
Create 3567
Destination 3572
Destroy 3568
Execute 3568
KeyViolCount 3572
KeyViolTableName 3573
Mappings 3573
Mode 3574
MovedCount 3574
Notification 3569
ProblemCount 3575
ProblemTableName 3575
RecordCount 3576
Source 3576
TIBBatchMove Members 3566
TIBBatchMove methods 3568
TIBBatchMove properties 3569
Transliterate 3577
TIBBatchOutput class 4040
about TIBBatchOutput class 4040
TIBBatchOutput Members 4041
TIBBatchOutput methods 4041
WriteColumns 4041
TIBBCDField class 3599
about TIBBCDField class 3599
CheckTypeSize 3600
Create 3600
GetAsCurrency 3601
GetAsString 3601
GetAsVariant 3601
GetDataSize 3601
Size 3602
TIBBCDField Members 3599
TIBBCDField methods 3600
TIBBCDField properties 3601
TIBBlobStream class 3580
about TIBBlobStream class 3580
BlobID 3590
BlobMaxSegmentSize 3591
BlobNumSegments 3591
BlobSize 3591
BlobType 3591
Call 3583
Cancel 3583
CheckReadable 3583
CheckWritable 3584
CloseBlob 3584
Create 3581
CreateBlob 3584
Database 3592
DBHandle 3592
Destroy 3582
EnsureBlobInitialized 3584
Finalize 3585
GetBlobInfo 3585
GetDatabase 3585
GetDBHandle 3585
GetTransaction 3585
GetTRHandle 3586
Handle 3592
LoadFromFile 3586
LoadFromStream 3586
Mode 3592
Modified 3593
OpenBlob 3586
Read 3587
SaveToFile 3587
SaveToStream 3587
Seek 3588
SetBlobID 3588
SetDatabase 3588
SetMode 3589

SetSize 3589
SetTransaction 3589
TIBBlobStream Members 3580
TIBBlobStream methods 3582
TIBBlobStream properties 3590
Transaction 3593
TRHandle 3593
Truncate 3589
Write 3590
TIBConfigService class 3952
about TIBConfigService class 3952
ActivateShadow 3954
BringDatabaseOnline 3954
DatabaseName 3957
ServiceStart 3954
SetAsyncMode 3955
SetDBSqlDialect 3955
SetPageBuffers 3955
SetReadOnly 3956
SetReserveSpace 3956
SetSweepInterval 3956
ShutdownDatabase 3957
TIBConfigService Members 3952
TIBConfigService methods 3953
TIBConfigService properties 3957
TIBControlAndQueryService class 3957
about TIBControlAndQueryService class 3957
Action 3960
BufferSize 3961
Create 3959
Eof 3961
GetNextChunk 3959
GetNextLine 3960
TIBControlAndQueryService Members 3958
TIBControlAndQueryService methods 3959
TIBControlAndQueryService properties 3960
TIBControlService class 3961
about TIBControlService class 3961
Create 3963
InternalServiceStart 3963
IsServiceRunning 3965
ServiceStart 3963
ServiceStartAddParam 3964
ServiceStartAddParam method 3964
ServiceStartParams 3965
SetServiceStartOptions 3964
TIBControlService Members 3962
TIBControlService methods 3963
TIBControlService properties 3964
TIBCustomDataSet class 3602
about TIBCustomDataSet class 3602
ActivateConnection 3609
ActivateTransaction 3609
AfterCancel 3634
AfterClose 3634
AfterDatabaseDisconnect 3649
AfterDelete 3634
AfterEdit 3634
AfterInsert 3635
AfterOpen 3635
AfterPost 3635
AfterRefresh 3635
AfterScroll 3635
AfterTransactionEnd 3649
AllocRecordBuffer 3609
ApplyUpdates 3609
AutoCalcFields 3636
BeforeCancel 3636
BeforeClose 3636
BeforeDatabaseDisconnect 3649
BeforeDelete 3636
BeforeEdit 3637
BeforeInsert 3637
BeforeOpen 3637
BeforePost 3637
BeforeRefresh 3637
BeforeScroll 3638
BeforeTransactionEnd 3649

BookmarkValid 3610
BufferChunks 3638
CachedUpdates 3638
CachedUpdateStatus 3610
CancelUpdates 3610
CheckDatasetClosed 3611
CheckDatasetOpen 3611
CheckNotUniDirectional 3611
ClearCalcFields 3611
CompareBookmarks 3611
ConstraintsStored 3612
Create 3606
CreateBlobStream 3612
CreateFields 3612
Current 3613
Database 3639
DatabaseFree 3650
DBHandle 3639
DeactivateTransaction 3613
DeleteSQL 3639
Destroy 3606
Disconnect 3613
DoBeforeDelete 3613
DoBeforeEdit 3613
DoBeforeInsert 3614
DoOnNewRecord 3614
FetchAll 3614
ForcedRefresh 3639
FreeRecordBuffer 3615
GeneratorField 3640
GetActiveBuf 3615
GetBookmarkData 3615
GetBookmarkFlag 3615
GetCanModify 3615
GetCurrentRecord 3616
GetDataSource 3616
GetFieldClass 3616
GetFieldData 3617
GetFieldData method 3616
GetRecNo 3617
GetRecord 3617
GetRecordCount 3618
GetRecordSize 3618
InitRecord 3618
InsertSQL 3640
InternalAddRecord 3618
InternalBatchInput 3619
InternalBatchOutput 3619
InternalCancel 3619
InternalClose 3619
InternalDelete 3619
InternalExecQuery 3620
InternalFirst 3620
InternalGotoBookmark 3620
InternalHandleException 3620
InternalInitFieldDefs 3621
InternalInitRecord 3621
InternalInsert 3621
InternalLast 3621
InternalOpen 3621
InternalPost 3622
InternalPrepare 3622
InternalPrepared 3640
InternalRefresh 3622
InternalRefreshRow 3622
InternalSetParamsFromCursor 3623
InternalSetToRecord 3623
InternalUnPrepare 3623
IsCursorOpen 3623
IsSequenced 3623
LiveMode 3641
Loaded 3624
Locate 3624
LocateNext 3625
Lookup 3625
ModifySQL 3641
ObjectView 3641
OnCalcFields 3641

OnDeleteError 3642
OnEditError 3642
OnNewRecord 3642
OnPostError 3642
OnUpdateError 3650
OnUpdateRecord 3651
ParamCheck 3642
Params 3643
Plan 3643
Post 3626
PSEndTransaction 3626
PSExecuteStatement 3626
PSGetQuoteChar 3627
PSGetTableName 3627
PSGetUpdateException 3627
PSInTransaction 3627
PSIsSQLBased 3628
PSIsSQLSupported 3628
PSReset 3628
PSStartTransaction 3628
PSUpdateRecord 3628
QDelete 3643
QInsert 3644
QModify 3644
QRefresh 3644
QSelect 3644
RecordModified 3629
RefreshSQL 3645
ReQuery 3629
RevertRecord 3629
RowsAffected 3645
SelectSQL 3645
SelectStmtHandle 3645
SetActive 3629
SetBookmarkData 3630
SetBookmarkFlag 3630
SetCachedUpdates 3630
SetDataSource 3630
SetFieldData 3631
SetFieldData method 3630
SetRecNo 3631
SQLParams 3646
SQLType 3631
StatementType 3646
TIBCustomDataSet events 3648
TIBCustomDataSet Members 3602
TIBCustomDataSet methods 3607
TIBCustomDataSet properties 3632
Transaction 3646
TransactionFree 3651
TRHandle 3646
Undelete 3632
UniDirectional 3647
UpdateMode 3647
UpdateObject 3647
UpdateRecordTypes 3648
UpdatesPending 3648
UpdateStatus 3632
TIBCustomService class 3965
about TIBCustomService class 3965
Active 3970
Attach 3967
BufferSize 3970
CheckActive 3968
CheckInactive 3968
Create 3966
Destroy 3967
Detach 3968
Handle 3970
InternalServiceQuery 3968
Loaded 3968
Login 3969
LoginPrompt 3971
OnAttach 3973
OnLogin 3973
OutputBuffer 3971
OutputBufferOption 3971
Params 3971

Protocol 3971
ServerName 3972
ServiceParamBySPB 3972
ServiceQueryParams 3972
TIBCustomService events 3973
TIBCustomService Members 3966
TIBCustomService methods 3967
TIBCustomService properties 3969
TraceFlags 3972
TIBCustomSQLMonitor class 4103
about TIBCustomSQLMonitor class 4103
Create 4104
Destroy 4104
Enabled 4105
Handle 4105
OnSQL 4106
Release 4104
TIBCustomSQLMonitor events 4106
TIBCustomSQLMonitor Members 4103
TIBCustomSQLMonitor methods 4104
TIBCustomSQLMonitor properties 4105
TraceFlags 4106
TIBDataBase class 3716
about TIBDataBase class 3716
AddEventNotifier 3720
AddTransaction 3720
AfterConnect 3738
AfterDisconnect 3739
AllowStreamedConnected 3729
ApplyUpdates 3720
BeforeConnect 3739
BeforeDisconnect 3739
Call 3721
CheckActive 3721
CheckDatabaseName 3721
CheckInactive 3721
CloseDataSets 3722
Connected 3729
Create 3718
CreateDatabase 3722
DatabaseName 3730
DBParamByDPB 3730
DBSQLDialect 3730
DefaultTransaction 3731
Destroy 3718
DoConnect 3722
DoDisconnect 3723
DropDatabase 3723
FindDefaultTransaction 3723
FindTransaction 3724
FlushSchema 3724
ForceClose 3724
GetConnected 3724
GetFieldNames 3725
GetTableNames 3725
Handle 3731
HandleIsShared 3731
Has_COMPUTED_BLR 3725
Has_DEFAULT_VALUE 3726
IdleTimer 3731
IndexOfDBConst 3726
InternalTransaction 3732
IsReadOnly 3732
Loaded 3726
LoginPrompt 3732
Notification 3726
OnDialectDowngradeWarning 3740
OnIdleTimer 3740
OnLogin 3741
Params 3736
RemoveEventNotifier 3727
RemoveTransaction 3727
RemoveTransactions 3728
SetHandle 3728
SQLDialect 3736
SQLObjectCount 3737
SQLObjects 3737
TestConnected 3728

TIBDataBase events 3738
TIBDataBase Members 3716
TIBDataBase methods 3719
TIBDataBase properties 3728
TraceFlags 3737
TransactionCount 3738
Transactions 3738
TIBDatabaseInfo class 3772
about TIBDatabaseInfo class 3772
Allocation 3787
BackoutCount 3787
BaseLevel 3788
Call 3778
Create 3774
CurrentMemory 3788
Database 3788
DBFileName 3788
DBImplementationClass 3789
DBImplementationNo 3789
DBSiteName 3789
DBSQLDialect 3789
DeleteCount 3790
Destroy 3774
ExpungeCount 3790
FBackoutCount 3775
FDatabase 3775
FDeleteCount 3776
Fetches 3790
FExpungeCount 3776
FIBLoaded 3776
FInsertCount 3776
ForcedWrites 3790
FPurgeCount 3776
FReadIdxCount 3777
FReadSeqCount 3777
FUpdateCount 3777
FUserNames 3777
GetAllocation 3779
GetBackoutCount 3779
GetBaseLevel 3779
GetCurrentMemory 3779
GetDBFileName 3779
GetDBImplementationClass 3780
GetDBImplementationNo 3780
GetDBSiteName 3780
GetDBSQLDialect 3780
GetDeleteCount 3781
GetExpungeCount 3781
GetFetches 3781
GetForcedWrites 3781
GetInsertCount 3781
GetLongDatabaseInfo 3782
GetMarks 3782
GetMaxMemory 3782
GetNoReserve 3782
GetNumBuffers 3783
GetODSMajorVersion 3783
GetODSMinorVersion 3783
GetOperationCounts 3783
GetPageSize 3784
GetPurgeCount 3784
GetReadIdxCount 3784
GetReadOnly 3784
GetReads 3784
GetReadSeqCount 3785
GetStringDatabaseInfo 3785
GetSweepInterval 3785
GetUpdateCount 3785
GetUserNames 3786
GetVersion 3786
GetWrites 3786
InsertCount 3791
Marks 3791
MaxMemory 3791
NoReserve 3791
NumBuffers 3792
ODSMajorVersion 3792
ODSMinorVersion 3792

PageSize 3792
PurgeCount 3793
ReadIdxCount 3793
ReadOnly 3793
Reads 3793
ReadSeqCount 3794
SweepInterval 3794
TIBDatabaseInfo fields 3775
TIBDatabaseInfo Members 3772
TIBDatabaseInfo methods 3777
TIBDatabaseInfo properties 3786
UpdateCount 3794
UserNames 3795
Version 3795
Writes 3795
TIBDataLink class 3651
about TIBDataLink class 3651
ActiveChanged 3653
CheckBrowseMode 3653
Create 3652
Destroy 3652
GetDetailDataSet 3653
RecordChanged 3654
TIBDataLink Members 3652
TIBDataLink methods 3653
TIBDataSet class 3654
about TIBDataSet class 3654
Active 3662
AfterCancel 3662
AfterClose 3662
AfterDatabaseDisconnect 3673
AfterDelete 3663
AfterEdit 3663
AfterInsert 3663
AfterOpen 3663
AfterPost 3663
AfterScroll 3664
AfterTransactionEnd 3673
AutoCalcFields 3664
BatchInput 3659
BatchOutput 3659
BeforeCancel 3664
BeforeClose 3664
BeforeDatabaseDisconnect 3673
BeforeDelete 3665
BeforeEdit 3665
BeforeInsert 3665
BeforeOpen 3665
BeforePost 3665
BeforeScroll 3666
BeforeTransactionEnd 3674
BufferChunks 3666
CachedUpdates 3666
DatabaseFree 3674
DataSource 3667
DeleteSQL 3667
ExecSQL 3659
Filtered 3667
GeneratorField 3667
InsertSQL 3668
InternalOpen 3659
LiveMode 3668
ModifySQL 3668
OnCalcFields 3669
OnDeleteError 3669
OnEditError 3669
OnFilterRecord 3669
OnNewRecord 3669
OnPostError 3670
ParamByName 3660
ParamCheck 3670
Params 3670
Prepare 3660
Prepared 3671
PSSetCommandText 3660
RefreshSQL 3671
SelectSQL 3671
SelectStmtHandle 3671

SetFiltered 3660
StatementType 3672
TIBDataSet events 3673
TIBDataSet Members 3654
TIBDataSet methods 3658
TIBDataSet properties 3661
TransactionFree 3674
UniDirectional 3672
UnPrepare 3661
UpdateObject 3672
TIBDataSetUpdateObject class 3674
about TIBDataSetUpdateObject class 3674
Apply 3676
Create 3675
DataSet 3677
Destroy 3676
GetDataSet 3676
GetSQL 3677
RefreshSQL 3677
SetDataSet 3677
TIBDataSetUpdateObject Members 3675
TIBDataSetUpdateObject methods 3676
TIBDataSetUpdateObject properties 3677
TIBDSBlobStream class 3678
about TIBDSBlobStream class 3678
Create 3679
Destroy 3679
FBlobStream 3679
FField 3680
FModified 3680
Read 3680
Seek 3681
SetSize 3681
TIBDSBlobStream fields 3679
TIBDSBlobStream Members 3678
TIBDSBlobStream methods 3680
Write 3681
TIBEvents class 3798
about TIBEvents class 3798
AutoRegister 3803
Create 3799
Database 3803
Destroy 3800
EventChange 3800
Events 3804
GetAutoRegister 3800
GetNativeHandle 3801
NativeHandle 3804
Notification 3801
OnError 3805
OnEventAlert 3805
Registered 3804
RegisterEvents 3801
SetAutoRegister 3802
ThreadEnded 3802
TIBEvents events 3805
TIBEvents Members 3798
TIBEvents methods 3800
TIBEvents properties 3803
UnRegisterEvents 3802
ValidateDatabase 3803
TIBExtract class 3811
about TIBExtract class 3811
Create 3812
Database 3821
DatabaseInfo 3821
Destroy 3813
ExtractDDL 3814
ExtractListTable 3814
ExtractListView 3814
ExtractObject 3814
GetArrayField 3816
GetCharacterSets 3816
GetFieldType 3816
Items 3821
ListAllTables 3816
ListCheck 3817
ListCreateDb 3817

ListData 3817
ListDomains 3817
ListException 3818
ListFilters 3818
ListForeign 3818
ListFunctions 3818
ListGenerators 3818
ListGrants 3819
ListIndex 3819
ListProcs 3819
ListRoles 3819
ListTriggers 3820
ListViews 3820
Notification 3820
PrintSet 3820
ShowSystem 3822
TIBExtract Members 3811
TIBExtract methods 3813
TIBExtract properties 3821
Transaction 3822
TIBGeneratorField class 3682
about TIBGeneratorField class 3682
Apply 3683
ApplyEvent 3685
Assign 3684
Create 3683
Field 3685
Generator 3685
IncrementBy 3686
TIBGeneratorField Members 3682
TIBGeneratorField methods 3683
TIBGeneratorField properties 3684
ValueName 3684
TIBInputDelimitedFile class 4042
about TIBInputDelimitedFile class 4042
ColDelimiter 4046
Destroy 4043
FColDelimiter 4043
FEOF 4043
FFile 4044
FLookAhead 4044
FReadBlanksAsNull 4044
FRowDelimiter 4044
FSkipTitles 4045
GetColumn 4045
ReadBlanksAsNull 4046
ReadParameters 4045
ReadyFile 4045
RowDelimiter 4046
SkipTitles 4047
TIBInputDelimitedFile fields 4043
TIBInputDelimitedFile Members 4042
TIBInputDelimitedFile methods 4045
TIBInputDelimitedFile properties 4046
TIBInputRawFile class 4047
about TIBInputRawFile class 4047
Destroy 4048
FFile 4048
ReadParameters 4049
ReadyFile 4049
TIBInputRawFile fields 4048
TIBInputRawFile Members 4047
TIBInputRawFile methods 4048
TIBInstall class 3850
about TIBInstall class 3850
Create 3851
DestinationDirectory 3855
Destroy 3852
GetOptionDescription 3852
GetOptionDescription method 3852
GetOptionName 3853
GetOptionName method 3853
GetOptionSpaceRequired 3854
GetOptionSpaceRequired method 3853
InstallCheck 3854
InstallExecute 3854
InstallOptions 3855
SourceDirectory 3856

SuggestedDestination 3856
TIBInstall Members 3850
TIBInstall methods 3852
TIBInstall properties 3855
UnInstallFile 3856
TIBLicensingService class 3973
about TIBLicensingService class 3973
Action 3976
AddLicense 3975
ID 3977
Key 3977
RemoveLicense 3976
SetServiceStartOptions 3976
TIBLicensingService Members 3974
TIBLicensingService methods 3975
TIBLicensingService properties 3976
TIBLogService class 3977
about TIBLogService class 3977
SetServiceStartOptions 3979
TIBLogService Members 3978
TIBLogService methods 3979
TIBOutputDelimitedFile class 4049
about TIBOutputDelimitedFile class 4049
ColDelimiter 4052
Destroy 4050
FColDelimiter 4051
FFile 4051
FOutputTitles 4051
FRowDelimiter 4051
OutputTitles 4053
ReadyFile 4052
RowDelimiter 4053
TIBOutputDelimitedFile fields 4050
TIBOutputDelimitedFile Members 4050
TIBOutputDelimitedFile methods 4051
TIBOutputDelimitedFile properties 4052
WriteColumns 4052
TIBOutputRawFile class 4053
about TIBOutputRawFile class 4053
Destroy 4054
FFile 4054
ReadyFile 4055
TIBOutputRawFile fields 4054
TIBOutputRawFile Members 4054
TIBOutputRawFile methods 4055
WriteColumns 4055
TIBQuery class 3916
about TIBQuery class 3916
Active 3927
AfterDatabaseDisconnect 3934
AfterTransactionEnd 3934
BatchInput 3922
BatchOutput 3922
BeforeDatabaseDisconnect 3934
BeforeTransactionEnd 3935
BufferChunks 3927
CachedUpdates 3928
Constraints 3928
Create 3921
DatabaseFree 3935
DataSource 3928
DefineProperties 3922
Destroy 3921
Disconnect 3923
ExecSQL 3923
Filtered 3929
GenerateParamNames 3929
GenerateQueryForLiveUpdate 3923
GeneratorField 3929
GetDetailLinkFields 3923
GetParamsCount 3924
InitFieldDefs 3924
InternalOpen 3924
LiveMode 3929
OnFilterRecord 3930
ParamByName 3924
ParamCheck 3930
ParamCount 3930

Params 3931
Prepare 3925
Prepared 3931
PSExecute 3925
PSGetParams 3925
PSGetTableName 3925
PSSetCommandText 3926
PSSetParams 3926
RowsAffected 3931
SetFiltered 3926
SQL 3932
StatementType 3932
StmtHandle 3932
Text 3932
TIBQuery events 3934
TIBQuery Members 3916
TIBQuery methods 3921
TIBQuery properties 3927
TransactionFree 3935
UniDirectional 3933
UnPrepare 3926
UpdateObject 3933
TIBRestoreService class 3979
about TIBRestoreService class 3979
BackupFile 3982
Create 3981
DatabaseName 3982
Destroy 3981
Options 3983
PageBuffers 3983
PageSize 3983
SetServiceStartOptions 3982
TIBRestoreService Members 3980
TIBRestoreService methods 3981
TIBRestoreService properties 3982
TIBSchema class 3741
about TIBSchema class 3741
FreeNodes 3742
Has_COMPUTED_BLR 3742
Has_DEFAULT_VALUE 3742
TIBSchema Members 3741
TIBSchema methods 3742
TIBSecurityService class 3983
about TIBSecurityService class 3983
AddUser 3986
Create 3985
DeleteUser 3986
Destroy 3986
DisplayUser 3987
DisplayUsers 3987
FirstName 3989
GroupID 3989
LastName 3989
Loaded 3987
MiddleName 3990
ModifyUser 3988
Password 3990
SecurityAction 3990
SetServiceStartOptions 3988
SQlRole 3991
TIBSecurityService Members 3984
TIBSecurityService methods 3986
TIBSecurityService properties 3988
UserID 3991
UserInfo 3991
UserInfoCount 3991
UserName 3992
TIBServerProperties class 3992
about TIBServerProperties class 3992
ConfigParams 3996
Create 3993
DatabaseInfo 3997
Destroy 3994
Fetch 3994
FetchConfigParams 3995
FetchDatabaseInfo 3995
FetchLicenseInfo 3995
FetchLicenseMaskInfo 3995

FetchVersionInfo 3996
LicenseInfo 3997
LicenseMaskInfo 3997
Options 3997
TIBServerProperties Members 3992
TIBServerProperties methods 3994
TIBServerProperties properties 3996
VersionInfo 3998
TIBSetup class 3857
about TIBSetup class 3857
Call 3858
Create 3858
ErrorContext 3859
ErrorInternal 3858
GetInstallMessage 3859
IBInstallError 3859
MsgFilePath 3860
OnError 3861
OnStatusChange 3862
OnWarning 3862
Progress 3860
RebootToComplete 3860
StatusContext 3861
StatusInternal 3859
TIBSetup events 3861
TIBSetup Members 3857
TIBSetup methods 3858
TIBSetup properties 3859
TIBSQL class 4055
about TIBSQL class 4055
BatchInput 4063
BatchOutput 4064
BeforeTransactionEnd 4064
Bof 4072
Call 4064
CheckClosed 4064
CheckOpen 4065
CheckValidStatement 4065
Close 4065
Create 4058
Current 4065
Database 4072
DBHandle 4072
Destroy 4058
DoBeforeDatabaseDisconnect 4066
Eof 4072
ExecQuery 4066
FBase 4059
FBOF 4059
FCursor 4059
FEOF 4059
FGenerateParamNames 4060
FGoToFirstRecordOnExecute 4060
FHandle 4060
FieldByName 4066
FieldIndex 4073
Fields 4073
FOnSQLChanging 4060
FOpen 4061
FParamCheck 4061
FPrepared 4061
FProcessedSQL 4061
FRecordCount 4061
FreeHandle 4066
FSQL 4062
FSQLParams 4062
FSQLRecord 4062
FSQLType 4062
GenerateParamNames 4073
GetDatabase 4067
GetDBHandle 4067
GetEOF 4067
GetFieldIndex 4067
GetFields 4067
GetPlan 4068
GetRecordCount 4068
GetRowsAffected 4068
GetSQLParams 4068

GetTransaction 4069
GetTRHandle 4069
GetUniqueRelationName 4069
GoToFirstRecordOnExecute 4074
Handle 4074
Next 4069
OnSQLChanging 4077
Open 4074
ParamByName 4069
ParamCheck 4074
Params 4075
Plan 4075
Prepare 4070
Prepared 4075
PreprocessSQL 4070
RecordCount 4075
RowsAffected 4076
SetDatabase 4070
SetSQL 4070
SetTransaction 4071
SQL 4076
SQLChanging 4071
SQLType 4076
TIBSQL events 4077
TIBSQL fields 4058
TIBSQL Members 4056
TIBSQL methods 4062
TIBSQL properties 4071
Transaction 4077
TRHandle 4077
UniqueRelationName 4077
TIBSQLMonitor class 4106
about TIBSQLMonitor class 4106
Enabled 4107
OnSQL 4108
TIBSQLMonitor events 4108
TIBSQLMonitor Members 4107
TIBSQLMonitor properties 4107
TraceFlags 4108
TIBStatisticalService class 3998
about TIBStatisticalService class 3998
DatabaseName 4000
Options 4001
SetServiceStartOptions 4000
TIBStatisticalService Members 3999
TIBStatisticalService methods 4000
TIBStatisticalService properties 4000
TIBStoredProc class 4124
about TIBStoredProc class 4124
AfterDatabaseDisconnect 4137
AfterTransactionEnd 4137
BeforeDatabaseDisconnect 4138
BeforeTransactionEnd 4138
CopyParams 4130
Create 4128
DatabaseFree 4138
DefineProperties 4130
Destroy 4129
Disconnect 4130
ExecProc 4130
Filtered 4134
GetParamsCount 4131
InternalOpen 4131
OnFilterRecord 4135
ParamByName 4131
ParamCount 4135
Params 4135
Prepare 4132
Prepared 4136
PSExecute 4132
PSGetParams 4132
PSGetTableName 4132
PSSetCommandText 4133
PSSetParams 4133
SetFiltered 4133
SetPrepare 4133
SetPrepared 4133
SetProcName 4134

StmtHandle 4136
StoredProcedureNames 4136
StoredProcName 4137
TIBStoredProc events 4137
TIBStoredProc Members 4124
TIBStoredProc methods 4129
TIBStoredProc properties 4134
TransactionFree 4138
UnPrepare 4134
TIBStringField class 3686
about TIBStringField class 3686
CheckTypeSize 3687
Create 3687
GetAsString 3688
GetAsVariant 3688
GetValue 3688
SetAsString 3688
TIBStringField Members 3686
TIBStringField methods 3687
TIBTable class 4142
about TIBTable class 4142
Active 4157
AddIndex 4148
AfterDatabaseDisconnect 4165
AfterTransactionEnd 4165
BeforeDatabaseDisconnect 4166
BeforeTransactionEnd 4166
BufferChunks 4157
CachedUpdates 4157
Constraints 4158
Create 4147
CreateTable 4148
CurrentDBKey 4158
DatabaseFree 4166
DataEvent 4149
DefaultIndex 4158
DefChanged 4149
DeleteIndex 4149
DeleteTable 4150
Destroy 4147
DoOnNewRecord 4150
EmptyTable 4150
Exists 4158
FieldDefs 4159
Filter 4159
Filtered 4159
GetCanModify 4150
GetDataSource 4151
GetDetailLinkFields 4151
GetIndexNames 4151
GetIndexParams 4151
GotoCurrent 4152
IndexDefs 4160
IndexFieldCount 4160
IndexFieldNames 4160
IndexFields 4161
IndexName 4161
InitFieldDefs 4152
InternalClose 4152
InternalOpen 4152
InternalRefresh 4153
InternalRefreshRow 4153
LiveMode 4161
Locate 4153
MasterFields 4162
MasterSource 4162
OnFilterRecord 4162
PSGetDefaultOrder 4154
PSGetIndexDefs 4154
PSGetKeyFields 4154
PSGetTableName 4154
PSSetCommandText 4155
PSSetParams 4155
ReadOnly 4163
SetFiltered 4155
SetFilterOptions 4155
SetFilterText 4156
StoreDefs 4163

TableName 4163
TableNames 4164
TableTypes 4164
TIBTable events 4165
TIBTable Members 4142
TIBTable methods 4147
TIBTable properties 4156
TransactionFree 4166
UniDirectional 4164
UpdateIndexDefs 4156
UpdateObject 4165
TIBTimer class 3742
about TIBTimer class 3742
Create 3743
Destroy 3744
Enabled 3745
Interval 3746
OnTimer 3746
TIBTimer events 3746
TIBTimer Members 3743
TIBTimer methods 3745
TIBTimer properties 3745
Timer 3745
TIBTransaction class 3746
about TIBTransaction class 3746
Active 3755
AddDatabase 3749
AutoStopAction 3755
Call 3749
CheckAutoStop 3750
CheckDatabasesInList 3750
CheckInTransaction 3750
CheckNotInTransaction 3750
Commit 3751
CommitRetaining 3751
Create 3748
DatabaseCount 3756
Databases 3756
DefaultAction 3756
DefaultDatabase 3757
Destroy 3748
FindDatabase 3751
FindDefaultDatabase 3752
Handle 3757
HandleIsShared 3757
IdleTimer 3757
InTransaction 3758
Loaded 3752
Notification 3752
OnIdleTimer 3759
Params 3758
RemoveDatabase 3753
RemoveDatabases 3753
Rollback 3753
RollbackRetaining 3754
SetHandle 3754
SQLObjectCount 3758
SQLObjects 3758
StartTransaction 3754
TIBTransaction events 3759
TIBTransaction Members 3747
TIBTransaction methods 3748
TIBTransaction properties 3755
TPB 3759
TPBLength 3759
TIBUnInstall class 3862
about TIBUnInstall class 3862
TIBUnInstall Members 3863
TIBUnInstall methods 3863
TIBUnInstall properties 3864
UnInstallCheck 3864
UnInstallExecute 3864
UnInstallFile 3864
TIBUpdateSQL class 4172
about TIBUpdateSQL class 4172
Apply 4174
Create 4173
DataSet 4177

DeleteSQL 4177
Destroy 4173
ExecSQL 4175
GetDataSet 4175
GetSQL 4175
InsertSQL 4178
ModifySQL 4178
Query 4178
SetDataSet 4176
SetParams 4176
SQL 4179
SQLChanged 4176
TIBUpdateSQL Members 4172
TIBUpdateSQL methods 4174
TIBUpdateSQL properties 4176
TIBValidationService class 4001
about TIBValidationService class 4001
Create 4003
DatabaseName 4005
Destroy 4003
FetchLimboTransactionInfo 4003
FixLimboTransactionErrors 4004
GlobalAction 4005
LimboTransactionInfo 4005
LimboTransactionInfoCount 4005
Options 4006
SetServiceStartOptions 4004
TIBValidationService Members 4001
TIBValidationService methods 4003
TIBValidationService properties 4004
TIBXSQLDA class 4078
about TIBXSQLDA class 4078
AddName 4082
AsXSQLDA 4084
ByName 4082
Count 4085
Create 4079
Destroy 4079
FCount 4080
FNames 4080
FSize 4080
FSQL 4081
FUniqueRelationName 4081
FXSQLDA 4081
FXSQLVARs 4081
GetModified 4082
GetNames 4083
GetRecordSize 4083
GetXSQLDA 4083
GetXSQLVAR 4083
GetXSQLVARByName 4083
Initialize 4084
Modified 4085
Names 4085
RecordSize 4085
SetCount 4084
TIBXSQLDA fields 4080
TIBXSQLDA Members 4078
TIBXSQLDA methods 4081
TIBXSQLDA properties 4084
UniqueRelationName 4086
Vars 4086
TIBXSQLVAR class 4086
about TIBXSQLVAR class 4086
AsCurrency 4090
AsDate 4090
AsDateTime 4091
AsDouble 4091
AsFloat 4091
AsInt64 4092
AsInteger 4092
AsLong 4092
AsPointer 4092
AsQuad 4093
AsShort 4093
Assign 4088
AsString 4093
AsTime 4093

AsTrimString 4094
AsVariant 4094
AsXSQLVAR 4094
Clear 4088
Create 4087
Data 4095
Index 4095
IsNull 4095
IsNullable 4095
LoadFromFile 4089
LoadFromStream 4089
Modified 4096
Name 4096
SaveToFile 4089
SaveToStream 4089
Size 4096
SQLType 4096
TIBXSQLVAR Members 4087
TIBXSQLVAR methods 4088
TIBXSQLVAR properties 4090
Value 4097
TIcon class 2768
about TIcon class 2768
Assign 2772
Create 2770
Destroy 2772
Draw 2773
GetEmpty 2773
GetHeight 2773
GetWidth 2773
Handle 2777
HandleAllocated 2774
LoadFromClipboardFormat 2774
LoadFromResourceID 2774
LoadFromResourceName 2775
LoadFromStream 2775
ReleaseHandle 2775
SaveToClipboardFormat 2776
SaveToStream 2776
SetHeight 2776
SetSize 2777
SetTransparent 2777
SetWidth 2777
TIcon Members 2769
TIcon methods 2772
TIcon properties 2777
TIconImage class 2778
about TIconImage class 2778
Destroy 2779
FreeHandle 2779
TIconImage Members 2779
TIconImage methods 2779
TIExpert class 3
about TIExpert class 3
Execute 4
GetAuthor 4
GetComment 4
GetGlyph 4
GetIDString 5
GetMenuText 5
GetName 5
GetPage 5
GetState 6
GetStyle 6
TIExpert Members 3
TIExpert methods 3
TImage class 1048
about TImage class 1048
Align 1054
Anchors 1057
AutoSize 1058
CanAutoSize 1051
Canvas 1058
Center 1061
Constraints 1062
Create 1050
DestRect 1052
Destroy 1050

DoPaletteChange 1052
DragCursor 1062
DragKind 1063
DragMode 1063
Enabled 1065
GetPalette 1052
IncrementalDisplay 1065
OnClick 1079
OnContextPopup 1081
OnDblClick 1082
OnDragDrop 1082
OnDragOver 1085
OnEndDock 1088
OnEndDrag 1089
OnMouseActivate 1090
OnMouseDown 1091
OnMouseEnter 1093
OnMouseLeave 1093
OnMouseMove 1093
OnMouseUp 1096
OnProgress 1098
OnStartDock 1098
OnStartDrag 1099
Paint 1052
ParentShowHint 1066
Picture 1066
PopupMenu 1068
Progress 1053
Proportional 1073
ShowHint 1073
Stretch 1075
TImage events 1079
TImage Members 1049
TImage methods 1051
TImage properties 1053
Transparent 1077
Visible 1077
TInetXPageElements class 5727
about TInetXPageElements class 5727
BodyContent 5729
FormsContent 5728
IncludesContent 5728
ScriptContent 5729
StylesContent 5729
TInetXPageElements fields 5728
TInetXPageElements Members 5728
TInetXPageElements methods 5729
WarningsContent 5729
TInetXPageProducer class 5730
about TInetXPageProducer class 5730
EnableXMLIslands 5733
HTMLDoc 5733
HTMLFile 5734
IncludePathURL 5735
OnAfterGetContent 5738
OnAfterGetXMLData 5738
OnBeforeGetContent 5739
OnBeforeGetXMLData 5739
ScriptEngine 5735
Styles 5736
StylesFile 5736
TInetXPageProducer events 5737
TInetXPageProducer Members 5730
TInetXPageProducer properties 5733
WebPageItems 5737
TIniFile class 4294
about TIniFile class 4294
DeleteKey 4299
Destroy 4298
EraseSection 4299
ReadSection 4299
ReadSections 4300
ReadSectionValues 4300
ReadString 4301
TIniFile Members 4297
TIniFile methods 4298
UpdateFile 4304
WriteString 4304

TInplaceEdit class 3161
about TInplaceEdit class 3161
BoundsChanged 3163
Create 3162
CreateParams 3164
DblClick 3164
Deselect 3165
DoMouseWheel 3165
EditCanModify 3165
Grid 3171
Hide 3166
Invalidate 3166
KeyDown 3167
KeyPress 3167
KeyUp 3168
Move 3168
PosEqual 3168
SetFocus 3169
TInplaceEdit Members 3161
TInplaceEdit methods 3163
TInplaceEdit properties 3171
UpdateContents 3169
UpdateLoc 3170
Visible 3170
WndProc 3170
TInplaceEditList class 3171
about TInplaceEditList class 3171
ActiveList 3178
BoundsChanged 3173
ButtonRect 3173
ButtonWidth 3178
CloseUp 3174
Create 3172
DblClick 3174
DoDropDownKeys 3174
DoEditButtonClick 3174
DoGetPickListItems 3174
DropDown 3175
DropDownRows 3178
EditStyle 3178
KeyDown 3175
ListMouseUp 3175
ListVisible 3179
MouseDown 3175
MouseMove 3176
MouseUp 3176
OnEditButtonClick 3180
OnGetPickListitems 3180
OverButton 3176
PaintWindow 3176
PickList 3179
PickListLoaded 3179
Pressed 3179
RestoreContents 3176
StopTracking 3177
TInplaceEditList events 3179
TInplaceEditList Members 3171
TInplaceEditList methods 3173
TInplaceEditList properties 3177
TrackButton 3177
UpdateContents 3177
WndProc 3177
TInsertButton class 5385
about TInsertButton class 5385
Create 5386
TInsertButton Members 5385
TInstallOptions class 3865
about TInstallOptions class 3865
CmdLineTools 3865
ConnectivityClients 3866
Examples 3866
MainComponents 3866
TInstallOptions Members 3865
TInstallOptions properties 3865
TInvContext class 4369
about TInvContext class 4369
AllocServerData 4371
GetParamPointer 4371

GetResultPointer 4372
SetMethodInfo 4372
SetParamPointer 4373
SetResultPointer 4373
TInvContext Members 4370
TInvContext methods 4371
TInvokableClass class 4373
_AddRef 4376
_Release 4377
about TInvokableClass class 4373
AfterConstruction 4377
BeforeDestruction 4377
Create 4375
FRefCount 4375
FSOAPHeaders 4376
NewInstance 4378
QueryInterface 4378
RefCount 4379
TInvokableClass fields 4375
TInvokableClass Members 4374
TInvokableClass methods 4376
TInvokableClass properties 4379
TInvokableClassRegistry class 4379
about TInvokableClassRegistry class 4379
Create 4381
Destroy 4382
GetActionURIOfIID 4383
GetActionURIOfInfo 4384
GetClassFromIntfInfo 4384
GetExceptionInfoForInterface 4385
GetHeaderClass 4385
GetHeaderInfoForInterface 4386
GetHeaderName 4386
GetHeaderNamespace 4387
GetHeaderNamespace method 4387
GetInfoForURI 4388
GetInterfaceCount 4388
GetInterfaceExternalName 4389
GetInterfaceExternalName method 4389
GetInterfaceInfoFromName 4390
GetInterfaceTypeInfo 4390
GetIntfInvokeOptions 4391
GetIntfInvokeOptions method 4390
GetInvokableObjectFromClass 4391
GetMethExternalName 4392
GetMethInternalName 4392
GetNamespaceByGUID 4393
GetParamExternalName 4393
GetParamInternalName 4393
GetRegInterfaceEntry 4394
GetRequestHeaderInfoForInterface 4394
GetResponseHeaderInfoForInterface 4395
GetReturnParamNames 4396
GetUDDIInfo 4396
GetUDDIInfo method 4396
GetWSDLEncoding 4397
GetWSDLEncoding method 4397
HasRegInterfaceImpl 4398
RegisterAllSOAPActions 4398
RegisterDefaultSOAPAction 4399
RegisterException 4399
RegisterExceptionMethod 4400
RegisterExternalMethName 4400
RegisterExternalParamName 4401
RegisterHeaderClass 4402
RegisterHeaderClass method 4402
RegisterHeaderMethod 4403
RegisterInterface 4404
RegisterInvokableClass 4405
RegisterInvokableClass method 4405
RegisterInvokeOptions 4406
RegisterInvokeOptions method 4405
RegisterReturnParamNames 4406
RegisterUDDIInfo 4407
TInvokableClassRegistry Members 4380
TInvokableClassRegistry methods 4382
UnRegisterInterface 4407
UnRegisterInvokableClass 4408

TISAPIApplication class 4500
about TISAPIApplication class 4500
Create 4501
Destroy 4501
GetExtensionVersion 4502
HttpExtensionProc 4502
TerminateExtension 4503
TISAPIApplication Members 4501
TISAPIApplication methods 4502
TISAPIFactory class 4503
about TISAPIFactory class 4503
Create 4503
NewRequest 4504
NewResponse 4504
TISAPIFactory Members 4503
TISAPIFactory methods 4504
TISAPIRequest class 4510
about TISAPIRequest class 4510
Create 4512
ECB 4518
GetDateVariable 4513
GetFieldByName 4513
GetIntegerVariable 4514
GetStringVariable 4514
ReadClient 4515
ReadString 4515
TISAPIRequest Members 4510
TISAPIRequest methods 4512
TISAPIRequest properties 4517
TranslateURI 4516
WriteClient 4516
WriteHeaders 4516
WriteString 4517
TISAPIResponse class 4518
about TISAPIResponse class 4518
Create 4520
GetContent 4521
GetDateVariable 4521
GetIntegerVariable 4522
GetLogMessage 4522
GetStatusCode 4522
GetStringVariable 4523
InitResponse 4523
SendRedirect 4524
SendResponse 4524
SendStream 4524
Sent 4525
SetContent 4525
SetDateVariable 4525
SetIntegerVariable 4526
SetLogMessage 4526
SetStatusCode 4526
SetStringVariable 4527
TISAPIResponse Members 4519
TISAPIResponse methods 4520
tk_alias enumeration member 6246
tk_any enumeration member 6246
tk_array enumeration member 6246
tk_boolean enumeration member 6246
tk_char enumeration member 6246
tk_double enumeration member 6246
tk_enum enumeration member 6246
tk_except enumeration member 6246
tk_fixed enumeration member 6246
tk_float enumeration member 6246
tk_long enumeration member 6246
tk_longdouble enumeration member 6246
tk_longlong enumeration member 6246
tk_null enumeration member 6246
tk_objref enumeration member 6246
tk_octet enumeration member 6246
tk_Principal enumeration member 6246
tk_sequence enumeration member 6246
tk_short enumeration member 6246
tk_string enumeration member 6246
tk_struct enumeration member 6246
tk_TypeCode enumeration member 6246
tk_ulong enumeration member 6246

tk_ulonglong enumeration member 6246
tk_union enumeration member 6246
tk_ushort enumeration member 6246
tk_void enumeration member 6246
tk_wchar enumeration member 6246
tk_wstring enumeration member 6246
TKeyTipList class 6956
about TKeyTipList class 6956
Add 6958
Clear 6958
Count 6958
Create 6957
Current 6959
Destroy 6957
Full 6959
Items 6959
Partial 6960
TKeyTipList Members 6956
TKeyTipList methods 6957
TKeyTipList properties 6958
TKeyTipListEnumerator class 6960
about TKeyTipListEnumerator class 6960
Create 6961
Current 6962
GetCurrent 6961
MoveNext 6961
TKeyTipListEnumerator Members 6960
TKeyTipListEnumerator methods 6961
TKeyTipListEnumerator properties 6961
TKeyTipMatchItem class 6962
about TKeyTipMatchItem class 6962
Control 6963
Create 6963
Handled 6963
Index 6964
MatchType 6964
TKeyTipMatchItem Members 6962
TKeyTipMatchItem properties 6963
TKeyTipValidator class 6964
about TKeyTipValidator class 6964
TKeyTipValidator Members 6965
TKeyTipValidator methods 6965
Validate 6965
TLabeledEdit class 1100
about TLabeledEdit class 1100
Alignment 1103
Anchors 1104
AutoSelect 1104
AutoSize 1105
BevelEdges 1106
BevelInner 1106
BevelKind 1107
BevelOuter 1107
BiDiMode 1107
BorderStyle 1108
CharCase 1108
Color 1109
Constraints 1109
Ctl3D 1110
DoubleBuffered 1110
DragCursor 1112
DragKind 1113
DragMode 1113
EditLabel 1115
Enabled 1115
Font 1115
HideSelection 1116
ImeMode 1117
ImeName 1117
LabelPosition 1118
LabelSpacing 1118
MaxLength 1119
NumbersOnly 1119
OEMConvert 1120
OnChange 1134
OnClick 1135
OnContextPopup 1137
OnDblClick 1137

OnDragDrop 1138
OnDragOver 1141
OnEndDock 1144
OnEndDrag 1144
OnEnter 1145
OnExit 1148
OnKeyDown 1151
OnKeyPress 1154
OnKeyUp 1155
OnMouseActivate 1157
OnMouseDown 1158
OnMouseEnter 1160
OnMouseLeave 1160
OnMouseMove 1161
OnMouseUp 1163
OnStartDock 1165
OnStartDrag 1166
ParentBiDiMode 1120
ParentColor 1120
ParentCtl3D 1121
ParentDoubleBuffered 1121
ParentFont 1121
ParentShowHint 1122
PasswordChar 1123
PopupMenu 1123
ReadOnly 1128
ShowHint 1128
TabOrder 1130
TabStop 1130
Text 1131
TextHint 1132
TLabeledEdit events 1134
TLabeledEdit Members 1100
TLabeledEdit properties 1102
Visible 1132
TLastButton class 5387
about TLastButton class 5387
Create 5388
TLastButton Members 5387
TLayoutGroup class 5389
about TLayoutGroup class 5389
Custom 5391
DisplayColumns 5391
Style 5391
StyleRule 5392
TLayoutGroup Members 5389
TLayoutGroup properties 5390
TLicenseInfo class 4006
about TLicenseInfo class 4006
Create 4007
Desc 4007
Destroy 4007
Id 4008
Key 4008
LicensedUsers 4008
TLicenseInfo fields 4007
TLicenseInfo Members 4006
TLicenseMaskInfo class 4008
about TLicenseMaskInfo class 4008
CapabilityMask 4009
LicenseMask 4009
TLicenseMaskInfo fields 4009
TLicenseMaskInfo Members 4009
TLimboTransactionInfo class 4010
about TLimboTransactionInfo class 4010
Action 4011
Advise 4011
HostSite 4011
ID 4012
MultiDatabase 4012
RemoteDatabasePath 4012
RemoteSite 4012
State 4012
TLimboTransactionInfo fields 4011
TLimboTransactionInfo Members 4010
TLinkLabel class 1167
about TLinkLabel class 1167
Alignment 1169

Anchors 1169
AutoSize 1170
BevelEdges 1170
BevelInner 1171
BevelKind 1171
BevelOuter 1172
Caption 1172
Color 1175
Constraints 1175
DragCursor 1176
DragKind 1176
DragMode 1177
Enabled 1178
Font 1179
OnClick 1191
OnContextPopup 1193
OnDblClick 1194
OnDragDrop 1194
OnDragOver 1197
OnEndDock 1200
OnEndDrag 1201
OnLinkClick 1202
OnMouseActivate 1202
OnMouseDown 1203
OnMouseEnter 1205
OnMouseLeave 1205
OnMouseMove 1206
OnMouseUp 1208
OnStartDock 1210
OnStartDrag 1211
ParentColor 1180
ParentFont 1180
ParentShowHint 1181
PopupMenu 1181
ShowHint 1186
TabOrder 1188
TabStop 1188
TLinkLabel events 1191
TLinkLabel Members 1167
TLinkLabel properties 1169
UseVisualStyle 1189
Visible 1189
TListActionLink class 4550
about TListActionLink class 4550
AddItem 4552
AddItem method 4551
IsActiveLinked 4552
IsImagesLinked 4552
RefreshControl 4553
SetAction 4553
SetActive 4553
SetImages 4553
SetItemIndex 4553
TListActionLink Members 4551
TListActionLink methods 4551
TListControlAction class 37
about TListControlAction class 37
Create 38
HandlesTarget 39
IsEnabled 40
ListControl 40
Notification 40
TListControlAction Members 38
TListControlAction methods 39
TListControlAction properties 40
TListControlClearSelection class 41
about TListControlClearSelection class 41
Caption 44
Enabled 44
ExecuteTarget 42
HelpContext 44
HelpKeyword 45
HelpType 45
Hint 46
ImageIndex 46
ListControl 46
OnHint 48
OnUpdate 48

SecondaryShortCuts 47
ShortCut 47
TListControlClearSelection events 48
TListControlClearSelection Members 41
TListControlClearSelection methods 42
TListControlClearSelection properties 43
UpdateTarget 43
Visible 47
TListControlCopySelection class 50
about TListControlCopySelection class 50
Caption 53
Destination 53
Enabled 54
ExecuteTarget 51
HandlesTarget 52
HelpContext 54
HelpKeyword 55
HelpType 55
Hint 56
ImageIndex 56
ListControl 56
OnHint 58
OnUpdate 58
SecondaryShortCuts 57
ShortCut 57
TListControlCopySelection events 58
TListControlCopySelection Members 50
TListControlCopySelection methods 51
TListControlCopySelection properties 53
UpdateTarget 52
Visible 57
TListControlDeleteSelection class 60
about TListControlDeleteSelection class 60
Caption 62
Enabled 63
ExecuteTarget 61
HelpContext 63
HelpKeyword 64
HelpType 64
Hint 65
ImageIndex 65
ListControl 65
OnHint 67
OnUpdate 67
SecondaryShortCuts 66
ShortCut 66
TListControlDeleteSelection events 67
TListControlDeleteSelection Members 60
TListControlDeleteSelection methods 61
TListControlDeleteSelection properties 62
UpdateTarget 62
Visible 66
TListControlItem class 4554
about TListControlItem class 4554
Assign 4556
Caption 4559
Changed 4557
Create 4555
Data 4560
FCaption 4555
FData 4556
FImageIndex 4556
GetDisplayName 4558
ImageIndex 4560
SetCaption 4558
SetData 4558
SetImageIndex 4559
TListControlItem fields 4555
TListControlItem Members 4554
TListControlItem methods 4556
TListControlItem properties 4559
TListControlItems class 4560
about TListControlItems class 4560
Add 4562
CaseSensitive 4565
CompareItems 4563
Create 4562
CustomSort 4563

Items 4565
OnCompare 4567
Sort 4564
SortType 4566
TListControlItems events 4567
TListControlItems Members 4561
TListControlItems methods 4562
TListControlItems properties 4565
Update 4564
TListControlMoveSelection class 68
about TListControlMoveSelection class 68
ExecuteTarget 70
TListControlMoveSelection Members 69
TListControlMoveSelection methods 70
TListControlSelectAll class 71
about TListControlSelectAll class 71
Caption 74
Enabled 74
ExecuteTarget 72
HelpContext 74
HelpKeyword 75
HelpType 75
Hint 76
ImageIndex 76
ListControl 76
OnHint 78
OnUpdate 78
SecondaryShortCuts 77
ShortCut 77
TListControlSelectAll events 78
TListControlSelectAll Members 71
TListControlSelectAll methods 72
TListControlSelectAll properties 73
UpdateTarget 73
Visible 77
TLocalAppServer class 6546
about TLocalAppServer class 6546
AS_ApplyUpdates 6548
AS_DataRequest 6549
AS_Execute 6549
AS_GetParams 6550
AS_GetProviderNames 6551
AS_GetRecords 6552
AS_RowRequest 6553
Create 6547
Create constructor 6547
Destroy 6547
GetIDsOfNames 6554
GetTypeInfo 6554
GetTypeInfoCount 6554
InterfaceSupportsErrorInfo 6554
Invoke 6555
SafeCallException 6555
TLocalAppServer Members 6546
TLocalAppServer methods 6548
TMainMenu class 4828
about TMainMenu class 4828
AutoHotkeys 4833
AutoLineReduction 4834
AutoMerge 4834
BiDiMode 4835
GetHandle 4830
GetOle2AcceleratorTable 4830
Images 4835
MenuChanged 4831
Merge 4831
OnChange 4837
OwnerDraw 4836
ParentBiDiMode 4836
PopulateOle2Menu 4832
SetOle2MenuHandle 4832
TMainMenu events 4836
TMainMenu Members 4828
TMainMenu methods 4829
TMainMenu properties 4833
Unmerge 4832
TMask class 4684
about TMask class 4684

Create 4685
Destroy 4686
Matches 4686
TMask Members 4685
TMask methods 4686
TMaskEdit class 4610
about TMaskEdit class 4610
Align 4614
Alignment 4617
Anchors 4618
AutoSelect 4618
AutoSize 4619
BevelEdges 4620
BevelInner 4620
BevelKind 4620
BevelOuter 4621
BevelWidth 4621
BiDiMode 4622
BorderStyle 4622
CharCase 4622
Color 4623
Constraints 4624
Ctl3D 4624
DoubleBuffered 4625
DragCursor 4627
DragKind 4627
DragMode 4627
EditMask 4629
Enabled 4631
Font 4631
ImeMode 4632
ImeName 4633
MaxLength 4633
OnChange 4649
OnClick 4649
OnDblClick 4651
OnDragDrop 4652
OnDragOver 4655
OnEndDock 4658
OnEndDrag 4658
OnEnter 4659
OnExit 4662
OnKeyDown 4664
OnKeyPress 4668
OnKeyUp 4669
OnMouseActivate 4671
OnMouseDown 4672
OnMouseEnter 4674
OnMouseLeave 4674
OnMouseMove 4675
OnMouseUp 4677
OnStartDock 4679
OnStartDrag 4680
ParentBiDiMode 4634
ParentColor 4634
ParentCtl3D 4635
ParentDoubleBuffered 4635
ParentFont 4635
ParentShowHint 4636
PasswordChar 4636
PopupMenu 4637
ReadOnly 4642
ShowHint 4642
TabOrder 4644
TabStop 4644
Text 4645
TextHint 4647
TMaskEdit events 4648
TMaskEdit Members 4610
TMaskEdit properties 4613
Visible 4647
tmAuto enumeration member 2874
TMediaPlayer class 5749
about TMediaPlayer class 5749
Anchors 5776
AutoButtonSet 5756
AutoEnable 5777
AutoOpen 5780

AutoRewind 5781
Back 5757
Capabilities 5781
Click 5757
Close 5758
ColoredButtons 5781
Constraints 5782
Create 5755
Destroy 5755
DeviceID 5782
DeviceType 5783
Display 5783
DisplayRect 5784
DoNotify 5759
DoubleBuffered 5784
Eject 5759
Enabled 5786
EnabledButtons 5786
EndPos 5789
Error 5789
ErrorMessage 5790
FileName 5791
Frames 5793
KeyDown 5761
Length 5794
Loaded 5761
MMNotify 5762
Mode 5794
Next 5762
Notification 5763
Notify 5794
NotifyValue 5795
OnClick 5810
OnContextPopup 5811
OnEnter 5811
OnExit 5814
OnMouseEnter 5817
OnMouseLeave 5817
OnNotify 5817
OnPostClick 5818
Open 5763
Paint 5766
ParentDoubleBuffered 5795
ParentShowHint 5796
Pause 5766
PauseOnly 5767
Play 5767
PopupMenu 5796
Position 5801
PostClick 5770
Previous 5771
Resume 5771
Rewind 5772
Save 5772
Shareable 5802
ShowHint 5802
Start 5804
StartPos 5804
StartRecording 5773
Step 5774
Stop 5774
TabOrder 5804
TabStop 5805
TimeFormat 5805
TMediaPlayer events 5810
TMediaPlayer Members 5753
TMediaPlayer methods 5756
TMediaPlayer properties 5775
TrackLength 5806
TrackPosition 5806
Tracks 5807
Updated 5775
Visible 5807
VisibleButtons 5809
Wait 5809
TMemIniFile class 4307
about TMemIniFile class 4307
CaseSensitive 4324

Clear 4313
Create 4311
Create constructor 4311
DeleteKey 4313
Destroy 4312
Encoding 4324
EraseSection 4314
GetStrings 4314
ReadSection 4314
ReadSections 4315
ReadSectionValues 4315
ReadString 4316
Rename 4319
SetStrings 4319
TMemIniFile Members 4310
TMemIniFile methods 4312
TMemIniFile properties 4323
UpdateFile 4320
WriteString 4320
TMenu class 4837
about TMenu class 4837
AdjustBiDiBehavior 4841
AutoHotkeys 4849
AutoLineReduction 4850
BiDiMode 4850
Create 4839
CreateMenuItem 4841
Destroy 4840
DispatchCommand 4841
DispatchPopup 4842
DoBiDiModeChanged 4842
DoChange 4842
DoGetMenuString 4843
FindItem 4843
GetChildren 4844
GetHandle 4844
GetHelpContext 4844
Handle 4851
Images 4851
IsOwnerDraw 4845
IsRightToLeft 4845
IsShortCut 4845
Items 4852
Loaded 4846
MenuChanged 4846
Notification 4847
OnChange 4858
OwnerDraw 4856
ParentBiDiMode 4857
ParentBiDiModeChanged 4847
ParentBiDiModeChanged method 4847
ProcessMenuChar 4848
SetChildOrder 4848
TMenu events 4857
TMenu Members 4838
TMenu methods 4840
TMenu properties 4849
UpdateItems 4849
WindowHandle 4857
TMenuActionLink class 4858
about TMenuActionLink class 4858
AssignClient 4860
FClient 4860
IsAutoCheckLinked 4861
IsCaptionLinked 4861
IsCheckedLinked 4862
IsEnabledLinked 4862
IsGroupIndexLinked 4862
IsHelpContextLinked 4863
IsHintLinked 4863
IsImageIndexLinked 4864
IsOnExecuteLinked 4864
IsShortCutLinked 4864
IsVisibleLinked 4865
SetAutoCheck 4865
SetCaption 4866
SetChecked 4866
SetEnabled 4866

SetHelpContext 4867
SetHint 4867
SetImageIndex 4867
SetOnExecute 4868
SetShortCut 4868
SetVisible 4869
TMenuActionLink fields 4859
TMenuActionLink Members 4859
TMenuActionLink methods 4860
TMenuItem class 4869
about TMenuItem class 4869
Action 4899
ActionChange 4875
ActionLink 4899
Add 4875
Add method 4875
AdvancedDrawItem 4880
AssignTo 4880
AutoCheck 4899
AutoHotkeys 4900
AutoLineReduction 4900
Bitmap 4901
Break 4901
Caption 4902
Checked 4906
Clear 4881
Click 4881
Command 4907
Count 4907
Create 4872
Default 4912
DefineProperties 4881
Delete 4882
Destroy 4873
DoDrawText 4882
DrawItem 4883
Enabled 4912
Find 4883
GetActionLinkClass 4883
GetAutoHotkeys 4884
GetAutoLineReduction 4884
GetChildren 4884
GetCount 4884
GetEnumerator 4884
GetHandle 4885
GetImageList 4885
GetItem 4886
GetMenuIndex 4886
GetParentComponent 4886
GetParentMenu 4886
GroupIndex 4914
Handle 4915
HasParent 4887
HelpContext 4915
Hint 4916
ImageIndex 4916
IndexOf 4887
InitiateAction 4887
Insert 4888
InsertNewLine 4889
InsertNewLineAfter 4889
InsertNewLineBefore 4890
IsLine 4890
Items 4917
Loaded 4891
MeasureItem 4891
MenuChanged 4892
MenuIndex 4919
NewBottomLine 4892
NewTopLine 4892
Notification 4893
OnAdvancedDrawItem 4922
OnClick 4922
OnDrawItem 4928
OnMeasureItem 4929
Parent 4919
RadioItem 4920
Remove 4893

RethinkHotkeys 4894
RethinkLines 4894
SetBreak 4895
SetCaption 4895
SetChecked 4895
SetChildOrder 4896
SetDefault 4896
SetEnabled 4896
SetGroupIndex 4896
SetImageIndex 4897
SetMenuIndex 4897
SetParentComponent 4897
SetRadioItem 4897
SetShortCut 4897
SetVisible 4898
ShortCut 4920
SubMenuImages 4921
TMenuItem events 4922
TMenuItem Members 4870
TMenuItem methods 4874
TMenuItem properties 4898
UpdateItems 4898
Visible 4921
TMenuItemEnumerator class 4929
about TMenuItemEnumerator class 4929
Create 4930
Current 4931
GetCurrent 4930
MoveNext 4930
TMenuItemEnumerator Members 4930
TMenuItemEnumerator methods 4930
TMenuItemEnumerator properties 4931
TMenuItemStack class 4931
about TMenuItemStack class 4931
ClearItem 4932
TMenuItemStack Members 4931
TMenuItemStack methods 4932
TMetafile class 2780
about TMetafile class 2780
Assign 2785
Clear 2785
Create 2782
CreatedBy 2792
Description 2792
Destroy 2783
Draw 2785
Enhanced 2793
GetEmpty 2786
GetHeight 2786
GetPalette 2786
GetWidth 2786
Handle 2793
HandleAllocated 2786
Inch 2793
LoadFromClipboardFormat 2787
LoadFromStream 2787
MMHeight 2793
MMWidth 2794
ReadData 2787
ReadEMFStream 2788
ReadWMFStream 2788
ReleaseHandle 2788
SaveToClipboardFormat 2788
SaveToFile 2789
SaveToStream 2789
SetHeight 2790
SetSize 2790
SetTransparent 2790
SetWidth 2790
TestEMF 2791
TMetafile Members 2780
TMetafile methods 2784
TMetafile properties 2791
WriteData 2791
WriteEMFStream 2791
WriteWMFStream 2791
TMetafileCanvas class 2794
about TMetafileCanvas class 2794

Create 2796
CreateWithComment 2798
Destroy 2799
TMetafileCanvas Members 2795
TMetafileImage class 2799
about TMetafileImage class 2799
Destroy 2800
FreeHandle 2800
TMetafileImage Members 2800
TMetafileImage methods 2800
tmFixed enumeration member 2874
tmHide enumeration member 2593
TMidasConnection class 5234
about TMidasConnection class 5234
ConnectType 5238
Create 5235
DoConnect 5236
DoDisconnect 5237
ServerPort 5238
SetConnectType 5237
SetUseBroker 5237
TMidasConnection Members 5234
TMidasConnection methods 5236
TMidasConnection properties 5237
UseBroker 5239
TMidasPageProducer class 5740
about TMidasPageProducer class 5740
TMonitor class 2439
about TMonitor class 2439
BoundsRect 2441
Handle 2441
Height 2442
Left 2442
MonitorNum 2443
Primary 2443
TMonitor Members 2440
TMonitor properties 2440
Top 2443
Width 2444
WorkareaRect 2444
tmShow enumeration member 2593
TMtsAutoObject class 5832
about TMtsAutoObject class 5832
Activate 5833
CanBePooled 5834
Deactivate 5834
DisableCommit 5834
EnableCommit 5835
IsCallerInRole 5835
IsInTransaction 5836
IsSecurityEnabled 5836
ObjectContext 5838
OnActivate 5836
OnDeactivate 5837
Pooled 5838
SetAbort 5837
SetComplete 5837
TMtsAutoObject Members 5832
TMtsAutoObject methods 5833
TMtsAutoObject properties 5838
TMtsDataModule class 5840
about TMtsDataModule class 5840
Activate 5842
AS_ApplyUpdates 5843
AS_DataRequest 5843
AS_Execute 5844
AS_GetParams 5845
AS_GetProviderNames 5846
AS_GetRecords 5846
AS_RowRequest 5847
AutoComplete 5853
CallComplete 5848
CanBePooled 5848
Create 5841
Deactivate 5848
DisableCommit 5849
EnableCommit 5849
IsCallerInRole 5850

IsInTransaction 5850
IsSecurityEnabled 5851
ObjectContext 5853
OnActivate 5854
OnDeactivate 5854
Pooled 5853
SetAbort 5851
SetComplete 5852
TMtsDataModule events 5854
TMtsDataModule Members 5841
TMtsDataModule methods 5842
TMtsDataModule properties 5852
TMultipartContentParser class 6706
about TMultipartContentParser class 6706
BufferContent 6708
CanParse 6708
Destroy 6707
ExtractContentTypeFields 6709
ExtractMultipartContentFields 6709
GetContentFields 6709
GetFiles 6709
ParseMultiPart 6710
ParseMultiPartContent 6710
ParseMultipartHeaders 6710
TMultipartContentParser Members 6707
TMultipartContentParser methods 6707
TNextButton class 5392
about TNextButton class 5392
Create 5394
TNextButton Members 5393
TNextPageButton class 5394
about TNextPageButton class 5394
Create 5396
TNextPageButton Members 5395
TNextTab class 80
about TNextTab class 80
AfterTabChange 89
BeforeTabChange 89
Caption 83
Enabled 83
ExecuteTarget 81
HelpContext 84
HelpKeyword 84
HelpType 85
Hint 85
ImageIndex 85
LastTabCaption 86
OnFinish 90
OnHint 90
OnUpdate 90
OnValidateTab 92
ShortCut 86
SkipHiddenTabs 87
TabControl 87
TNextTab events 88
TNextTab Members 80
TNextTab methods 81
TNextTab properties 83
UpdateTarget 82
Visible 88
Wrap 88
TNotebook class 1211
about TNotebook class 1211
ActivePage 1217
Align 1218
Anchors 1221
Color 1222
Constraints 1222
Create 1213
CreateParams 1214
Ctl3D 1223
Destroy 1214
DoubleBuffered 1223
DragCursor 1225
DragKind 1226
DragMode 1226
Enabled 1228
Font 1228

GetChildOwner 1215
GetChildren 1215
OnClick 1242
OnContextPopup 1244
OnDblClick 1244
OnDragDrop 1245
OnDragOver 1248
OnEndDock 1251
OnEndDrag 1251
OnEnter 1252
OnExit 1255
OnMouseActivate 1257
OnMouseDown 1258
OnMouseEnter 1261
OnMouseLeave 1261
OnMouseMove 1261
OnMouseUp 1263
OnPageChanged 1266
OnStartDock 1266
OnStartDrag 1267
PageIndex 1229
Pages 1230
ParentColor 1230
ParentCtl3D 1230
ParentDoubleBuffered 1231
ParentFont 1231
ParentShowHint 1232
PopupMenu 1232
ReadState 1216
ShowControl 1216
ShowHint 1237
TabOrder 1239
TabStop 1239
TNotebook events 1241
TNotebook Members 1212
TNotebook methods 1214
TNotebook properties 1216
Visible 1240
TObjectDispatch class 5926
about TObjectDispatch class 5926
Create 5927
Destroy 5927
GetIDsOfNames 5928
GetMethodInfo 5928
GetObjectDispatch 5928
GetPropInfo 5929
GetTypeInfo 5929
GetTypeInfoCount 5929
Instance 5930
Invoke 5929
TObjectDispatch Members 5926
TObjectDispatch methods 5927
TObjectDispatch properties 5929
TOffice2007Button class 6826
about TOffice2007Button class 6826
CMMouseenter 6829
CMMouseleave 6829
Create 6827
Destroy 6828
DoClick 6829
DrawBackground 6829
DrawImage 6830
MouseDown 6830
MouseInControl 6832
MouseMove 6830
MouseUp 6831
OnClick 6833
Paint 6831
RibbonStyle 6833
SetParent 6832
TOffice2007Button events 6833
TOffice2007Button Members 6827
TOffice2007Button methods 6828
TOffice2007Button properties 6832
TOleContainer class 5969
_AddRef 5974
_Release 5975
about TOleContainer class 5969

Align 5991
AllowActiveDoc 5995
AllowInPlace 5995
Anchors 5996
AutoActivate 5996
AutoVerbMenu 5997
BorderStyle 5997
CanPaste 5997
Caption 5998
Changed 5975
ChangeIconDialog 5976
Close 5976
Color 6000
Constraints 6001
Copy 5976
CopyOnSave 6002
Create 5972
CreateLinkToFile 5977
CreateObject 5977
CreateObjectFromFile 5978
CreateObjectFromInfo 5979
CreateParams 5979
Ctl3D 6002
DblClick 5980
DefineProperties 5980
Destroy 5973
DestroyObject 5981
DoEnter 5981
DoVerb 5981
DragCursor 6003
DragMode 6003
Enabled 6005
GetIconMetaPict 5982
GetPopupMenu 5982
Iconic 6005
InsertObjectDialog 5982
KeyDown 5983
Linked 6005
LoadFromFile 5984
LoadFromStream 5984
Modified 6006
MouseDown 5984
NewInserted 6006
ObjectPropertiesDialog 5985
ObjectVerbs 6006
OldStreamFormat 6007
OleClassName 6007
OleObject 6008
OleObjectInterface 6008
OnActivate 6021
OnDeactivate 6022
OnDragDrop 6022
OnDragOver 6025
OnEndDrag 6028
OnEnter 6029
OnExit 6032
OnKeyDown 6035
OnKeyPress 6038
OnKeyUp 6039
OnMouseActivate 6041
OnMouseDown 6042
OnMouseMove 6044
OnMouseUp 6047
OnObjectMove 6049
OnResize 6049
OnStartDrag 6049
Paint 5985
ParentColor 6008
ParentCtl3D 6009
ParentShowHint 6009
Paste 5986
PasteSpecialDialog 5986
PopupMenu 6010
PrimaryVerb 6015
QueryInterface 5987
Run 5987
SaveAsDocument 5987
SaveToFile 5989

SaveToStream 5989
ShowHint 6015
SizeMode 6017
SourceDoc 6017
State 6017
StorageInterface 6018
TabOrder 6018
TabStop 6019
TOleContainer events 6021
TOleContainer Members 5970
TOleContainer methods 5973
TOleContainer properties 5990
UpdateObject 5990
UpdateVerbs 5990
Visible 6019
TOleControl class 6074
_AddRef 6081
_Release 6081
_SetColorProp 6082
_SetTColorProp 6082
_SetTFontProp 6082
_SetTOleEnumProp 6083
_SetTPictureProp 6083
about TOleControl class 6074
Anchors 6119
BrowseProperties 6083
CanInPlaceActivate 6083
ContextSensitiveHelp 6083
ControlData 6119
Create 6077
CreateWnd 6084
DeactivateAndUndo 6084
DefaultDispatch 6120
DefaultHandler 6084
DefineProperties 6085
Destroy 6078
DestroyWindowHandle 6085
DiscardUndoState 6086
DoObjectVerb 6086
EnableModeless 6086
GetBorder 6087
GetByteProp 6087
GetColorProp 6087
GetCompProp 6087
GetContainer 6087
GetCurrencyProp 6088
GetDoubleProp 6088
GetEnumPropDesc 6088
GetExtendedControl 6088
GetHelpContext 6089
GetIDispatchProp 6089
GetIDsOfNames 6089
GetIntegerProp 6089
GetIUnknownProp 6090
GetMoniker 6090
GetObjectVerbs 6090
GetOleBoolProp 6090
GetOleDateProp 6091
GetOleEnumProp 6091
GetOleVariantProp 6091
GetPropDisplayString 6091
GetPropDisplayStrings 6092
GetProperty 6092
GetShortIntProp 6092
GetSingleProp 6092
GetSmallintProp 6093
GetStringProp 6093
GetTColorProp 6093
GetTDateTimeProp 6093
GetTFontProp 6094
GetTOleEnumProp 6094
GetTPictureProp 6094
GetTypeInfo 6094
GetTypeInfoCount 6094
GetVariantProp 6095
GetWideStringProp 6095
GetWindowContext 6095
GetWordBoolProp 6095

GetWordProp 6096
InitControlData 6096
InitControlInterface 6096
InsertMenus 6096
Invoke 6096
InvokeEvent 6097
InvokeMethod 6097
IsCustomProperty 6097
IsPropPageProperty 6097
LockInPlaceActive 6098
OleControlSite_TranslateAccelerator 6098
OleInPlaceFrame_GetWindow 6098
OleInPlaceFrame_TranslateAccelerator 6098
OleInPlaceSite_GetWindow 6099
OleObject 6120
OnChanged 6099
OnControlInfoChanged 6099
OnFocus 6099
OnInPlaceActivate 6100
OnInPlaceDeactivate 6100
OnPosRectChange 6100
OnRequestEdit 6100
OnShowWindow 6100
OnUIActivate 6101
OnUIDeactivate 6101
PaletteChanged 6101
PerPropBrowsing 6120
PictureChanged 6102
PostMessageFilter 6102
PreMessageFilter 6102
QueryInterface 6103
QueryService 6103
RemoveMenus 6103
RequestBorderSpace 6104
RequestNewObjectLayout 6104
SaveObject 6104
Scroll 6104
ServiceQuery 6121
SetActiveObject 6104
SetBorderSpace 6105
SetBounds 6105
SetByteProp 6105
SetColorProp 6105
SetCompProp 6106
SetCurrencyProp 6106
SetDoubleProp 6106
SetIDispatchProp 6106
SetIntegerProp 6107
SetIUnknownProp 6107
SetMenu 6107
SetName 6107
SetOleBoolProp 6108
SetOleDateProp 6108
SetOleEnumProp 6108
SetOleVariantProp 6108
SetParent 6108
SetPropDisplayString 6109
SetProperty 6109
SetShortIntProp 6110
SetSingleProp 6110
SetSmallintProp 6110
SetStatusText 6110
SetStringProp 6111
SetTColorProp 6111
SetTDateTimeProp 6111
SetTFontProp 6111
SetTOleEnumProp 6111
SetTPictureProp 6112
SetVariantProp 6112
SetWideStringProp 6112
SetWordBoolProp 6112
SetWordProp 6113
ShowAboutBox 6113
ShowObject 6113
ShowPropertyFrame 6113
StandardEvent 6113
SuppressException 6114
TabStop 6121

TOleControl events 6121
TOleControl Members 6074
TOleControl methods 6078
TOleControl properties 6119
TransformCoords 6114
WndProc 6114
TOleForm class 6050
about TOleForm class 6050
Create 6051
Destroy 6051
TOleForm Members 6051
TOLEnterpriseConnection class 4811
about TOLEnterpriseConnection class 4811
BrokerName 4813
ComputerName 4814
DoConnect 4813
TOLEnterpriseConnection Members 4812
TOLEnterpriseConnection methods 4813
TOLEnterpriseConnection properties 4813
TOleServer class 6132
_AddRef 6135
_Release 6136
about TOleServer class 6132
AutoConnect 6141
Connect 6136
ConnectEvents 6137
ConnectKind 6142
Create 6133
Destroy 6134
Disconnect 6137
DisconnectEvents 6137
EventDispatch 6143
GetAutoConnect 6138
GetConnectKind 6138
GetServer 6138
InitServerData 6139
InvokeEvent 6139
Loaded 6140
QueryInterface 6140
RemoteMachineName 6143
ServerData 6143
SetAutoConnect 6140
SetConnectKind 6141
TOleServer Members 6132
TOleServer methods 6135
TOleServer properties 6141
TOpenPicture class 92
about TOpenPicture class 92
Caption 94
Dialog 94
Enabled 95
GetDialogClass 94
HelpContext 95
HelpKeyword 96
HelpType 96
Hint 97
ImageIndex 97
OnAccept 99
OnCancel 99
OnHint 99
OnUpdate 99
SecondaryShortCuts 97
ShortCut 98
TOpenPicture events 98
TOpenPicture Members 93
TOpenPicture methods 93
TOpenPicture properties 94
Visible 98
TOpenPictureDialog class 1560
about TOpenPictureDialog class 1560
Create 1562
DoClose 1562
DoSelectionChange 1563
DoShow 1563
Execute 1564
Filter 1566
ImageCtrl 1569
PictureLabel 1570

PreviewClick 1565
TOpenPictureDialog Members 1561
TOpenPictureDialog methods 1562
TOpenPictureDialog properties 1565
TOpenTextFileDialog class 1570
about TOpenTextFileDialog class 1570
Create 1571
Destroy 1571
DoClose 1572
DoShow 1572
EncodingIndex 1576
Encodings 1576
Execute 1573
TOpenTextFileDialog Members 1570
TOpenTextFileDialog methods 1571
TOpenTextFileDialog properties 1576
TOptionItem class 7294
about TOptionItem class 7294
Action 7296
Assign 7295
Caption 7297
GetDisplayName 7296
Hint 7297
KeyTip 7297
Tag 7297
TOptionItem Members 7294
TOptionItem methods 7295
TOptionItem properties 7296
TOptionItems class 7298
about TOptionItems class 7298
Add 7298
ApplicationMenu 7299
Insert 7299
Items 7300
TOptionItems Members 7298
TOptionItems methods 7298
TOptionItems properties 7299
TOPToSoapDomConvert class 6167
about TOPToSoapDomConvert class 6167
Attachments 6176
Create 6171
Destroy 6171
Encoding 6177
InvContextToMsg 6172
MakeFault 6172
MakeResponse 6173
MsgToInvContext 6174
MsgToInvContext method 6173
NewXMLDocument 6174
ProcessResponse 6175
ProcessResponse method 6175
TempDir 6177
TOPToSoapDomConvert Members 6168
TOPToSoapDomConvert methods 6171
TOPToSoapDomConvert properties 6176
WSDLView 6178
TOutline class 6296
about TOutline class 6296
Align 6302
BorderStyle 6305
Color 6306
Ctl3D 6306
DoubleBuffered 6307
DragCursor 6309
DragKind 6309
DragMode 6309
Enabled 6311
Font 6311
ItemHeight 6313
ItemSeparator 6313
Lines 6313
OnClick 6330
OnCollapse 6332
OnContextPopup 6332
OnDblClick 6333
OnDragDrop 6333
OnDragOver 6336
OnDrawItem 6340

OnEndDock 6340
OnEndDrag 6341
OnEnter 6342
OnExit 6344
OnExpand 6347
OnKeyDown 6347
OnKeyPress 6351
OnKeyUp 6352
OnMouseActivate 6354
OnMouseDown 6355
OnMouseEnter 6357
OnMouseLeave 6357
OnMouseMove 6357
OnMouseUp 6360
OnStartDock 6362
OnStartDrag 6362
Options 6314
OutlineStyle 6315
ParentColor 6315
ParentCtl3D 6316
ParentDoubleBuffered 6316
ParentFont 6316
ParentShowHint 6317
PictureClosed 6318
PictureLeaf 6318
PictureMinus 6318
PictureOpen 6319
PicturePlus 6319
PopupMenu 6320
ScrollBars 6325
ShowHint 6325
Style 6327
TabOrder 6327
TabStop 6328
TOutline events 6330
TOutline Members 6296
TOutline properties 6301
Visible 6328
TOutlineNode class 6363
about TOutlineNode class 6363
AddNode 6366
ChangeLevelBy 6367
Collapse 6367
Create 6365
Data 6374
Destroy 6365
Expand 6368
ExpandCount 6374
Expanded 6375
FullExpand 6368
FullPath 6375
GetDataItem 6368
GetDisplayWidth 6369
getFirstChild 6369
GetLastChild 6369
GetNextChild 6370
GetNodeAtIndex 6370
GetPrevChild 6370
GetRowOfNode 6371
GetTextItem 6371
GetVisibleNode 6371
HasAsParent 6372
HasItems 6376
Index 6376
InsertNode 6372
InternalRemove 6372
IsVisible 6376
Items 6377
Level 6377
List 6378
MoveTo 6372
Outline 6378
Parent 6378
Remove 6373
Text 6378
TopItem 6379
TOutlineNode Members 6364
TOutlineNode methods 6366

TOutlineNode properties 6374
WriteNode 6373
WriteNode method 6373
TPacketDataSet class 6555
about TPacketDataSet class 6555
AssignCurValues 6558
AssignCurValues method 6557
Create 6556
CreateFromDelta 6558
DataEvent 6558
GetStateFieldValue 6559
HasCurValues 6559
HasMergeConflicts 6559
InitAltRecBuffers 6560
InternalClose 6560
InternalInitRecord 6560
InternalOpen 6561
NewValuesModified 6562
SetFieldData 6561
SetStateFieldValue 6561
StreamMetaData 6562
TPacketDataSet Members 6556
TPacketDataSet methods 6557
TPacketDataSet properties 6562
UpdateKind 6561
UseCurValues 6562
WritingCurValues 6563
TPage class 1267
about TPage class 1267
Caption 1270
Create 1268
Height 1273
OnAlignInsertBefore 1278
OnAlignPosition 1278
Paint 1269
ReadState 1269
TabOrder 1274
TPage events 1278
TPage Members 1268
TPage methods 1269
TPage properties 1270
Visible 1274
Width 1276
TPageItemsProducer class 6394
about TPageItemsProducer class 6394
Create 6395
Destroy 6395
GetTemplateStream 6396
HTMLDoc 6396
HTMLFile 6397
TPageItemsProducer Members 6394
TPageItemsProducer methods 6395
TPageItemsProducer properties 6396
TPageProducer class 3496
about TPageProducer class 3496
HTMLDoc 3498
HTMLFile 3499
OnHTMLTag 3500
ScriptEngine 3500
StripParamQuotes 3500
TPageProducer events 3500
TPageProducer Members 3497
TPageProducer properties 3498
TPaintBox class 1279
about TPaintBox class 1279
Align 1282
Anchors 1286
Canvas 1286
Color 1288
Constraints 1289
Create 1281
DragCursor 1290
DragKind 1290
DragMode 1290
Enabled 1292
Font 1292
OnClick 1304
OnContextPopup 1305

OnDblClick 1306
OnDragDrop 1306
OnDragOver 1309
OnEndDock 1313
OnEndDrag 1313
OnMouseActivate 1314
OnMouseDown 1315
OnMouseEnter 1317
OnMouseLeave 1318
OnMouseMove 1318
OnMouseUp 1320
OnPaint 1322
OnStartDock 1324
OnStartDrag 1325
Paint 1281
ParentColor 1293
ParentFont 1294
ParentShowHint 1294
PopupMenu 1295
ShowHint 1300
TPaintBox events 1303
TPaintBox Members 1280
TPaintBox methods 1281
TPaintBox properties 1282
Visible 1301
TPanel class 1326
about TPanel class 1326
Align 1330
Alignment 1333
Anchors 1334
AutoSize 1334
BevelEdges 1335
BevelInner 1335
BevelKind 1336
BevelOuter 1336
BevelWidth 1337
BiDiMode 1339
BorderStyle 1339
BorderWidth 1339
Caption 1341
Color 1343
Constraints 1344
Ctl3D 1345
DockManager 1345
DockSite 1347
DoubleBuffered 1348
DragCursor 1350
DragKind 1350
DragMode 1351
Enabled 1352
Font 1353
FullRepaint 1354
Locked 1354
OnAlignInsertBefore 1368
OnAlignPosition 1369
OnCanResize 1370
OnClick 1370
OnConstrainedResize 1372
OnContextPopup 1372
OnDblClick 1373
OnDockDrop 1373
OnDockOver 1375
OnDragDrop 1378
OnDragOver 1381
OnEndDock 1384
OnEndDrag 1384
OnEnter 1385
OnExit 1388
OnGetSiteInfo 1390
OnMouseActivate 1393
OnMouseDown 1394
OnMouseEnter 1396
OnMouseLeave 1396
OnMouseMove 1396
OnMouseUp 1399
OnResize 1401
OnStartDock 1402
OnStartDrag 1402

OnUnDock 1403
Padding 1354
ParentBackground 1355
ParentBiDiMode 1355
ParentColor 1355
ParentCtl3D 1356
ParentDoubleBuffered 1356
ParentFont 1356
ParentShowHint 1357
PopupMenu 1358
ShowCaption 1363
ShowHint 1363
TabOrder 1364
TabStop 1365
TPanel events 1368
TPanel Members 1327
TPanel properties 1329
UseDockManager 1365
VerticalAlignment 1366
Visible 1366
TPanningWindow class 4235
about TPanningWindow class 4235
Create 4236
CreateParams 4237
Destroy 4236
DoUpdate 4238
GetIsPanning 4238
GetPanOptions 4238
IsPanning 4245
MouseMove 4239
OnUpdate 4246
Paint 4239
PanControl 4245
PanInterval 4246
PanOptions 4246
SetPanCursor 4240
StartPanning 4240
StopPanning 4240
TPanningWindow events 4246
TPanningWindow Members 4235
TPanningWindow methods 4237
TPanningWindow properties 4245
UpdateScroll 4240
WndProc 4241
TPen class 2801
about TPen class 2801
Assign 2803
Color 2806
Create 2802
Destroy 2802
GetColor 2804
GetHandle 2804
GetStyle 2804
GetWidth 2804
Handle 2807
Mode 2808
SetColor 2805
SetHandle 2805
SetMode 2805
SetStyle 2805
SetWidth 2805
Style 2808
TPen Members 2801
TPen methods 2803
TPen properties 2806
Width 2809
TPenRecall class 2811
about TPenRecall class 2811
Create 2811
TPenRecall Members 2811
TPicture class 2812
about TPicture class 2812
Assign 2815
AssignTo 2816
Bitmap 2825
Changed 2817
Create 2814
DefineProperties 2817

Destroy 2814
Graphic 2827
Height 2828
Icon 2828
LoadFromClipboardFormat 2818
LoadFromFile 2818
LoadFromStream 2819
Metafile 2828
OnChange 2830
OnProgress 2830
PictureAdapter 2829
Progress 2819
RegisterClipboardFormat 2819
RegisterFileFormat 2821
RegisterFileFormatRes 2822
SaveToClipboardFormat 2822
SaveToFile 2823
SaveToStream 2824
SupportsClipboardFormat 2824
TPicture events 2829
TPicture Members 2813
TPicture methods 2814
TPicture properties 2825
UnregisterGraphicClass 2825
Width 2829
TPlatformDefaultStyleActionBars class 6400
about TPlatformDefaultStyleActionBars class 6400
GetAddRemoveItemClass 6401
GetColorMapClass 6401
GetControlClass 6401
GetPopupClass 6402
GetScrollBtnClass 6402
GetStyleName 6402
TPlatformDefaultStyleActionBars Members 6400
TPlatformDefaultStyleActionBars methods 6400
TPopupList class 4932
about TPopupList class 4932
Add 4934
FWindow 4933
MainWndProc 4934
Remove 4934
TPopupList fields 4933
TPopupList Members 4933
TPopupList methods 4933
TPopupList properties 4935
Window 4935
WndProc 4935
TPopupMenu class 4935
about TPopupMenu class 4935
Alignment 4941
AutoHotkeys 4941
AutoLineReduction 4942
AutoPopup 4942
BiDiMode 4943
Create 4938
Destroy 4938
DoPopup 4939
HelpContext 4944
Images 4944
MenuAnimation 4945
OnChange 4949
OnPopup 4949
OwnerDraw 4945
ParentBiDiMode 4946
Popup 4939
PopupComponent 4946
PopupPoint 4948
SetPopupPoint 4940
TPopupMenu events 4949
TPopupMenu Members 4936
TPopupMenu methods 4938
TPopupMenu properties 4940
TrackButton 4948
UseRightToLeftAlignment 4940
TPostButton class 5396
about TPostButton class 5396
Create 5398
TPostButton Members 5397

TPreviousTab class 101
about TPreviousTab class 101
AfterTabChange 109
BeforeTabChange 109
Caption 104
Enabled 104
ExecuteTarget 102
HelpContext 105
HelpKeyword 105
HelpType 106
Hint 106
ImageIndex 106
OnHint 110
OnUpdate 110
OnValidateTab 111
ShortCut 107
SkipHiddenTabs 107
TabControl 107
TPreviousTab events 109
TPreviousTab Members 101
TPreviousTab methods 102
TPreviousTab properties 103
UpdateTarget 103
Visible 108
Wrap 108
TPrinter class 6406
Abort 6409
Aborted 6419
about TPrinter class 6406
BeginDoc 6411
Canvas 6421
Capabilities 6421
Copies 6422
Create 6408
Destroy 6408
EndDoc 6413
Fonts 6422
GetPrinter 6415
Handle 6422
NewPage 6415
Orientation 6424
PageHeight 6425
PageNumber 6427
PageWidth 6428
PrinterIndex 6430
Printers 6430
Printing 6430
Refresh 6417
SetPrinter 6418
Title 6433
TPrinter Members 6407
TPrinter methods 6408
TPrinter properties 6418
TPriorButton class 5399
about TPriorButton class 5399
Create 5400
TPriorButton Members 5399
TPriorPageButton class 5401
about TPriorPageButton class 5401
Create 5403
TPriorPageButton Members 5401
TProvider class 6563
about TProvider class 6563
TQueryButton class 5403
about TQueryButton class 5403
Caption 5409
Create 5404
Custom 5409
DefaultCaption 5405
EventContent 5406
GetCaption 5406
GetHTMLControlName 5406
GetHTMLForm 5407
GetInputType 5407
GetXMLComponent 5407
ImplContent 5407
InputType 5405
Notification 5408

ParentXMLComponent 5408
SetXMLComponent 5408
SetXMLUseParent 5409
Style 5410
StyleRule 5410
TQueryButton fields 5405
TQueryButton Members 5404
TQueryButton methods 5405
TQueryButton properties 5409
XMLComponent 5411
XMLUseParent 5411
TQueryButtons class 5412
about TQueryButtons class 5412
Custom 5413
Style 5414
StyleRule 5414
TQueryButtons Members 5412
TQueryButtons properties 5413
TQueryFieldGroup class 5415
about TQueryFieldGroup class 5415
Custom 5419
FindParam 5418
GetParamsList 5418
ImplCanAddClass 5419
IsParamInUse 5419
Style 5420
StyleRule 5420
TQueryFieldGroup Members 5416
TQueryFieldGroup methods 5418
TQueryFieldGroup properties 5419
TQueryForm class 5421
about TQueryForm class 5421
Action 5423
Custom 5423
Method 5424
Style 5424
StyleRule 5424
TQueryForm Members 5422
TQueryForm properties 5423
TQueryRadioGroup class 5425
about TQueryRadioGroup class 5425
AddAttributes 5428
Caption 5430
CaptionAttributes 5430
CaptionPosition 5431
Custom 5431
DataSet 5431
DisplayColumns 5432
DisplayWidth 5432
FieldName 5433
GetCheckIndex 5428
GetText 5429
IsQueryField 5429
Items 5433
ItemsField 5434
ParamName 5434
ReadOnly 5435
SetText 5429
Style 5435
StyleRule 5435
TabIndex 5436
Text 5436
TQueryRadioGroup Members 5426
TQueryRadioGroup methods 5428
TQueryRadioGroup properties 5429
Values 5436
ValuesField 5437
TQuerySelectOptions class 5437
about TQuerySelectOptions class 5437
Caption 5442
CaptionAttributes 5442
CaptionPosition 5443
Custom 5443
DataSet 5444
DisplayRows 5444
FieldName 5445
GetSelectIndex 5441
GetText 5441

IsQueryField 5441
Items 5445
ItemsField 5446
ParamName 5446
SetText 5441
Style 5447
StyleRule 5447
TabIndex 5447
Text 5448
TQuerySelectOptions Members 5438
TQuerySelectOptions methods 5440
TQuerySelectOptions properties 5442
Values 5448
ValuesField 5448
TQueryText class 5449
about TQueryText class 5449
AddAttributes 5452
Caption 5453
CaptionAttributes 5454
CaptionPosition 5454
Create 5451
Custom 5455
DisplayWidth 5455
FieldName 5455
GetText 5452
IsQueryField 5452
MaxWidth 5456
ParamName 5456
ReadOnly 5457
SetText 5453
Style 5457
StyleRule 5458
TabIndex 5458
Text 5458
TQueryText Members 5450
TQueryText methods 5452
TQueryText properties 5453
TQueryTextArea class 5458
about TQueryTextArea class 5458
AddAttributes 5462
Caption 5464
CaptionAttributes 5464
CaptionPosition 5465
ControlContent 5462
Create 5461
Custom 5465
Destroy 5461
DisplayRows 5466
DisplayWidth 5466
FieldName 5466
GetText 5462
IsQueryField 5463
Lines 5467
ParamName 5467
ReadOnly 5468
SetLines 5463
SetText 5463
Style 5468
StyleRule 5468
TabIndex 5469
TQueryTextArea Members 5459
TQueryTextArea methods 5462
TQueryTextArea properties 5464
Wrap 5469
TQuickAccessToolbar class 6835
about TQuickAccessToolbar class 6835
ActionBar 6839
Assign 6837
BarWidth 6839
Create 6836
DoChange 6838
OnChange 6840
Position 6839
Show 6838
ShowMinimize 6839
ShowMoreCommands 6840
ShowPosition 6840
TogglePosition 6838

TQuickAccessToolbar enumerations 6836
TQuickAccessToolbar events 6840
TQuickAccessToolbar Members 6835
TQuickAccessToolbar methods 6837
TQuickAccessToolbar properties 6838
TQuickAccessToolbarGroupItem class 6841
about TQuickAccessToolbarGroupItem class 6841
Action 6842
Caption 6842
Create 6841
Destroy 6842
Group 6842
TQuickAccessToolbarGroupItem Members 6841
TQuickAccessToolbarGroupItem properties 6842
TQuickAccessToolbarGroups class 6843
about TQuickAccessToolbarGroups class 6843
Add 6844
Clean 6844
Create 6843
Destroy 6844
Exists 6844
Group 6845
Remove 6845
TQuickAccessToolbarGroups Members 6843
TQuickAccessToolbarGroups methods 6844
TRadioGroup class 1406
about TRadioGroup class 1406
Align 1409
Anchors 1412
BiDiMode 1413
Caption 1413
Color 1416
Columns 1416
Constraints 1417
Ctl3D 1417
DoubleBuffered 1418
DragCursor 1420
DragKind 1420
DragMode 1420
Enabled 1422
Font 1422
ItemIndex 1423
Items 1424
OnClick 1438
OnContextPopup 1440
OnDragDrop 1440
OnDragOver 1443
OnEndDock 1447
OnEndDrag 1447
OnEnter 1448
OnExit 1451
OnStartDock 1453
OnStartDrag 1454
ParentBackground 1425
ParentBiDiMode 1426
ParentColor 1426
ParentCtl3D 1427
ParentDoubleBuffered 1427
ParentFont 1427
ParentShowHint 1428
PopupMenu 1428
ShowHint 1433
TabOrder 1435
TabStop 1435
TRadioGroup events 1438
TRadioGroup Members 1406
TRadioGroup properties 1408
Visible 1436
WordWrap 1437
TRegIniFile class 6601
about TRegIniFile class 6601
Create 6603
Create constructor 6603
DeleteKey 6604
EraseSection 6605
FileName 6610
ReadBool 6605
ReadInteger 6606

ReadSection 6606
ReadSections 6607
ReadSectionValues 6607
ReadString 6608
TRegIniFile Members 6601
TRegIniFile methods 6604
TRegIniFile properties 6610
WriteBool 6608
WriteInteger 6609
WriteString 6609
TRegistry class 6610
about TRegistry class 6610
Access 6657
ChangeKey 6619
CloseKey 6619
Create 6615
Create constructor 6615
CreateKey 6622
CurrentKey 6660
CurrentPath 6660
DeleteKey 6623
DeleteValue 6625
Destroy 6617
GetBaseKey 6626
GetData 6626
GetDataAsString 6626
GetDataInfo 6627
GetDataSize 6627
GetDataType 6628
GetKey 6628
GetKeyInfo 6628
GetKeyNames 6629
GetValueNames 6629
HasSubKeys 6630
KeyExists 6630
LazyWrite 6660
LoadKey 6632
MoveKey 6633
OpenKey 6634
OpenKeyReadOnly 6636
PutData 6637
ReadBinaryData 6637
ReadBool 6638
ReadCurrency 6638
ReadDate 6638
ReadDateTime 6639
ReadFloat 6639
ReadInteger 6640
ReadString 6640
ReadTime 6641
RegistryConnect 6641
RenameValue 6642
ReplaceKey 6642
RestoreKey 6643
RootKey 6661
SaveKey 6643
SetCurrentKey 6644
TRegistry Members 6613
TRegistry methods 6618
TRegistry properties 6656
UnLoadKey 6644
ValueExists 6645
WriteBinaryData 6645
WriteBool 6646
WriteCurrency 6646
WriteDate 6646
WriteDateTime 6647
WriteExpandString 6650
WriteFloat 6652
WriteInteger 6653
WriteString 6653
WriteTime 6656
TRegistryClass class 5953
about TRegistryClass class 5953
Create 5953
Destroy 5953
TRegistryClass Members 5953
TRegistryClass methods 5954

UpdateRegistry 5954
TRegistryIniFile class 6663
about TRegistryIniFile class 6663
Create 6668
Create constructor 6668
DeleteKey 6669
Destroy 6668
EraseSection 6670
ReadBinaryStream 6670
ReadDate 6671
ReadDateTime 6671
ReadFloat 6672
ReadInteger 6672
ReadSection 6676
ReadSections 6676
ReadSections method 6676
ReadSectionValues 6677
ReadString 6677
ReadTime 6680
RegIniFile 6690
TRegistryIniFile Members 6667
TRegistryIniFile methods 6669
TRegistryIniFile properties 6690
UpdateFile 6681
WriteBinaryStream 6681
WriteDate 6682
WriteDateTime 6682
WriteFloat 6683
WriteInteger 6683
WriteString 6686
WriteTime 6689
TRemotable class 4408
about TRemotable class 4408
Create 4410
DataContext 4413
Destroy 4410
FSerializationOptions 4411
ObjectToSOAP 4411
SerializationOptions 4413
SOAPToObject 4412
TRemotable fields 4411
TRemotable Members 4409
TRemotable methods 4411
TRemotable properties 4412
TRemotableTypeRegistry class 4414
about TRemotableTypeRegistry class 4414
AutoRegisterNativeTypes 4431
ClassOptions 4417
ClassToURI 4418
ClassToURI method 4418
Create 4415
Destroy 4416
GetExternalPropName 4418
GetInternalPropName 4419
GetURICount 4419
GetURIMap 4420
GetURIMap method 4420
GetVarTypeFromXSD 4421
GetXSDInfoForClass 4421
InfoToURI 4421
IsClassScalar 4422
Lock 4422
RegisterExternalPropName 4423
RegisterHolderClsMember 4423
RegisterSerializeOptions 4424
RegisterSerializeOptions method 4424
RegisterXSClass 4425
RegisterXSInfo 4426
SerializeOptions 4427
SerializeOptions method 4426
TRemotableTypeRegistry Members 4414
TRemotableTypeRegistry methods 4416
TRemotableTypeRegistry properties 4431
TypeInfoToXSD 4427
UnLock 4428
UnRegisterXSClass 4428
UnRegisterXSInfo 4429
URIToClass 4429

URIToClass method 4429
URIToInfo 4430
VariantToInfo 4430
XSDToTypeInfo 4431
TRemotableXS class 4432
about TRemotableXS class 4432
NativeToXS 4433
ObjectToSOAP 4434
SOAPToObject 4434
TRemotableXS Members 4433
TRemotableXS methods 4433
XSToNative 4435
TRemoteServer class 5239
about TRemoteServer class 5239
TResetQueryButton class 5469
about TResetQueryButton class 5469
Create 5471
EventContent 5471
TResetQueryButton Members 5470
TResetQueryButton methods 5471
TRibbon class 6845
about TRibbon class 6845
ActionManager 6849
Align 6850
Anchors 6853
ApplicationMenu 6854
BiDiMode 6854
Caption 6854
DocumentName 6857
Enabled 6857
Font 6858
Height 6859
HideTabs 6860
OnHelpButtonClick 6863
OnRecentItemClick 6863
OnTabChange 6864
ParentBiDiMode 6860
ParentFont 6861
QuickAccessToolbar 6861
ScreenTips 6861
ShowHelpButton 6862
Style 6862
TabIndex 6862
Tabs 6862
TRibbon events 6863
TRibbon Members 6846
TRibbon properties 6849
UseCustomFrame 6863
TRibbonActionPopupMenu class 7300
about TRibbonActionPopupMenu class 7300
CreateControl 7301
NCPaint 7301
SelectFirstItem 7302
TrackMenu 7302
TRibbonActionPopupMenu Members 7301
TRibbonActionPopupMenu methods 7301
TRibbonActionPopupMenu properties 7302
TRibbonAddRemoveItem class 6965
about TRibbonAddRemoveItem class 6965
DrawBackground 6966
DrawGlyph 6966
TRibbonAddRemoveItem Members 6966
TRibbonAddRemoveItem methods 6966
TRibbonApplicationButton class 6967
about TRibbonApplicationButton class 6967
CalcBounds 6967
TRibbonApplicationButton Members 6967
TRibbonApplicationButton methods 6967
TRibbonApplicationMenuBar class 7302
about TRibbonApplicationMenuBar class 7302
ActionManager 7310
Align 7310
CloseMenu 7305
Create 7304
CreateControl 7305
CreateControls 7306
Destroy 7304
DragOver 7306

DrawBackground 7306
GetControlClass 7306
GetRibbon 7307
Height 7314
HorzMargin 7315
Left 7315
NewPopup 7307
Notification 7307
OptionItems 7316
Paint 7308
RecentItems 7316
RibbonStyle 7308
SelectApplicationButton 7308
SetBounds 7308
SetRibbon 7309
ShowKeyTips 7309
Spacing 7316
Top 7317
TRibbonApplicationMenuBar Members 7303
TRibbonApplicationMenuBar methods 7305
TRibbonApplicationMenuBar properties 7309
VertMargin 7318
VisibleMenu 7319
VisibleMenuCount 7319
Width 7319
WMSysCommand 7309
TRibbonApplicationMenuButton class 6968
about TRibbonApplicationMenuButton class 6968
CalcBounds 6970
CMMouseEnter 6971
CMMouseLeave 6971
Create 6969
Destroy 6969
DragOver 6971
GetKeyTip 6971
HandleKeyTip 6972
HideKeyTip 6972
KeyTip 6976
KeyTipClick 6973
MouseInControl 6976
Paint 6973
RibbonStyle 6973
SetKeyTip 6974
SetParent 6974
ShowKeyTip 6975
TRibbonApplicationMenuButton Members 6968
TRibbonApplicationMenuButton methods 6970
TRibbonApplicationMenuButton properties 6975
UMFadeControl 6975
WMLButtonDblClk 6975
TRibbonApplicationMenuItem class 6976
about TRibbonApplicationMenuItem class 6976
CalcBounds 6978
Create 6977
DrawGlyph 6978
DrawSelectedBackground 6979
DrawSeparator 6979
DrawShadowedText 6979
DrawSubMenuGlyph 6979
DrawText 6980
GetImageSize 6980
TRibbonApplicationMenuItem Members 6977
TRibbonApplicationMenuItem methods 6978
TRibbonApplicationPopupMenu class 7321
about TRibbonApplicationPopupMenu class 7321
AlignControls 7323
CMTextchanged 7323
Create 7322
CreateControls 7323
CreateOptionsItems 7324
CreateRecentDocuments 7324
CreateWnd 7324
CustomAlignPosition 7325
Destroy 7322
DoDropActions 7325
DoDropCategory 7326
DoDropItem 7326
DrawBackground 7326

GetBorderHeight 7326
GetRibbon 7326
NCPaint 7327
PositionPopup 7327
RibbonStyle 7327
SetColorMap 7327
SetRibbon 7328
TRibbonApplicationPopupMenu Members 7321
TRibbonApplicationPopupMenu methods 7322
TRibbonBaseButtonControl class 6980
about TRibbonBaseButtonControl class 6980
AllowHotKeys 6983
CalcBounds 6983
CanAddToQuickAccessToolbar 6984
Click 6984
Create 6981
Destroy 6982
DragDrop 6984
DrawBackground 6985
DrawDown 6990
DrawGlyph 6985
DrawSeparator 6985
DrawShadowedText 6986
DrawText 6986
GetImageSize 6986
GetKeyTip 6986
HandleKeyTip 6987
HideKeyTip 6987
IsButton 6988
IsRibbonAvailable 6988
KeyTipClick 6988
RibbonStyle 6988
SetKeyTip 6989
ShowKeyTip 6989
TRibbonBaseButtonControl Members 6981
TRibbonBaseButtonControl methods 6983
TRibbonBaseButtonControl properties 6989
TRibbonCheckBoxControl class 6990
about TRibbonCheckBoxControl class 6990
CalcBounds 6991
DrawBackground 6992
DrawGlyph 6992
TRibbonCheckBoxControl Members 6990
TRibbonCheckBoxControl methods 6991
TRibbonComboBox class 6992
about TRibbonComboBox class 6992
Alignment 6995
AutoSelect 6995
CharCase 6996
Font 6996
HideSelection 6997
ImeMode 6998
ImeName 6998
Items 6999
MaxLength 6999
NumbersOnly 7000
OnChange 7012
OnClick 7013
OnContextPopup 7015
OnDblClick 7015
OnDragDrop 7016
OnDragOver 7019
OnEndDock 7022
OnEndDrag 7022
OnEnter 7023
OnExit 7026
OnKeyDown 7029
OnKeyPress 7032
OnKeyUp 7033
OnMouseActivate 7035
OnMouseDown 7036
OnMouseEnter 7038
OnMouseLeave 7038
OnMouseMove 7039
OnMouseUp 7041
OnStartDock 7043
OnStartDrag 7044
ParentFont 7000

PasswordChar 7000
PopupMenu 7001
ReadOnly 7006
ShowHint 7006
TabOrder 7008
TabStop 7008
Text 7009
TextHint 7010
TRibbonComboBox events 7012
TRibbonComboBox Members 6993
TRibbonComboBox properties 6994
Visible 7010
TRibbonComboBoxItem class 7045
about TRibbonComboBoxItem class 7045
CalcBounds 7046
DrawBackground 7046
DrawText 7046
TRibbonComboBoxItem Members 7045
TRibbonComboBoxItem methods 7045
TRibbonComboControl class 7047
about TRibbonComboControl class 7047
CalcBounds 7050
CanAddToQuickAccessToolbar 7050
Click 7051
CMBidimodechanged 7051
CMMouseenter 7051
CMMouseleave 7051
CMVisiblechanged 7052
ComboBox 7060
ComboClick 7052
ContainedControl 7060
Create 7048
Destroy 7049
DrawBackground 7052
DrawFrame 7052
DropDownClick 7053
GetCaptionWidth 7053
GetContainedControl 7053
GetGlyphWidth 7053
GetKeyTip 7053
GetLabelWidth 7054
HandleKeyTip 7054
HideKeyTip 7055
KeyTipClick 7055
MouseDown 7055
MouseInControl 7060
Notification 7056
Paint 7056
RibbonStyle 7057
SetActionClient 7057
SetBounds 7057
SetContainedControl 7057
SetDragMode 7058
SetKeyTip 7058
SetParent 7058
ShowComboItems 7059
ShowKeyTip 7059
TRibbonComboControl Members 7047
TRibbonComboControl methods 7049
TRibbonComboControl properties 7060
TRibbonCustomControl class 7061
about TRibbonCustomControl class 7061
CalcBounds 7063
CanAddToQuickAccessToolbar 7064
Click 7064
CMVisiblechanged 7064
ContainedControl 7073
Create 7062
Destroy 7062
DrawBackground 7065
GetCaptionWidth 7065
GetContainedControl 7065
GetGlyphWidth 7065
GetKeyTip 7066
GetLabelWidth 7066
HandleKeyTip 7066
HideKeyTip 7067
KeyTipClick 7067

Notification 7067
Paint 7068
SetActionClient 7068
SetBounds 7068
SetContainedControl 7071
SetDragMode 7071
SetKeyTip 7071
SetParent 7072
ShowKeyTip 7072
TRibbonCustomControl Members 7061
TRibbonCustomControl methods 7063
TRibbonCustomControl properties 7073
TRibbonCustomizeDlg class 6864
about TRibbonCustomizeDlg class 6864
SetupDlg 6865
TRibbonCustomizeDlg Members 6864
TRibbonCustomizeDlg methods 6865
TRibbonCustomizeItem class 7073
about TRibbonCustomizeItem class 7073
CalcBounds 7074
DrawBackground 7075
DrawGlyph 7075
RibbonStyle 7075
TRibbonCustomizeItem Members 7074
TRibbonCustomizeItem methods 7074
TRibbonCustomStatusControl class 7076
about TRibbonCustomStatusControl class 7076
CalcBounds 7077
DrawBackground 7078
TRibbonCustomStatusControl Members 7076
TRibbonCustomStatusControl methods 7077
TRibbonDropDownButton class 7078
about TRibbonDropDownButton class 7078
CalcBounds 7080
CalcLayout 7080
Click 7080
DoDropDown 7081
DrawArrow 7081
DrawBackground 7081
DrawShadowedText 7081
DrawText 7082
GetArrowPosition 7082
GetPopupClass 7082
KeyTipClick 7082
TRibbonDropDownButton Members 7078
TRibbonDropDownButton methods 7079
TRibbonGallery class 7083
about TRibbonGallery class 7083
TRibbonGalleryBar class 7360
about TRibbonGalleryBar class 7360
Items 7361
ItemsPerRow 7362
ShowItemCaptions 7362
TRibbonGalleryBar Members 7361
TRibbonGalleryBar properties 7361
TRibbonGalleryControl class 7083
about TRibbonGalleryControl class 7083
CanAddToQuickAccessToolbar 7084
ContainedControl 7085
Gallery 7085
GetContainedControl 7084
SetContainedControl 7084
TRibbonGalleryControl Members 7083
TRibbonGalleryControl methods 7084
TRibbonGalleryControl properties 7085
TRibbonGalleryControlMenuItem class 7085
about TRibbonGalleryControlMenuItem class 7085
CalcBounds 7086
CanAddToQuickAccessToolbar 7087
ContainedControl 7091
GetContainedControl 7087
Paint 7087
SetBounds 7088
SetContainedControl 7090
SetParent 7090
TRibbonGalleryControlMenuItem Members 7086
TRibbonGalleryControlMenuItem methods 7086
TRibbonGalleryControlMenuItem properties 7091

TRibbonGalleryItem class 7091
about TRibbonGalleryItem class 7091
CalcBounds 7093
CalcLayout 7093
TRibbonGalleryItem Members 7092
TRibbonGalleryItem methods 7093
TRibbonGalleryMenuItem class 7093
about TRibbonGalleryMenuItem class 7093
CalcBounds 7094
DrawBackground 7095
DrawGlyph 7095
DrawShadowedText 7095
DrawText 7096
TRibbonGalleryMenuItem Members 7094
TRibbonGalleryMenuItem methods 7094
TRibbonGalleryPopupBar class 7362
about TRibbonGalleryPopupBar class 7362
TRibbonGallerySplitButton class 7096
about TRibbonGallerySplitButton class 7096
GetPopupClass 7098
TRibbonGallerySplitButton Members 7096
TRibbonGallerySplitButton methods 7098
TRibbonGripItem class 7098
about TRibbonGripItem class 7098
CalcBounds 7100
Create 7099
GalleryResize 7101
Paint 7100
RibbonStyle 7101
TRibbonGripItem Members 7099
TRibbonGripItem methods 7100
TRibbonGripItem properties 7101
TRibbonGroup class 6865
about TRibbonGroup class 6865
ActionManager 6868
BiDiMode 6868
Caption 6869
DialogAction 6869
GroupAlign 6869
GroupIndex 6869
Height 6870
KeyTip 6870
ParentBiDiMode 6870
Rows 6870
TRibbonGroup Members 6866
TRibbonGroup properties 6868
TRibbonGroupCollapsedControl class 6871
about TRibbonGroupCollapsedControl class 6871
ActionManager 6880
AdjustBounds 6874
Caption 6880
Click 6874
CloseUp 6875
CMFontchanged 6875
CMMouseenter 6875
CMMouseleave 6875
CMTextchanged 6875
Create 6872
Destroy 6873
GetKeyTip 6876
GetParentControl 6876
GetTopPosition 6876
Group 6883
HandleKeyTip 6876
HideKeyTip 6877
KeyTip 6883
KeyTipClick 6877
MouseInControl 6883
Notification 6878
OldWidth 6883
Paint 6878
SetKeyTip 6878
SetupGroupImage 6879
ShowKeyTip 6879
TRibbonGroupCollapsedControl Members 6871
TRibbonGroupCollapsedControl methods 6873
TRibbonGroupCollapsedControl properties 6880
TRibbonGroupDockForm class 6884

about TRibbonGroupDockForm class 6884
Control 6886
Create 6885
CreateParams 6885
ParentControl 6886
TRibbonGroupDockForm Members 6884
TRibbonGroupDockForm methods 6885
TRibbonGroupDockForm properties 6886
WMActivate 6885
TRibbonGroupDropDown class 7101
about TRibbonGroupDropDown class 7101
CloseUp 7103
DoDropDown 7103
DrawArrow 7104
DrawBackground 7104
GetParentControl 7104
GetTopPosition 7104
Group 7105
TRibbonGroupDropDown Members 7102
TRibbonGroupDropDown methods 7103
TRibbonGroupDropDown properties 7105
TRibbonGroupScrollButton class 6886
about TRibbonGroupScrollButton class 6886
CMDesignHitTest 6888
CMMouseenter 6888
CMMouseleave 6888
Create 6887
Direction 6889
MouseInControl 6889
Paint 6888
RibbonStyle 6890
TRibbonGroupScrollButton Members 6887
TRibbonGroupScrollButton methods 6888
TRibbonGroupScrollButton properties 6889
TRibbonLargeButtonControl class 7105
about TRibbonLargeButtonControl class 7105
CalcBounds 7107
CalcLayout 7107
DrawBackground 7107
DrawShadowedText 7108
DrawText 7108
TRibbonLargeButtonControl Members 7105
TRibbonLargeButtonControl methods 7106
TRibbonLargeSplitButton class 7108
about TRibbonLargeSplitButton class 7108
CalcBounds 7110
CalcLayout 7111
Click 7111
CMMouseleave 7111
CursorInSplit 7115
DoSplitButtonClick 7111
DrawArrow 7112
DrawBackground 7112
GetArrowPosition 7112
InSplit 7112
KeyTipClick 7113
MouseDown 7113
MouseMove 7113
MouseUp 7114
TRibbonLargeSplitButton Members 7109
TRibbonLargeSplitButton methods 7110
TRibbonLargeSplitButton properties 7115
UpdateSplitBounds 7114
TRibbonLunaColorMap class 6890
about TRibbonLunaColorMap class 6890
ActiveTabFontColor 6892
BevelMinimizedInner 6892
BevelMinimizedOuter 6892
BtnSelectedFont 6892
CaptionFontColor 6893
ControlColor 6893
ControlFrameColor 6893
ControlInactiveColor 6893
DisabledFontColor 6894
DocumentFontColor 6894
FontColor 6894
GroupFontColor 6894
HighlightColor 6895

InactiveTabFontColor 6895
QuickAccessToolbarMoreBorderColor 6895
QuickAccessToolbarMoreColor 6895
TRibbonLunaColorMap Members 6890
TRibbonLunaColorMap methods 6891
TRibbonLunaColorMap properties 6891
UpdateColors 6891
TRibbonMainMenuItem class 7115
about TRibbonMainMenuItem class 7115
DrawShadowedText 7116
DrawText 7116
TRibbonMainMenuItem Members 7115
TRibbonMainMenuItem methods 7116
TRibbonMainMenuSeparator class 7117
about TRibbonMainMenuSeparator class 7117
CalcBounds 7118
TRibbonMainMenuSeparator Members 7117
TRibbonMainMenuSeparator methods 7118
TRibbonMenuItem class 7118
about TRibbonMenuItem class 7118
CalcBounds 7120
CalcLayout 7120
Click 7120
Create 7119
DrawBackground 7121
DrawBanner 7123
DrawGlyph 7121
DrawSelectedBackground 7121
DrawSeparator 7122
DrawShadowedText 7122
DrawText 7122
GetShowShortCut 7122
RibbonStyle 7123
TRibbonMenuItem Members 7118
TRibbonMenuItem methods 7119
TRibbonMenuItem properties 7123
TRibbonMenuSeparatorItem class 7123
about TRibbonMenuSeparatorItem class 7123
CalcBounds 7124
DrawSeparator 7125
TRibbonMenuSeparatorItem Members 7124
TRibbonMenuSeparatorItem methods 7124
TRibbonMenuSmallButtonControl class 7125
about TRibbonMenuSmallButtonControl class 7125
Create 7126
MouseUp 7127
TRibbonMenuSmallButtonControl Members 7125
TRibbonMenuSmallButtonControl methods 7127
TRibbonObsidianColorMap class 6896
about TRibbonObsidianColorMap class 6896
ActiveTabFontColor 6898
BevelMinimizedInner 6898
BevelMinimizedOuter 6898
BtnSelectedFont 6898
CaptionFontColor 6898
ControlColor 6899
ControlFrameColor 6899
ControlInactiveColor 6899
DisabledFontColor 6899
DocumentFontColor 6900
FontColor 6900
GroupFontColor 6900
HighlightColor 6900
InactiveTabFontColor 6901
QuickAccessToolbarMoreBorderColor 6901
QuickAccessToolbarMoreColor 6901
TRibbonObsidianColorMap Members 6896
TRibbonObsidianColorMap methods 6897
TRibbonObsidianColorMap properties 6897
UpdateColors 6897
TRibbonOptionButtonControl class 7128
about TRibbonOptionButtonControl class 7128
DrawBackground 7129
TRibbonOptionButtonControl Members 7128
TRibbonOptionButtonControl methods 7129
TRibbonPage class 6901
about TRibbonPage class 6901
BiDiMode 6903

Caption 6904
Height 6906
Index 6908
KeyTip 6908
Name 6908
ParentBiDiMode 6910
TRibbonPage Members 6902
TRibbonPage properties 6903
TRibbonPopupGalleryBar class 7362
about TRibbonPopupGalleryBar class 7362
Assign 7364
CreateParams 7364
DisplayGallery 7365
TRibbonPopupGalleryBar Members 7363
TRibbonPopupGalleryBar methods 7363
TRibbonQATCustomizeMenu class 7328
about TRibbonQATCustomizeMenu class 7328
DisplayShadow 7329
DoAddCustomizeItems 7329
GetControlClass 7329
HideShadow 7329
NCPaint 7330
PositionPopup 7330
TRibbonQATCustomizeMenu Members 7328
TRibbonQATCustomizeMenu methods 7328
VisibleChanging 7330
TRibbonQATScrollButton class 7130
about TRibbonQATScrollButton class 7130
CalcBounds 7131
Click 7131
CMMouseEnter 7132
CMMouseLeave 7132
CursorInBtn 7134
DrawArrows 7132
DrawBackground 7132
DrawDown 7134
MouseMove 7133
RibbonStyle 7133
ScrollMode 7134
TRibbonQATScrollButton Members 7130
TRibbonQATScrollButton methods 7130
TRibbonQATScrollButton properties 7133
TRibbonQuickAccessPopupToolbar class 6911
about TRibbonQuickAccessPopupToolbar class 6911
TRibbonQuickAccessToolbar class 6911
about TRibbonQuickAccessToolbar class 6911
ActionManager 6913
OnMoreCommands 6913
TRibbonQuickAccessToolbar Members 6912
TRibbonQuickAccessToolbar properties 6913
VertMargin 6914
TRibbonRadioButtonControl class 7134
about TRibbonRadioButtonControl class 7134
CalcBounds 7136
DrawBackground 7136
DrawGlyph 7136
TRibbonRadioButtonControl Members 7135
TRibbonRadioButtonControl methods 7136
TRibbonRecentButtonControl class 7137
about TRibbonRecentButtonControl class 7137
AllowHotKeys 7138
TRibbonRecentButtonControl Members 7137
TRibbonRecentButtonControl methods 7138
TRibbonRecentDocuments class 7330
about TRibbonRecentDocuments class 7330
Caption 7332
Create 7331
Paint 7332
TRibbonRecentDocuments Members 7331
TRibbonRecentDocuments methods 7332
TRibbonRecentDocuments properties 7332
TRibbonRichContentItem class 7139
about TRibbonRichContentItem class 7139
CalcBounds 7140
CanCalculateHeight 7141
Create 7140
DrawBackground 7141
DrawGlyph 7141

DrawShadowedText 7142
DrawText 7142
GetItemHeight 7142
GetItemWidth 7143
GetMenuProps 7143
TRibbonRichContentItem Members 7139
TRibbonRichContentItem methods 7140
TRibbonRichContentMainMenuItem class 7143
about TRibbonRichContentMainMenuItem class 7143
CalcBounds 7145
CalcLayout 7145
Create 7144
DrawGlyph 7145
DrawText 7146
GetImageSize 7146
GetShowShortCut 7146
TRibbonRichContentMainMenuItem Members 7143
TRibbonRichContentMainMenuItem methods 7145
TRibbonSeparator class 7147
about TRibbonSeparator class 7147
CalcBounds 7148
Create 7147
DrawBackground 7148
DrawGlyph 7149
DrawShadowedText 7149
DrawText 7149
RibbonStyle 7150
TRibbonSeparator Members 7147
TRibbonSeparator methods 7148
TRibbonSilverColorMap class 6914
about TRibbonSilverColorMap class 6914
ActiveTabFontColor 6916
BevelMinimizedInner 6916
BevelMinimizedOuter 6916
BtnSelectedFont 6917
CaptionFontColor 6917
ControlColor 6917
ControlFrameColor 6917
ControlInactiveColor 6917
DisabledFontColor 6918
DocumentFontColor 6918
FontColor 6918
GroupFontColor 6918
HighlightColor 6919
InactiveTabFontColor 6919
QuickAccessToolbarMoreBorderColor 6919
QuickAccessToolbarMoreColor 6919
TRibbonSilverColorMap Members 6914
TRibbonSilverColorMap methods 6915
TRibbonSilverColorMap properties 6915
UpdateColors 6915
TRibbonSmallButtonControl class 7150
about TRibbonSmallButtonControl class 7150
CalcBounds 7151
TRibbonSmallButtonControl Members 7150
TRibbonSmallButtonControl methods 7151
TRibbonSmallSplitButton class 7152
about TRibbonSmallSplitButton class 7152
CalcBounds 7155
CalcLayout 7155
Click 7156
CMMouseLeave 7156
Create 7153
CursorInSplit 7159
DrawBackground 7156
FCursorInSplit 7154
FSplitRect 7154
GetArrowPosition 7157
InSplit 7157
KeyTipClick 7157
MouseDown 7157
MouseMove 7158
TRibbonSmallSplitButton fields 7154
TRibbonSmallSplitButton Members 7152
TRibbonSmallSplitButton methods 7155
TRibbonSmallSplitButton properties 7158
UpdateSplitBounds 7158
TRibbonSpinButton class 7159

about TRibbonSpinButton class 7159
Align 7164
Anchors 7167
Constraints 7168
Control 7161
Create 7160
Ctl3D 7169
DownGlyph 7169
DownNumGlyphs 7169
DragCursor 7170
DragKind 7170
DragMode 7170
Enabled 7172
FocusControl 7172
KeyDown 7161
Loaded 7162
Notification 7162
OnDownClick 7183
OnDragDrop 7183
OnDragOver 7186
OnEndDock 7190
OnEndDrag 7190
OnEnter 7191
OnExit 7194
OnStartDock 7196
OnStartDrag 7197
OnUpClick 7198
ParentCtl3D 7173
ParentShowHint 7173
PopupMenu 7173
SetBounds 7163
ShowHint 7178
TabOrder 7180
TabStop 7181
TRibbonSpinButton events 7183
TRibbonSpinButton Members 7159
TRibbonSpinButton methods 7161
TRibbonSpinButton properties 7163
UpGlyph 7181
UpNumGlyphs 7181
Visible 7181
TRibbonSpinEdit class 7198
about TRibbonSpinEdit class 7198
Anchors 7207
AutoSelect 7207
AutoSize 7208
Button 7209
CMMouseenter 7201
CMMouseleave 7201
Color 7209
Constraints 7210
Create 7200
CreateParams 7201
CreateWnd 7202
Ctl3D 7210
Destroy 7200
DownClick 7202
DragCursor 7211
DragMode 7211
EditorEnabled 7213
Enabled 7213
Font 7214
GetChildren 7203
Increment 7215
IsValidChar 7203
KeyDown 7203
KeyPress 7204
MaxLength 7215
MaxValue 7215
MinValue 7216
OnChange 7228
OnClick 7228
OnDblClick 7230
OnDragDrop 7230
OnDragOver 7233
OnEndDrag 7237
OnEnter 7238
OnExit 7240

OnKeyDown 7243
OnKeyPress 7246
OnKeyUp 7248
OnMouseDown 7249
OnMouseMove 7252
OnMouseUp 7254
OnStartDrag 7256
ParentColor 7216
ParentCtl3D 7216
ParentFont 7217
ParentShowHint 7217
PopupMenu 7218
ReadOnly 7223
SetEnabled 7204
SetParent 7205
ShowHint 7223
TabOrder 7224
TabStop 7225
TRibbonSpinEdit events 7227
TRibbonSpinEdit Members 7198
TRibbonSpinEdit methods 7201
TRibbonSpinEdit properties 7206
UpClick 7206
Value 7226
Visible 7226
WMKillFocus 7206
TRibbonSplitMenuItem class 7257
about TRibbonSplitMenuItem class 7257
CalcBounds 7259
Click 7260
Create 7258
DrawBackground 7260
DrawSelectedBackground 7260
FMouseInSplit 7259
FSplitRect 7259
IsInSplit 7261
MouseInSplit 7262
MouseMove 7261
TRibbonSplitMenuItem fields 7258
TRibbonSplitMenuItem Members 7257
TRibbonSplitMenuItem methods 7259
TRibbonSplitMenuItem properties 7262
UpdateSplitBounds 7261
TRibbonStyleActionBars class 7376
about TRibbonStyleActionBars class 7376
Create 7377
Destroy 7377
DrawElement 7379
DrawElement method 7378
GetAddRemoveItemClass 7380
GetColorMapClass 7380
GetControlClass 7380
GetPopupClass 7380
GetScrollBtnClass 7381
GetSkinClass 7381
RibbonSkin 7381
TRibbonStyleActionBars Members 7376
TRibbonStyleActionBars methods 7377
TRibbonStyleActionBars properties 7381
TRibbonStyleMainPopupMenu class 7335
about TRibbonStyleMainPopupMenu class 7335
Create 7335
NCPaint 7336
PositionPopup 7336
TRibbonStyleMainPopupMenu Members 7335
TRibbonStyleMainPopupMenu methods 7336
TRibbonStyleMenuButton class 7262
about TRibbonStyleMenuButton class 7262
DrawBackground 7263
RibbonStyle 7263
TRibbonStyleMenuButton Members 7262
TRibbonStyleMenuButton methods 7263
TRibbonStylePopupMenu class 7336
about TRibbonStylePopupMenu class 7336
AlignControls 7338
CanAutoSize 7339
Create 7337
CreateControl 7339

Destroy 7338
GalleryResize 7340
TRibbonStylePopupMenu Members 7337
TRibbonStylePopupMenu methods 7338
TRibbonStylePopupMenu properties 7340
WMExitsizemove 7339
WMNCHitTest 7340
TRibbonTabItem class 6920
about TRibbonTabItem class 6920
Assign 6921
BestTabWidth 6923
Caption 6923
Destroy 6920
GetDisplayName 6922
KeyTip 6923
KeyTipForm 6923
MinTabwidth 6923
MinTruncatedWidth 6924
Modified 6924
Page 6924
TRibbonTabItem Members 6920
TRibbonTabItem methods 6921
TRibbonTabItem properties 6922
X 6924
Y 6925
TRibbonTabs class 6925
about TRibbonTabs class 6925
Add 6926
Create 6926
DesignerAdd 6929
FindGroup 6927
GetPageFromPageCaption 6927
IndexOfCaption 6927
Items 6929
Notify 6927
Ribbon 6930
SelectNext 6928
SelectPrior 6928
TabExists 6929
TRibbonTabs Members 6925
TRibbonTabs methods 6926
TRibbonTabs properties 6929
TRibbonTextControl class 7263
about TRibbonTextControl class 7263
CalcBounds 7264
DrawBackground 7265
DrawText 7265
TRibbonTextControl Members 7264
TRibbonTextControl methods 7264
TRibbonToolScrollButton class 7265
about TRibbonToolScrollButton class 7265
DrawArrows 7266
DrawBackground 7266
TRibbonToolScrollButton Members 7266
TRibbonToolScrollButton methods 7266
TRichEditAction class 112
about TRichEditAction class 112
Create 113
CurrText 114
HandlesTarget 114
SetFontStyle 114
TRichEditAction Members 113
TRichEditAction methods 113
TRichEditAlignCenter class 115
about TRichEditAlignCenter class 115
ExecuteTarget 116
TRichEditAlignCenter Members 115
TRichEditAlignCenter methods 116
UpdateTarget 116
TRichEditAlignLeft class 117
about TRichEditAlignLeft class 117
ExecuteTarget 118
TRichEditAlignLeft Members 117
TRichEditAlignLeft methods 118
UpdateTarget 118
TRichEditAlignRight class 119
about TRichEditAlignRight class 119
ExecuteTarget 120

TRichEditAlignRight Members 120
TRichEditAlignRight methods 120
UpdateTarget 120
TRichEditBold class 121
about TRichEditBold class 121
ExecuteTarget 122
TRichEditBold Members 122
TRichEditBold methods 122
UpdateTarget 123
TRichEditBullets class 123
about TRichEditBullets class 123
ExecuteTarget 124
TRichEditBullets Members 124
TRichEditBullets methods 124
UpdateTarget 125
TRichEditItalic class 125
about TRichEditItalic class 125
ExecuteTarget 127
TRichEditItalic Members 126
TRichEditItalic methods 126
UpdateTarget 127
TRichEditStrikeOut class 127
about TRichEditStrikeOut class 127
ExecuteTarget 129
TRichEditStrikeOut Members 128
TRichEditStrikeOut methods 128
UpdateTarget 129
TRichEditUnderline class 130
about TRichEditUnderline class 130
ExecuteTarget 131
TRichEditUnderline Members 130
TRichEditUnderline methods 131
UpdateTarget 131
TRIO class 7408
about TRIO class 7408
AfterConstruction 7412
BeforeDestruction 7412
Converter 7415
Create 7410
Destroy 7410
DoAfterExecute 7413
DoBeforeExecute 7413
FConverter 7411
FIID 7411
FWebNode 7411
GetResponseStream 7414
IntfMD 7411
NewInstance 7414
OnAfterExecute 7417
OnBeforeExecute 7418
OnGetAttachment 7419
OnSendAttachment 7419
QueryInterface 7415
RefCount 7416
SOAPHeaders 7416
TRIO events 7417
TRIO fields 7410
TRIO Members 7409
TRIO methods 7412
TRIO properties 7415
WebNode 7417
TRowCollection class 1455
about TRowCollection class 1455
Add 1456
Create 1456
GetItemAttr 1456
Notify 1457
TRowCollection Members 1455
TRowCollection methods 1456
TRowItem class 1457
about TRowItem class 1457
TSavePicture class 132
about TSavePicture class 132
Caption 134
Dialog 134
Enabled 135
GetDialogClass 133
HelpContext 135

HelpKeyword 136
HelpType 136
Hint 137
ImageIndex 137
OnAccept 139
OnCancel 139
OnHint 139
OnUpdate 139
SecondaryShortCuts 137
ShortCut 138
TSavePicture events 138
TSavePicture Members 133
TSavePicture methods 133
TSavePicture properties 134
Visible 138
TSavePictureDialog class 1577
about TSavePictureDialog class 1577
Execute 1579
Execute method 1579
TSavePictureDialog Members 1578
TSavePictureDialog methods 1578
TSaveTextFileDialog class 1580
about TSaveTextFileDialog class 1580
Execute 1581
Execute method 1581
TSaveTextFileDialog Members 1580
TSaveTextFileDialog methods 1580
tsbGroupLeft enumeration member 6938
tsbGroupRight enumeration member 6938
tsbTabLeft enumeration member 6938
tsbTabRight enumeration member 6938
TScreen class 2445
about TScreen class 2445
ActiveControl 2452
ActiveCustomForm 2452
ActiveForm 2453
Create 2447
Cursor 2455
CursorCount 2458
Cursors 2458
CustomFormCount 2460
CustomForms 2460
DataModuleCount 2461
DataModules 2462
DefaultIme 2462
DefaultKbLayout 2463
DesktopHeight 2463
DesktopLeft 2463
DesktopRect 2464
DesktopTop 2465
DesktopWidth 2465
Destroy 2448
DisableAlign 2448
EnableAlign 2449
FocusedForm 2466
Fonts 2466
FormCount 2467
Forms 2468
Height 2470
HintFont 2471
IconFont 2472
Imes 2472
MenuFont 2473
MessageFont 2473
MonitorCount 2473
MonitorFromPoint 2449
MonitorFromRect 2449
MonitorFromWindow 2450
Monitors 2474
OnActiveControlChange 2479
OnActiveFormChange 2481
PixelsPerInch 2474
PrimaryMonitor 2474
Realign 2450
ResetFonts 2451
SaveFocusedList 2475
TScreen events 2479
TScreen Members 2446

TScreen methods 2448
TScreen properties 2451
Width 2475
WorkAreaHeight 2476
WorkAreaLeft 2477
WorkAreaRect 2477
WorkAreaTop 2478
WorkAreaWidth 2478
TScrollBox class 2483
about TScrollBox class 2483
Align 2488
Anchors 2492
AutoScroll 2492
AutoSize 2493
BevelEdges 2493
BevelInner 2493
BevelKind 2494
BevelOuter 2494
BevelWidth 2495
BiDiMode 2495
BorderStyle 2496
Color 2496
Constraints 2497
Create 2486
CreateParams 2487
Ctl3D 2497
DockSite 2498
DoubleBuffered 2499
DragCursor 2501
DragKind 2501
DragMode 2501
Enabled 2503
Font 2503
OnCanResize 2517
OnClick 2518
OnConstrainedResize 2520
OnContextPopup 2520
OnDblClick 2521
OnDockDrop 2521
OnDockOver 2523
OnDragDrop 2525
OnDragOver 2528
OnEndDock 2531
OnEndDrag 2532
OnEnter 2533
OnExit 2535
OnGetSiteInfo 2538
OnMouseActivate 2540
OnMouseDown 2541
OnMouseEnter 2543
OnMouseLeave 2544
OnMouseMove 2544
OnMouseUp 2546
OnMouseWheel 2548
OnMouseWheelDown 2549
OnMouseWheelUp 2549
OnResize 2549
OnStartDock 2550
OnStartDrag 2551
OnUnDock 2551
Padding 2504
PaintWindow 2487
ParentBackground 2505
ParentBiDiMode 2505
ParentColor 2505
ParentCtl3D 2506
ParentDoubleBuffered 2506
ParentFont 2506
ParentShowHint 2507
PopupMenu 2507
ShowHint 2512
TabOrder 2514
TabStop 2515
TScrollBox events 2516
TScrollBox Members 2484
TScrollBox methods 2486
TScrollBox properties 2487
Visible 2515

TScrollingWinControl class 2554
about TScrollingWinControl class 2554
AdjustClientRect 2557
AlignControls 2557
AutoScroll 2561
AutoScrollEnabled 2558
AutoScrollInView 2558
ChangeScale 2558
Create 2555
CreateParams 2559
CreateWnd 2559
Destroy 2556
DisableAutoRange 2559
DoFlipChildren 2560
EnableAutoRange 2560
HorzScrollBar 2562
OnAlignInsertBefore 2564
OnAlignPosition 2564
Resizing 2561
ScrollInView 2561
TScrollingWinControl events 2564
TScrollingWinControl Members 2555
TScrollingWinControl methods 2556
TScrollingWinControl properties 2561
VertScrollBar 2563
TSelectOptionsColumn class 5472
about TSelectOptionsColumn class 5472
Caption 5475
CaptionAttributes 5476
Custom 5476
DataSet 5477
DisplayRows 5477
FieldName 5478
IsColumn 5475
Items 5478
ItemsField 5478
Style 5479
StyleRule 5479
TabIndex 5480
TSelectOptionsColumn Members 5472
TSelectOptionsColumn methods 5474
TSelectOptionsColumn properties 5475
Values 5480
ValuesField 5480
TSendMail class 141
about TSendMail class 141
Caption 144
Destroy 142
Enabled 144
ExecuteTarget 142
HandlesTarget 143
HelpContext 144
HelpKeyword 145
HelpType 145
Hint 146
ImageIndex 146
OnHint 148
OnUpdate 148
SecondaryShortCuts 146
ShortCut 147
Text 147
TSendMail events 148
TSendMail Members 141
TSendMail methods 142
TSendMail properties 143
UpdateTarget 143
Visible 148
TServerCollection class 5908
about TServerCollection class 5908
Create 5909
FindServer 5910
GetBalancedName 5910
GetNextName 5911
Items 5911
TServerCollection Members 5909
TServerCollection methods 5910
TServerCollection properties 5911
TServerEventDispatch class 6144

_AddRef 6145
_Release 6145
about TServerEventDispatch class 6144
Create 6144
GetIDsOfNames 6145
GetTypeInfo 6146
GetTypeInfoCount 6146
Invoke 6146
QueryInterface 6146
Server 6147
ServerDisconnect 6147
TServerEventDispatch Members 6144
TServerEventDispatch methods 6145
TServerEventDispatch properties 6147
TServerItem class 5912
about TServerItem class 5912
ComputerName 5914
Create 5913
Enabled 5915
GetDisplayName 5914
HasFailed 5915
Port 5916
TServerItem Members 5912
TServerItem methods 5913
TServerItem properties 5914
TShape class 1458
about TShape class 1458
Align 1461
Anchors 1465
Brush 1465
Constraints 1466
Create 1459
Destroy 1460
DragCursor 1466
DragKind 1467
DragMode 1467
Enabled 1469
OnContextPopup 1474
OnDragDrop 1475
OnDragOver 1478
OnEndDock 1481
OnEndDrag 1481
OnMouseActivate 1482
OnMouseDown 1483
OnMouseEnter 1486
OnMouseLeave 1486
OnMouseMove 1486
OnMouseUp 1488
OnStartDock 1491
OnStartDrag 1491
Paint 1460
ParentShowHint 1469
Pen 1470
Shape 1470
ShowHint 1471
StyleChanged 1461
TShape events 1474
TShape Members 1458
TShape methods 1460
TShape properties 1461
Visible 1472
TSharedConnection class 4814
about TSharedConnection class 4814
AfterConnect 4820
AfterDisconnect 4821
BeforeConnect 4821
BeforeDisconnect 4822
ChildName 4819
ConnectEvent 4817
Create 4816
Destroy 4816
DoConnect 4817
DoDisconnect 4818
GetConnected 4818
GetServer 4818
Notification 4819
ParentConnection 4820
TSharedConnection events 4820

TSharedConnection Members 4815
TSharedConnection methods 4817
TSharedConnection properties 4819
TSharedImage class 2830
about TSharedImage class 2830
Destroy 2831
FreeHandle 2832
RefCount 2832
Reference 2832
Release 2832
TSharedImage Members 2831
TSharedImage methods 2831
TSharedImage properties 2832
TSimpleObjectBroker class 5916
about TSimpleObjectBroker class 5916
Create 5918
Destroy 5918
GetComputerForGUID 5919
GetComputerForProgID 5919
GetPortForComputer 5920
LoadBalanced 5922
LoadFromStream 5920
SaveToStream 5921
ServerData 5922
Servers 5923
SetConnectStatus 5921
TSimpleObjectBroker Members 5917
TSimpleObjectBroker methods 5918
TSimpleObjectBroker properties 5922
TSOAPAttachment class 4435
about TSOAPAttachment class 4435
CacheFile 4444
CacheFilePersist 4445
ContentType 4445
Create 4437
Destroy 4438
Encoding 4446
GetSourceStream 4439
Headers 4446
Init 4439
InternalSetCacheFile 4440
InternalSetSourceStream 4440
ObjectToSOAP 4440
Ownership 4447
SaveToFile 4441
SaveToStream 4441
SetOwnership 4442
SetSourceFile 4442
SetSourceStream 4443
SetSourceString 4443
SOAPToObject 4443
SourceStream 4447
SourceString 4448
TSOAPAttachment Members 4437
TSOAPAttachment methods 4438
TSOAPAttachment properties 4444
TSOAPDomConv class 6178
about TSOAPDomConv class 6178
AddAttachment 6183
AddMultiRefNode 6183
AddNamespaceDecl 6183
AddObjectAsWriting 6184
CheckEncodingStyle 6184
ChildNodesAreNull 6184
ConvertByteArrayToSoap 6184
ConvertEnumToSoap 6184
ConvertNativeArrayToSoap 6185
ConvertNativeDataToSoap 6185
ConvertObjectToSOAP 6186
ConvertSoapToEnum 6186
ConvertSoapToNativeArray 6186
ConvertSoapToNativeArrayElem 6186
ConvertSoapToNativeData 6187
ConvertSOAPToObject 6187
ConvertSoapToVariant 6187
ConvertVariantToSoap 6188
Create 6180
CreateMultiRefNode 6188

CreateNULLNode 6188
CreateObjectNode 6188
CreateScalarNodeXS 6189
CreateTypedNode 6189
Destroy 6181
FinalizeMultiRefNodes 6189
FindAttachment 6189
FindMultiRefNodeByID 6190
FindMultiRefNodeByInstance 6190
FindNodeByHREF 6190
FindPrefixForURI 6190
GetDataNode 6191
GetElementType 6191
GetNewID 6191
GetNodeAsText 6191
GetObjectPropAsText 6192
GetOptions 6192
GetTypeBySchemaNS 6192
InitObjectFromSOAP 6192
IsNodeAVarArray 6193
IsObjectWriting 6193
LoadObject 6193
MakeArrayNode 6194
MakeArrayNode method 6193
MultiRefObject 6194
NodeIsNULL 6194
ObjectMemberNoShow 6195
ObjInstanceToSOAP 6195
OnMemberDataNotReceived 6202
OnUnhandledNode 6202
Options 6201
ReadHeader 6195
ReadRectDynArray 6196
ReadRectDynArrayElem 6196
ReadRow 6196
ReadVarArrayDim 6196
ReadVariant 6197
RemoveObjectAsWriting 6197
ResetMultiRef 6197
SerializationOptions 6198
SerializationOptions method 6197
SetNodeType 6198
SetObjectPropFromText 6198
SetOptions 6199
TSOAPDomConv events 6202
TSOAPDomConv Members 6179
TSOAPDomConv methods 6181
TSOAPDomConv properties 6201
UnhandledNode 6199
WriteHeader 6199
WriteNonRectDynArray 6199
WriteNonRectDynArrayElem 6200
WriteRectDynArray 6200
WriteRectDynArrayElem 6200
WriteVarArray 6200
WriteVarArrayAsB64 6201
WriteVariant 6201
TSOAPHeader class 4448
about TSOAPHeader class 4448
Actor 4451
MustUnderstand 4452
ObjectToSOAP 4450
SOAPToObject 4451
TSOAPHeader Members 4449
TSOAPHeader methods 4450
TSOAPHeader properties 4451
TSOAPHeaders class 4452
about TSOAPHeaders class 4452
Get 4454
Get method 4454
GetOwnsSentHeaders 4455
Send 4455
SendAt 4455
SendCount 4456
SetOwnsSentHeaders 4456
TSOAPHeaders Members 4453
TSOAPHeaders methods 4454
TSOAPHeadersBase class 4457

about TSOAPHeadersBase class 4457
FHeadersInbound 4458
FHeadersOutBound 4458
SetHeadersInOut 4458
TSOAPHeadersBase fields 4457
TSOAPHeadersBase Members 4457
TSOAPHeadersBase methods 4458
TSplitter class 1492
about TSplitter class 1492
Align 1499
AutoSnap 1503
Beveled 1503
CanResize 1496
Canvas 1504
Color 1506
Constraints 1507
Create 1493
Cursor 1507
Destroy 1495
DoCanResize 1496
MinSize 1511
MouseDown 1497
MouseMove 1497
MouseUp 1497
OnCanResize 1517
OnMoved 1517
OnPaint 1520
Paint 1498
ParentColor 1512
RequestAlign 1498
ResizeStyle 1513
StopSizing 1499
TSplitter events 1517
TSplitter Members 1492
TSplitter methods 1495
TSplitter properties 1499
Visible 1514
Width 1515
TSQLResolver class 6564
about TSQLResolver class 6564
Create 6565
Destroy 6565
DoDelete 6566
DoExecSQL 6567
DoGetValues 6567
DoInsert 6567
DoUpdate 6568
FreeTreeData 6568
GenDeleteSQL 6568
GenInsertSQL 6569
GenSelectSQL 6569
GenUpdateSQL 6569
GenWhereSQL 6569
GetProvider 6570
InitializeConflictBuffer 6570
InitTreeData 6570
InternalDoUpdate 6570
Provider 6571
TSQLResolver Members 6564
TSQLResolver methods 6566
TSQLResolver properties 6571
UseFieldInUpdate 6571
UseFieldInWhere 6571
tssDark enumeration member 6939
tssHidden enumeration member 6939
tssLight enumeration member 6939
tssMedium enumeration member 6939
TStaticListAction class 4567
about TStaticListAction class 4567
Active 4569
Caption 4570
Enabled 4570
HelpContext 4571
Hint 4571
Images 4571
ItemIndex 4572
Items 4572
OnGetItem 4574

OnHint 4574
OnItemSelected 4574
OnUpdate 4575
SecondaryShortCuts 4572
ShortCut 4573
TStaticListAction events 4574
TStaticListAction Members 4568
TStaticListAction properties 4569
Visible 4573
TStaticListItems class 4576
about TStaticListItems class 4576
Notify 4577
TStaticListItems Members 4576
TStaticListItems methods 4577
Update 4578
TStatusColumn class 5481
about TStatusColumn class 5481
Caption 5484
CaptionAttributes 5484
Custom 5484
IsColumn 5483
Style 5485
StyleRule 5485
TStatusColumn Members 5482
TStatusColumn methods 5483
TStatusColumn properties 5483
TStringGrid class 3180
about TStringGrid class 3180
Cells 3188
Cols 3189
ColumnMoved 3186
Create 3185
Destroy 3186
DrawCell 3187
GetEditText 3187
Objects 3190
RowMoved 3187
Rows 3190
SetEditText 3188
TStringGrid Members 3181
TStringGrid methods 3186
TStringGrid properties 3188
TStringGridStrings class 3192
about TStringGridStrings class 3192
Add 3193
Assign 3194
Clear 3194
Create 3193
Delete 3195
Get 3195
GetCount 3195
GetObject 3196
Insert 3196
Put 3197
PutObject 3197
SetUpdateState 3197
TStringGridStrings Members 3192
TStringGridStrings methods 3193
TStringHash class 4324
about TStringHash class 4324
Add 4326
Clear 4326
Create 4325
Destroy 4325
Find 4326
HashOf 4327
Modify 4327
Remove 4327
TStringHash Members 4325
TStringHash methods 4326
ValueOf 4327
TSubmitQueryButton class 5486
about TSubmitQueryButton class 5486
Create 5487
EventContent 5488
TSubmitQueryButton Members 5486
TSubmitQueryButton methods 5487
TTabAction class 150

about TTabAction class 150
AfterTabChange 155
BeforeTabChange 155
Create 151
HandlesTarget 152
Notification 152
OnValidateTab 156
SelectNextTab 152
SkipHiddenTabs 153
TabControl 154
TTabAction events 155
TTabAction Members 150
TTabAction methods 151
TTabAction properties 153
UpdateTarget 153
Wrap 154
TTextAreaColumn class 5488
about TTextAreaColumn class 5488
Caption 5491
CaptionAttributes 5492
Custom 5492
DisplayRows 5493
DisplayWidth 5493
FieldName 5493
IsColumn 5491
ReadOnly 5494
Style 5494
StyleRule 5494
TabIndex 5495
TTextAreaColumn Members 5489
TTextAreaColumn methods 5490
TTextAreaColumn properties 5491
Wrap 5495
TTextColumn class 5495
about TTextColumn class 5495
Caption 5499
CaptionAttributes 5499
Custom 5500
DisplayWidth 5500
FieldName 5500
IsColumn 5498
ReadOnly 5501
Style 5501
StyleRule 5502
TabIndex 5502
TTextColumn Members 5496
TTextColumn methods 5498
TTextColumn properties 5498
TTimer class 1520
about TTimer class 1520
Create 1521
Destroy 1522
Enabled 1523
Interval 1525
OnTimer 1527
Timer 1522
TTimer events 1527
TTimer Members 1521
TTimer methods 1522
TTimer properties 1522
TTimerSpeedButton class 7267
about TTimerSpeedButton class 7267
ButtonStyle 7270
CMMouseenter 7268
CMMouseleave 7269
Create 7267
Destroy 7268
MouseDown 7269
MouseInControl 7271
MouseUp 7269
Paint 7270
TimeBtnState 7271
TTimerSpeedButton Members 7267
TTimerSpeedButton methods 7268
TTimerSpeedButton properties 7270
TTrayIcon class 1528
about TTrayIcon class 1528
Animate 1530

AnimateInterval 1530
BalloonFlags 1530
BalloonHint 1531
BalloonTimeout 1531
BalloonTitle 1531
Hint 1532
Icon 1532
IconIndex 1532
Icons 1532
OnAnimate 1533
OnBalloonClick 1534
OnClick 1534
OnDblClick 1534
OnMouseDown 1534
OnMouseMove 1535
OnMouseUp 1535
PopupMenu 1533
TTrayIcon events 1533
TTrayIcon Members 1528
TTrayIcon properties 1530
Visible 1533
ttSystem enumeration member 4168
ttView enumeration member 4168
TUndoButton class 5502
about TUndoButton class 5502
Create 5504
TUndoButton Members 5503
TUpdateTree class 6572
about TUpdateTree class 6572
Clear 6574
Create 6573
Data 6576
Delta 6576
Destroy 6573
DetailCount 6577
DetailList 6577
Details 6577
DoUpdates 6574
ErrorDS 6577
HasErrors 6578
InitData 6574
InitDelta 6575
InitDelta method 6575
InitErrorPacket 6575
IsNested 6578
Name 6578
Opened 6578
Parent 6578
RefreshData 6576
Resolver 6579
Source 6579
TUpdateTree Members 6572
TUpdateTree methods 6574
TUpdateTree properties 6576
TUserInfo class 4013
about TUserInfo class 4013
FirstName 4014
GroupID 4014
LastName 4014
MiddleName 4014
TUserInfo fields 4013
TUserInfo Members 4013
UserID 4015
UserName 4015
TVersionInfo class 4015
about TVersionInfo class 4015
ServerImplementation 4016
ServerVersion 4016
ServiceVersion 4016
TVersionInfo fields 4016
TVersionInfo Members 4016
TVirtualListAction class 4578
about TVirtualListAction class 4578
Active 4580
Caption 4581
Enabled 4581
HelpContext 4581
Hint 4582

Images 4582
ItemIndex 4582
OnGetItem 4584
OnGetItemCount 4584
OnHint 4585
OnItemSelected 4585
SecondaryShortCuts 4583
ShortCut 4583
TVirtualListAction events 4584
TVirtualListAction Members 4579
TVirtualListAction properties 4580
Visible 4584
TWebActionItem class 3335
about TWebActionItem class 3335
AssignTo 3337
Create 3336
Default 3339
Destroy 3337
Enabled 3340
GetDisplayName 3338
GetPathInfo 3338
MethodType 3341
Name 3341
OnAction 3343
PathInfo 3341
Producer 3342
ProducerContent 3342
SetDisplayName 3338
TWebActionItem events 3343
TWebActionItem Members 3336
TWebActionItem methods 3337
TWebActionItem properties 3339
TWebActionItems class 3344
about TWebActionItems class 3344
Add 3346
Create 3345
GetAttr 3346
GetAttrCount 3346
GetItemAttr 3347
GetOwner 3347
Items 3349
SetItemName 3348
TWebActionItems Members 3344
TWebActionItems methods 3345
TWebActionItems properties 3349
Update 3348
WebDispatcher 3349
TWebButton class 5504
about TWebButton class 5504
Caption 5508
Content 5506
Create 5506
Custom 5509
GetCaption 5507
GetLayoutAttributes 5507
ImplContent 5507
SetCaption 5508
Style 5509
StyleRule 5510
TWebButton Members 5505
TWebButton methods 5506
TWebButton properties 5508
TWebControlGroup class 5510
about TWebControlGroup class 5510
AddElements 5513
CanAddClass 5513
ChangeFieldControls 5514
Content 5514
Create 5512
DefaultWebComponents 5518
Destroy 5512
GetDefaultWebComponents 5515
GetLayoutAttributes 5515
GetSubComponents 5516
ImplAddElements 5516
ImplCanAddClass 5516
ImplContent 5517
ImplGetSubComponents 5517

SetDefaultWebComponents 5518
TWebControlGroup Members 5511
TWebControlGroup methods 5512
TWebControlGroup properties 5518
WebFieldControls 5519
TWebDataDisplay class 5519
about TWebDataDisplay class 5519
Caption 5529
CaptionAttributes 5530
CaptionPosition 5530
Content 5523
ControlContent 5523
Create 5521
Custom 5530
Destroy 5522
EventContent 5523
FormatCaption 5524
GetCaption 5524
GetHTMLControlName 5524
GetHTMLForm 5524
GetLayoutAttributes 5525
GetXmlDisplayName 5525
GetXmlRowSetName 5526
ImplContent 5526
ImplGetHTMLControlName 5527
IsColumn 5527
IsQueryField 5528
LabelContent 5528
LayoutAttributes 5531
SetCaption 5528
SetCaptionAttributes 5529
Style 5531
StyleRule 5532
TabIndex 5532
TWebDataDisplay Members 5520
TWebDataDisplay methods 5522
TWebDataDisplay properties 5529
TWebDataInput class 5532
about TWebDataInput class 5532
EventContent 5535
FieldName 5539
FindAssociatedField 5535
GetCaption 5536
GetFieldName 5536
GetParamName 5536
GetRowSetFieldAttributes 5536
ImplGetFieldName 5537
ImplGetHTMLControlName 5537
ImplGetRowSetFieldAttributes 5537
ImplRestoreDefaults 5537
ParamName 5540
RestoreDefaults 5538
RestoreFieldDefaults 5538
SetCaption 5538
SetFieldName 5538
SetParamName 5539
TWebDataInput Members 5533
TWebDataInput methods 5535
TWebDataInput properties 5539
ValidateField 5539
TWebDispatcher class 3349
about TWebDispatcher class 3349
Actions 3351
AfterDispatch 3352
BeforeDispatch 3353
OnException 3353
TWebDispatcher events 3352
TWebDispatcher Members 3350
TWebDispatcher properties 3351
TWebDisplay class 5540
about TWebDisplay class 5540
Changed 5542
ChangedDisplayComponent 5542
Create 5541
DisplayComponent 5543
GetDisplayComponent 5542
Notification 5542
OnChange 5543

Parent 5543
SetDisplayComponent 5542
TWebDisplay events 5543
TWebDisplay Members 5541
TWebDisplay methods 5541
TWebDisplay properties 5543
TWebDisplayReferenceGroup class 5544
about TWebDisplayReferenceGroup class 5544
Create 5545
Destroy 5546
GetWebDisplayComponent 5546
Notification 5546
SetWebDisplayComponent 5547
TWebDisplayReferenceGroup Members 5544
TWebDisplayReferenceGroup methods 5546
TWebDisplayReferenceGroup properties 5547
WebDisplay 5547
WebDisplayComponent 5547
TWebForm class 5548
about TWebForm class 5548
AddElements 5550
CanAddClass 5551
Content 5551
Create 5549
Custom 5556
Destroy 5550
GetDefaultWebComponents 5552
GetHTMLFormName 5552
GetHTMLFormVarName 5553
GetLayoutAttributes 5553
GetSubComponents 5553
GetVisibleFields 5554
ImplAddElements 5554
ImplCanAddClass 5554
ImplContent 5555
SetDefaultWebComponents 5555
Style 5556
StyleRule 5557
TWebForm Members 5549
TWebForm methods 5550
TWebForm properties 5556
VisibleFields 5557
WebComponents 5558
TWebListInput class 5558
about TWebListInput class 5558
AddElements 5562
ControlContent 5562
Create 5561
DataSet 5564
Destroy 5561
FormatInputs 5562
GetSubComponents 5563
ImplAddElements 5563
Items 5565
ItemsField 5565
Notification 5563
SetDataSet 5564
SetItems 5564
SetValues 5564
TWebListInput Members 5559
TWebListInput methods 5562
TWebListInput properties 5564
Values 5566
ValuesField 5566
TWebModule class 3354
about TWebModule class 3354
Actions 3356
AfterDispatch 3357
BeforeDispatch 3357
Create 3356
OnException 3358
TWebModule events 3357
TWebModule Members 3355
TWebModule properties 3356
TWebRadioGroupInput class 5567
about TWebRadioGroupInput class 5567
AddAttributes 5570
Create 5570

DisplayColumns 5572
DisplayWidth 5572
EventContent 5571
FormatInputs 5571
GetCheckIndex 5571
ImplGetRowSetFieldAttributes 5571
ReadOnly 5573
RestoreFieldDefaults 5571
TWebRadioGroupInput Members 5568
TWebRadioGroupInput methods 5570
TWebRadioGroupInput properties 5572
TWebRequest class 3358
about TWebRequest class 3358
Accept 3371
Authorization 3372
CacheControl 3372
Connection 3373
Content 3373
ContentEncoding 3374
ContentFields 3375
ContentLength 3375
ContentParser 3376
ContentType 3376
ContentVersion 3376
Cookie 3377
CookieFields 3377
Create 3361
Date 3378
DerivedFrom 3378
Destroy 3361
Expires 3379
ExtractContentFields 3363
ExtractCookieFields 3363
ExtractFields 3364
ExtractFields method 3363
ExtractQueryFields 3364
Files 3379
From 3379
GetDateVariable 3365
GetFieldByName 3365
GetIntegerVariable 3366
GetInternalPathInfo 3366
GetInternalScriptName 3366
GetStringVariable 3366
Host 3380
IfModifiedSince 3380
InternalPathInfo 3381
InternalScriptName 3381
Method 3382
MethodType 3382
PathInfo 3382
PathTranslated 3383
ProtocolVersion 3383
Query 3384
QueryFields 3384
RawContent 3385
ReadClient 3367
ReadString 3368
ReadUnicodeString 3368
Referer 3385
RemoteAddr 3386
RemoteHost 3386
ScriptName 3386
ServerPort 3387
Title 3387
TranslateURI 3368
TWebRequest Members 3359
TWebRequest methods 3362
TWebRequest properties 3370
UpdateMethodType 3369
URL 3388
UserAgent 3388
WriteClient 3369
WriteHeaders 3370
WriteString 3370
TWebRequestFile class 6696
about TWebRequestFile class 6696
Create 6697

Destroy 6698
GetContentType 6698
GetFieldName 6699
GetFileName 6699
GetStream 6699
TWebRequestFile Members 6697
TWebRequestFile methods 6698
TWebRequestFiles class 6700
about TWebRequestFiles class 6700
Add 6702
Add method 6701
Create 6701
Destroy 6701
GetCount 6703
GetItem 6703
TWebRequestFiles Members 6700
TWebRequestFiles methods 6701
TWebRequestFileStream class 6703
about TWebRequestFileStream class 6703
Create 6704
TWebRequestFileStream Members 6704
TWebRequestFileStream methods 6704
Write 6704
TWebResponse class 3389
about TWebResponse class 3389
AddCustomHeaders 3394
Allow 3404
Content 3404
ContentEncoding 3405
ContentLength 3405
ContentStream 3406
ContentType 3406
ContentVersion 3407
Cookies 3407
Create 3391
CustomHeaders 3408
Date 3408
DerivedFrom 3409
Destroy 3392
Expires 3409
FCustomHeaders 3392
FHTTPRequest 3393
FormatAuthenticate 3394
GetContent 3395
GetCustomHeader 3395
GetDateVariable 3395
GetIntegerVariable 3396
GetLogMessage 3396
GetStatusCode 3396
GetStringVariable 3397
HTTPRequest 3410
LastModified 3410
Location 3411
LogMessage 3411
RawContent 3411
Realm 3411
ReasonString 3412
SendRedirect 3398
SendResponse 3398
SendStream 3398
Sent 3399
Server 3412
SetContent 3399
SetContentStream 3400
SetCookieField 3400
SetCustomHeader 3400
SetDateVariable 3401
SetIntegerVariable 3401
SetLogMessage 3402
SetStatusCode 3402
SetStringVariable 3402
StatusCode 3413
Title 3414
TWebResponse fields 3392
TWebResponse Members 3390
TWebResponse methods 3393
TWebResponse properties 3403
Version 3415

WWWAuthenticate 3415
TWebSelectOptionsInput class 5573
about TWebSelectOptionsInput class 5573
AddAttributes 5576
Create 5576
DisplayRows 5577
EventContent 5577
FormatInputs 5577
GetSelectIndex 5577
TWebSelectOptionsInput Members 5574
TWebSelectOptionsInput methods 5576
TWebSelectOptionsInput properties 5577
TWebStatus class 5578
about TWebStatus class 5578
Caption 5582
CaptionAttributes 5582
CaptionPosition 5583
ControlContent 5580
Create 5580
DisplayWidth 5583
GetCaption 5581
Identifier 5581
ImplGetHTMLControlName 5581
TWebStatus Members 5579
TWebStatus methods 5580
TWebStatus properties 5582
TWebTextAreaInput class 5583
about TWebTextAreaInput class 5583
AddAttributes 5586
ControlContent 5586
Create 5586
DisplayRows 5588
DisplayWidth 5588
EventContent 5587
ImplGetRowSetFieldAttributes 5587
ReadOnly 5588
RestoreFieldDefaults 5587
TWebTextAreaInput Members 5584
TWebTextAreaInput methods 5586
TWebTextAreaInput properties 5587
Wrap 5588
TWebTextInput class 5589
about TWebTextInput class 5589
AddAttributes 5592
ControlContent 5592
Create 5592
Custom 5593
DisplayWidth 5594
ImplGetRowSetFieldAttributes 5593
ReadOnly 5594
RestoreFieldDefaults 5593
Style 5594
StyleRule 5595
TWebTextInput Members 5590
TWebTextInput methods 5592
TWebTextInput properties 5593
TXMLButton class 5595
about TXMLButton class 5595
Create 5596
DefaultCaption 5599
Destroy 5597
GetCaption 5597
GetXMLBroker 5598
GetXMLUseParent 5598
Notification 5598
SetXMLBroker 5599
SetXMLUseParent 5599
TXMLButton Members 5596
TXMLButton methods 5597
TXMLButton properties 5599
XMLBroker 5599
XMLData 5600
XMLUseParent 5600
TXMLData class 5601
about TXMLData class 5601
Changed 5602
ChangedXMLBroker 5603
Create 5602

GetXMLBroker 5603
Notification 5603
OnChange 5605
Parent 5604
SetXMLBroker 5603
TXMLData events 5604
TXMLData Members 5601
TXMLData methods 5602
TXMLData properties 5604
XMLBroker 5604
TXMLDataParent class 5605
about TXMLDataParent class 5605
ChangedUseParent 5606
ChangedXMLBroker 5606
GetXMLBroker 5607
ParentXMLBroker 5607
SetUseParent 5607
TXMLDataParent Members 5605
TXMLDataParent methods 5606
TXMLDataParent properties 5607
UseParent 5608
TXMLDataSet class 5608
_AddRef 5610
_Release 5611
about TXMLDataSet class 5608
ChangedDataSetField 5611
ChangedXMLBroker 5611
CompareXMLData 5611
ConnectionChange 5611
Create 5609
CreateDataSet 5612
DataSet 5614
DataSetField 5614
DataSetPath 5615
Destroy 5610
FindDataSet 5612
GetDataSet 5612
GetDataSetField 5612
GetDataSetPath 5613
Parent 5615
QueryInterface 5613
SetDataSetField 5613
SetXMLBroker 5613
TXMLDataSet Members 5608
TXMLDataSet methods 5610
TXMLDataSet properties 5614
WrapperDataChange 5613
TXMLDataSetParent class 5616
about TXMLDataSetParent class 5616
ChangedDataSetField 5617
ChangedUseParent 5618
ChangedXMLBroker 5618
GetDataSetField 5618
GetXMLBroker 5618
ParentDataSetField 5618
ParentXMLBroker 5619
ParentXMLDataSet 5619
SetUseParent 5619
TXMLDataSetParent Members 5616
TXMLDataSetParent methods 5617
TXMLDataSetParent properties 5619
UseParent 5619
TXMLDisplayGroup class 5620
about TXMLDisplayGroup class 5620
Create 5622
CreateDefaultFields 5623
Destroy 5622
EnableValidateFields 5630
FindField 5623
GetDataSet 5624
GetDataSetField 5624
GetEnableValidateFields 5624
GetFieldsList 5624
GetIsMultipleRecordView 5625
GetVisibleFields 5625
GetXMLBroker 5625
GetXMLDataSet 5625
GetXMLDisplayName 5626

GetXMLRowSetName 5626
HasStatusField 5626
ImplAddElements 5626
ImplGetSubComponents 5627
ImplIsMultipleRecordView 5627
IsFieldInUse 5627
Notification 5628
SetDataSetField 5628
SetEnableValidateFields 5628
SetXMLBroker 5629
TXMLDisplayGroup Members 5621
TXMLDisplayGroup methods 5623
TXMLDisplayGroup properties 5629
ValidateDataSet 5629
ValidateFields 5629
VisibleFields 5630
XMLBroker 5631
XMLDataChange 5629
XMLDataSet 5631
XMLDataSetField 5631
TXMLDisplayParent class 5632
about TXMLDisplayParent class 5632
ChangedDisplayComponent 5633
ChangedUseParent 5634
Create 5633
GetDisplayComponent 5634
SetUseParent 5634
TXMLDisplayParent Members 5632
TXMLDisplayParent methods 5633
TXMLDisplayParent properties 5634
UseParent 5634
TXMLDisplayReferenceButton class 5635
about TXMLDisplayReferenceButton class 5635
Create 5637
Destroy 5637
GetDisplayComponent 5637
GetDisplayComponentParent 5638
GetXmlDisplayName 5638
Notification 5638
SetDisplayComponent 5639
SetDisplayComponentParent 5639
TXMLDisplayReferenceButton Members 5636
TXMLDisplayReferenceButton methods 5637
TXMLDisplayReferenceButton properties 5639
XMLComponent 5639
XMLDisplay 5640
XMLUseParent 5640
TXMLDisplayReferenceGroup class 5640
about TXMLDisplayReferenceGroup class 5640
ImplAddElements 5642
TXMLDisplayReferenceGroup Members 5641
TXMLDisplayReferenceGroup methods 5642
TXMLDisplayReferenceGroup properties 5643
XMLComponent 5643

U

uaAbort enumeration member 3691
uaApplied enumeration member 3691
uaApply enumeration member 3691
uaFail enumeration member 3691
uaRetry enumeration member 3691
uaSkip enumeration member 3691
UnknownAdvise enumeration member 4024
UnknownState enumeration member 4025
UseAllSpace enumeration member 4021

V

ValidateDB enumeration member 4026
ValidateFull enumeration member 4026
Version enumeration member 4020

W

wrAbort enumeration member 3872
wrContinue enumeration member 3872
wrOff enumeration member 5683
wrPhysical enumeration member 5683
wrVirtual enumeration member 5683
wsMaximized enumeration member 2594

wsMinimized enumeration member 2594
wsNormal enumeration member 2594

X

xoAttributeOnLastMember enumeration member 4481
xoHolderClass enumeration member 4481
xoInlineArrays enumeration member 4481
xoLiteralParam enumeration member 4481
xoOption6 enumeration member 4481
xoOption7 enumeration member 4481
xoOption8 enumeration member 4481
xoOption9 enumeration member 4481
xoOptionA enumeration member 4481
xoOptionB enumeration member 4481
xoOptionC enumeration member 4481
xoOptionD enumeration member 4481
xoOptionE enumeration member 4481
xoOptionF enumeration member 4481
xoOptionG enumeration member 4481
xoOptionH enumeration member 4481
xoOptionI enumeration member 4481
xoSimpleTypeWrapper enumeration member 4481